

Dieter Mergel

Physics with Excel and Python

Using the Same Data Structure
Volume I: Basics, Exercises and Tasks

 Springer

Physics with Excel and Python

Dieter Mergel

Physics with Excel and Python

Using the Same Data Structure
Volume I: Basics, Exercises and Tasks

 Springer

Dieter Mergel
Fakultät für Physik
Universität Duisburg-Essen
Duisburg, Nordrhein-Westfalen, Germany

ISBN 978-3-030-82324-5 ISBN 978-3-030-82325-2 (eBook)
<https://doi.org/10.1007/978-3-030-82325-2>

Based on the German language edition: *Physik mit Excel und Visual Basic* by Dieter Mergel, © 2017 2017. Published by Springer-Spectrum. All Rights Reserved, and extended with Python solutions. © Springer Nature Switzerland AG 2022

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Ψ EXCEL—as powerful as necessary, Python—as simple as possible.

This book treats a series of physics exercises that were developed for courses at the University of Duisburg-Essen for students training to become physics/mathematics teachers or physics engineers. The exercises were intended to introduce computational physics based on spreadsheet calculation combined with simple VBA macros and also to broaden the beginner's knowledge of physics. This approach garnered positive reactions from practitioners and resulted in a textbook in German.¹ Furthermore, the methods developed turned out to be powerful enough to treat a broad range of topics in undergraduate physics, resulting in a second volume with exercises on particles, waves, fields, and random processes.²

Referees found the exercises interesting and ambitious enough to serve for undergraduate education. However, concerns arose that spreadsheet techniques, although useful in the business world, might be a dead end for students who would be required to use scientific computing in their future research work. Therefore, the concept has been changed for the present English version. Programming in Python is now included from the beginning, while the same topics are addressed as in the German textbook.

The key to all of the exercises is data structures, developed in introductory sections that explain the physical problems. They serve as an interface to both EXCEL and Python, and potentially also to other applications for scientific computation. To enable this approach, the EXCEL solutions in this edition use vectorized code and matrix formulas to mimic broadcasting, an essential Python technique for creating new arrays.

We feel that this approach is suitable as a low-threshold introduction to scientific computing as early as high school all the way up to undergraduate physics

¹ Dieter Mergel, Physik mit Excel und Visual Basic Grundlagen, Beispiele und Aufgaben, Springer Spektrum (2017), <https://doi.org/10.1007/978-3-642-37857-7>.

² Dieter Mergel, Physik lernen mit Excel und Visual Basic, Anwendungen auf Teilchen, Wellen, Felder und Zufallsprozesse, Springer Spektrum (2018), <https://doi.org/10.1007/978-3-662-57513-0>.

classes at the university and may also be a good start for students who later choose to specialize in computational physics.

Our approach is intended to make the student fit for a computer-oriented world, be it for spreadsheet calculations in business, scientific computing in research, or mathematics and physics teaching in high school. We take into account that not all students have the same attitude towards programming; some have to be encouraged to venture into a new world, whereas others have to be cautioned not to rush into blind programming.

Duisburg, Germany

Dieter Mergel

Contents

1	Introduction	1
1.1	A Two-Track Didactical Approach	1
1.2	What Can You Expect?	2
1.3	What Do You Need?	3
1.4	Tim, Alac, and Mag	3
1.5	Didactic Concept	5
1.6	Subject Matter	7
1.7	Getting Started with Excel	8
1.7.1	Start Menu	8
1.7.2	Spreadsheet Presentation	9
1.8	Getting Started with Python	10
1.9	Skills to Be Trained	11
2	Data Structures, Excel and Python Basics	15
2.1	Introduction: Named Ranges in Excel, Arrays in Numpy	15
2.2	Characteristics of a Parabola	17
2.2.1	Different Definitions of a Parabola	17
2.2.2	Data Structure and Nomenclature	19
2.3	Basic Exercise in Spreadsheet Calculation	19
2.3.1	Cell Addressing	19
2.3.2	Graphical Representation of a Function	22
2.3.3	Smart Legends in Figures	24
2.3.4	Scroll Bars	24
2.3.5	Summary: Cell References and Name Manager	26
2.3.6	What Have We Learned so Far, and How to Proceed Further?	30
2.3.7	Python Program	30
2.4	Python and NumPy Basics	31
2.4.1	Basic Exercise	31
2.4.2	Data Structures	34
2.4.3	Python Libraries	35
2.4.4	Numpy Constructions	36

2.4.5	Standard Plot Program	39
2.4.6	Formatted Output	41
2.5	Matrix Calculations in Excel and Python	41
2.5.1	Data Structure and Nomenclature	41
2.5.2	Operations on Arrays	42
2.5.3	Matrices in Spreadsheets	43
2.5.4	Matrices in Python	45
2.6	Four Parabolas and Their Upper Envelope	49
2.6.1	Graphical Representation	50
2.6.2	Data Structure and Nomenclature	51
2.6.3	Spreadsheet Calculation	51
2.6.4	Python Program	52
2.6.5	Extrema Along Different Axes	56
2.7	Sum of Four Cosine Functions	57
2.7.1	Sound and a Cosine Identity	57
2.7.2	Data Structure and Nomenclature	62
2.7.3	Spreadsheet Layout	62
2.7.4	Python Program	64
2.7.5	Producing Labels (as Strings) in Excel and Python	68
2.8	Questions	69
3	Formula Networks and Linked Diagrams	75
3.1	Introduction: Well-Structured Sheets and Programs	75
3.2	Image Construction for Focusing and Diverging Lenses	78
3.2.1	Straight Line Equation	78
3.2.2	Geometrical Image Construction for a Thin Focusing Lens	80
3.2.3	Imaging Equation with Correct Signs	82
3.2.4	Beam Through a Converging Lens that Really Contributes to the Image	83
3.2.5	Data Structure and Nomenclature	86
3.2.6	Spreadsheet Calculation	86
3.2.7	Python Program	88
3.3	Doppler Effect	92
3.3.1	A Formula for All Cases	92
3.3.2	A Sound Source Passes a Remote Receiver	95
3.3.3	Data Structure and Nomenclature	96
3.3.4	Spreadsheet Calculation “Remote Receiver”	97
3.3.5	Python Program “Remote Receiver”	97
3.4	Exponentials	99
3.4.1	Explosive Character of Exponentials	99
3.4.2	General Exponential Function	101
3.4.3	Representation in a Diagram	102
3.4.4	Diode Characteristics $I(U)$	103
3.4.5	Data Structure and Nomenclature	104

3.4.6	Spreadsheet Calculation	105
3.4.7	Python Program	107
3.5	Questions	109
4	Macros with Visual Basic and Their Correspondences in Python ...	115
4.1	Introduction: For, If, Sub/Def	115
4.2	Basic Exercise: FOR- Loops	118
4.2.1	Visual-Basic-Editor 1: Editing	118
4.2.2	Programming	119
4.3	Macro-Controlled Drawings with FOR, SUB, IF	123
4.3.1	Macro Recorder	124
4.3.2	Visual-Basic Editor 2: Macro Recording, Debugging	127
4.3.3	Programming Elements	128
4.4	A Checkerboard Pattern (Excel)	132
4.4.1	Checkerboard, Same-Colored and Multi-colored	132
4.4.2	Global Variables	134
4.5	A Checkerboard Pattern (Python)	135
4.5.1	Turtle	135
4.5.2	Differences to Visual Basic	137
4.5.3	Checkerboard with Squares, Triangles, and Circles	139
4.6	Drawing Densely-Packed Atomic Layers; Crystal Physics	143
4.6.1	Program Structure and Geometry	143
4.6.2	Data Structure and Nomenclature	146
4.6.3	Excel	146
4.6.4	Python	148
4.7	Text Processing	151
4.7.1	Cutting and Joining Strings	151
4.7.2	Data Structure and Program Flow	153
4.7.3	Excel	154
4.7.4	Programming Step by Step	156
4.7.5	VBA Constructs	157
4.7.6	Python	157
4.8	Processing the Protocol of a Measuring Device	160
4.8.1	Protocol of a Measuring Device	161
4.8.2	Detection of Code Words	162
4.8.3	Data Structure and Nomenclature	162
4.8.4	Excel	163
4.8.5	Python	165
4.9	User-Defined Functions	168
4.9.1	User-Defined Functions as Add-In	168
4.9.2	Scalar Product and Vector Product	170
4.9.3	Python	173
4.10	Questions and Tasks	174

5	Basic Mathematical Techniques	181
5.1	Introduction: Calculus, Vectors, and Linear Algebra	181
5.2	Straight-Line Segment Under a Magnifying Glass	182
5.2.1	Under a Magnifying Glass	182
5.2.2	Data Structure and Nomenclature	183
5.2.3	Spreadsheet Calculation	184
5.2.4	Plotting Vectors with Python Matplotlib	185
5.3	Differentiation	185
5.3.1	First and Second Derivative	186
5.3.2	Data Structure and Nomenclature	190
5.3.3	Spreadsheet Layout	190
5.3.4	Python Program	191
5.4	Integration	192
5.4.1	Area Under a Curve	192
5.4.2	Length of a Curve	194
5.4.3	Data Structure and Nomenclature for the Arrays in the Integration	194
5.4.4	Python Program	195
5.4.5	Spreadsheet Solution	197
5.5	Vectors in the Plane	198
5.5.1	Vectors	198
5.5.2	Data Structure and Nomenclature	200
5.5.3	Spreadsheet Layout	200
5.5.4	Python Program	202
5.6	Tangents to and Perpendiculars on a Curve	204
5.6.1	At/On a Polynomial and an Ellipse	204
5.6.2	Data Structure and Nomenclature	205
5.6.3	Python Program	205
5.6.4	Spreadsheet Solution	206
5.7	Banked Curve	207
5.7.1	Cross-Section of the Road	207
5.7.2	Data Structure and Nomenclature	209
5.7.3	Python Program	209
5.7.4	Spreadsheet Solution	211
5.8	Weighted Average	211
5.8.1	A Mobile with Two Arms	212
5.8.2	Data Structure and Nomenclature	213
5.8.3	Python Program	213
5.8.4	Spreadsheet Calculation	214
5.9	Systems of Linear Equations	215
5.9.1	Polynomial and Electrical Network	215
5.9.2	Data Structure and Nomenclature	218
5.9.3	Spreadsheet Solutions	219
5.9.4	Python Programs	220

5.10	Some Mathematical Functions	222
5.11	Questions and Tasks	224
6	Superposition of Movements	227
6.1	Introduction: Translations and Rotations	227
6.2	Projectile Trajectory with Velocity Vectors (T-T)	229
6.2.1	Projectile Trajectory and Velocity Vectors	229
6.2.2	Data Structure and Nomenclature	230
6.2.3	Spreadsheet	231
6.2.4	Python	233
6.2.5	Animation of Figures with FuncAnimation	234
6.3	Cycloid, Rolling Curve (R-T)	238
6.3.1	Trace of a Writing Point Fixed at a Rolling Wheel	238
6.3.2	Data Structure and Nomenclature	241
6.3.3	Excel	242
6.3.4	Python	244
6.4	Foucault's Pendulum (T-R)	246
6.4.1	A Lecture Experiment	246
6.4.2	Data Structure and Nomenclature	248
6.4.3	Excel	248
6.4.4	Python	250
6.5	Anchor, Deflected Out of Its Rest Position (R-R)	251
6.5.1	Deflected Anchor	251
6.5.2	Data Structure and Nomenclature	254
6.5.3	Excel	255
6.5.4	Python	256
6.6	Wavefronts, Sound Barriers, and Mach Cone (T-T)	258
6.6.1	Emitting Sound Waves	259
6.6.2	Data Structure and Nomenclature	261
6.6.3	Spreadsheet Solution	261
6.6.4	Python	264
6.7	Questions and Tasks	265
7	Integration of Newton's Equation of Motion	269
7.1	Introduction: Approximated Mean Value Instead of Exact Integration	269
7.1.1	Newton's Equation of Motion	269
7.1.2	Four Methods for Estimating the Average Acceleration in a Time Segment	271
7.1.3	Tactical Approaches in Python and Excel	272
7.2	Harmonic Oscillation with "Progress with Look-Ahead" and "Runge-Kutta"	272
7.2.1	Equation of Motion	273
7.2.2	Data Structure and Nomenclature	275
7.2.3	Spreadsheet Calculation	275
7.2.4	Python	276

7.3	Falling from a (Not Too) Great Height	278
7.3.1	Limiting Cases, Analytically Solved	279
7.3.2	Data Structure and Nomenclature	280
7.3.3	Spreadsheet	280
7.3.4	Python	282
7.4	Stratospheric Jump	283
7.4.1	Data Structure and Nomenclature	285
7.4.2	Spreadsheet Calculation	285
7.4.3	Python	286
7.5	A Car Drives with Variable Power	286
7.5.1	Various Types of Power	287
7.5.2	Data Structure and Nomenclature	291
7.5.3	Excel	291
7.5.4	Python	295
7.6	Bungee Jump	298
7.6.1	Simulation of the Motion	298
7.6.2	Analytical Calculations	300
7.6.3	Data Structure and Nomenclature	302
7.6.4	Excel	302
7.6.5	Python	304
7.7	Questions and Tasks	304
8	Random Numbers and Statistical Reasoning	307
8.1	Introduction: Statistical Experiments Instead of Theoretical Derivations	307
8.2	Equi-Distributed Random Numbers, Frequencies of Occurrence, χ^2 Test	310
8.2.1	A Spreadsheet Experiment with Random Numbers	310
8.2.2	Data Structure and Nomenclature	315
8.2.3	Python	315
8.3	Points Randomly Distributed in a Unit Square	318
8.3.1	Creation and Distribution of the Points	318
8.3.2	Data Structure and Nomenclature	321
8.3.3	Excel	321
8.3.4	Python	324
8.3.5	Why Calculate Twice?	326
8.4	Set Operations in Numpy	327
8.4.1	Sets	327
8.4.2	Data Structure and Nomenclature	330
8.4.3	Python	330
8.5	Normally Distributed Random Numbers	332
8.5.1	Normal Distribution, Probability Density and Distribution Function	333
8.5.2	Random-Number Generator and Frequencies of Occurrence	336

8.5.3	Where Do Observed and Theoretical Frequencies Fit Better Together?	337
8.5.4	Data Structure and Nomenclature	340
8.5.5	Python	341
8.5.6	Excel	342
8.6	Random-Number Generator, General Principle	345
8.7	Diffraction of Photons at a Double-Slit	348
8.7.1	Physical Background: Wave-Particle Dualism	348
8.7.2	Cos^2 Distribution	350
8.7.3	Data Structure and Nomenclature	351
8.7.4	Python	351
8.7.5	Excel	353
8.7.6	Simulation in a Spreadsheet	355
8.8	Chi^2 Distribution and Degrees of Freedom	358
8.8.1	Data Structure, Nomenclature	359
8.8.2	Python	360
8.9	Questions and Tasks	361
9	Evaluation of Measurements	365
9.1	Introduction: We Know Everything and Play Stupid	365
9.2	Weighing a Glass Substrate	369
9.2.1	Discussion on the Accuracy of a Balance	369
9.2.2	Data Structure and Nomenclature	370
9.2.3	Excel	370
9.2.4	Python	371
9.3	A Procedure for Rounding to Relevant Digits	372
9.3.1	Numerical Evaluations	372
9.3.2	Spreadsheet Calculation	372
9.3.3	Python Function	373
9.3.4	VBA Function	373
9.4	Increasing the Measuring Accuracy Through Repetition	374
9.4.1	Standard Deviation and Standard Error of the Mean Value of a Measurement Series	375
9.4.2	Data Structure and Nomenclature	379
9.4.3	Python Program	379
9.4.4	Spreadsheet Layout for This Task	380
9.4.5	How to Report a Measurement Result	381
9.5	The t Statistics Connects Confidence Interval with Confidence Level	382
9.5.1	Student's t Distribution	382
9.5.2	Data Structure and Nomenclature	386
9.5.3	Spreadsheet Calculation	386
9.5.4	Python Program	388
9.6	Combining Results from Several Measurement Series	389
9.6.1	Combining Two Measurement Results	390

9.6.2	Data Structure and Nomenclature	393
9.6.3	Spreadsheet Calculation	394
9.6.4	Python, Internally and Externally Consistent Error of the Combined Result	395
9.7	Propagation of Standard Deviations	397
9.7.1	Rules for Propagation of Standard Deviations	397
9.7.2	Data Structure and Nomenclature	402
9.7.3	Spreadsheet Calculation	402
9.7.4	Python Program	404
9.8	Propagation of Confidence Intervals	407
9.8.1	From Variance to Confidence	407
9.8.2	Sum and Product of Two Measurands	408
9.9	Mass of a Thin Film on a Glass Substrate	409
9.9.1	Instructions for Use for Accurate Measurements and Their Results	410
9.9.2	Data Structure and Nomenclature	413
9.9.3	Spreadsheet Solution	413
9.9.4	Python Program	415
9.10	Questions and Tasks	417
10	Fitting Trend Curves to Data Points	419
10.1	Introduction: Linear and Nonlinear Regression	419
10.1.1	Straight Line Through Data Points by Sight	419
10.1.2	Multilinear Regression	420
10.1.3	Nonlinear Regression	421
10.1.4	Coefficient of Determination R^2	422
10.1.5	C-spec Error with Iterative t Adaptation	423
10.2	Linear Trend Line	424
10.2.1	Creating Data Points and Evaluating Them	424
10.2.2	Data Structure and Nomenclature	426
10.2.3	Spreadsheet Calculation with Linest	426
10.2.4	Python Program	429
10.3	Fitting a Polynomial Trend Line to Data Points with Multilinear Regression	431
10.3.1	Introduction	431
10.3.2	Data Structure and Nomenclature	435
10.3.3	Spreadsheet Solution	436
10.3.4	Python Solution	437
10.4	Exponential Trend Line	440
10.4.1	Exponential and Logarithm	440
10.4.2	Exponential or Polynomial?	444
10.4.3	Data Structure and Nomenclature	445
10.4.4	Python Program	445
10.4.5	Spreadsheet Solution	447

10.5	Solving Nonlinear Equations	448
10.5.1	Intersection of Straight Lines with a Parabola	448
10.5.2	Data Structure and Nomenclature	449
10.5.3	Spreadsheet Calculation	450
10.5.4	Python Program	455
10.6	Temperature Dependence of the Saturation Magnetization of a Ferromagnet	456
10.6.1	Langevin Function	456
10.6.2	Data Structure and Nomenclature	458
10.6.3	Spreadsheet Layout	458
10.6.4	Python	459
10.7	Fitting Gaussians to Spectral Lines with Nonlinear Regression	460
10.7.1	Fitting the Sum of Two Gaussians to Data Points	460
10.7.2	C-spec Errors of the Coefficients by a Statistical Simulation	463
10.7.3	Data Structure and Nomenclature	464
10.7.4	Python	465
10.7.5	Spreadsheet	468
10.7.6	C-spec Error of the Optimized Coefficients by Simulation-Based t Adaptation	470
10.8	Questions and Tasks	472
Index	475

About the Author

Dieter Mergel studied physics in Göttingen, obtained his doctorate at the Technical University of Clausthal in the field of solid-state physics, and worked 11 years in the Philips Research Laboratories Hamburg/Aachen on automatic speech recognition and optical data storage. Since 1993, he is Professor of Technical Physics at the University of Duisburg-Essen. His professional activities include research in the field of solid-state layers and lectures for students in teaching and medical professions.



Possible errata and corrections in the internet at: go.sn.pub/Ob4vCR.

For every chapter, solutions for each two exercises in Excel and Python can be found at internet addresses.

1.1 A Two-Track Didactical Approach

History

The exercises in this book arise from a German textbook that emerged from courses for prospective teachers and students of Technical Physics at the University of Duisburg-Essen with the intention to prepare the students for a computerized world. The participants in the courses had already been studying physics for at least one year. However, the explanations of the exercises are so explicit that they should also be suitable for beginners.

Said courses are based on EXCEL and Visual Basic (VBA). The current English version includes Python from the very beginning so as to make it more generally useful for students who later choose to dive deeper into Scientific Computation.

Exercises

The subject matter is presented in nine chapters as a series of exercises. Every exercise consists of three steps:

1. The physical concept is introduced with mathematical equations and diagrams.
2. An adequate data structure is set up independent of the implementation in a particular programming platform, but taking care that the same nomenclature can be used in both mathematical equations and programming. This serves as an interface to any programming application.
3. Solutions in EXCEL and Python are designed so that a solution in one application can directly be translated into the other one.

To enable this approach, training in EXCEL emphasizes vectorized code, matrix formulas, and constructs that allow for broadcasting in the same way as Python. Furthermore, programming VBA macros interacting with spreadsheets introduces looping, logical queries, and functions.

Didactical advantages of the two-track system

We strive to combine the didactical advantages of both programming applications:

- Spreadsheets are interactive; charts react immediately to changes, and data structures (but not formulas) are immediately visible in spreadsheets.
- In Python, formulas (but not data structures) are immediately visible.
- The VBA interpreter allows us to run the program step by step and to watch intermediate results in a spreadsheet, or read the content of variables by mouse-over in the code.

Our guideline: spreadsheet calculations as powerful as necessary, Python programming as simple as possible.

Emphasis is not on mere computational techniques, but on exercises that may be regarded as small projects so that project-related difficulties manifest and can be addressed, e.g., by checking the consistency of equations and numerical solutions with limiting cases that can be solved analytically.

We maintain that this approach is not only suitable as a low-threshold introduction to scientific computing as early as High School and up to undergraduate Physics classes at University. It should also be a good start for students who later choose to specialize in Computational Physics or, more generally, for professional use.

1.2 What Can You Expect?

What can you, dear reader, expect from this book?

You can expect to be introduced to the world of Python and EXCEL by:

- training to work with numpy arrays, list slicing and broadcasting in Python,
- working with similar constructs, vector structures, and matrix operations also in EXCEL,
- learning how to write programs with looping, logical queries, and functions in Python and VBA for EXCEL,
- training how to lay out spreadsheets clearly so that they are apt for simple scientific computing,
- developing VBA macros that exchange data with spreadsheets,
- applying standard mathematical methods numerically.

and, with that,

- getting a better understanding of certain mathematical and physical concepts.

After having successfully completed the exercises, you should have gained so much self-confidence that you can answer the question “Programming practice?” with an enthusiastic “Yes!”.

1.3 What Do You Need?

You will need a Physics textbook (anyone will do, e.g., the one you have at hand during your studies anyway) and two more books on programming as indicated below.

EXCEL

To work with EXCEL, you only need a computer in which EXCEL has been implemented (any version; the exercises in this book have been checked in EXCEL 2010 and EXCEL 2019) and an introduction to EXCEL (do not buy one before having done the basic exercise in Sect. 2.3). In particular, you do not need a special development environment for VISUAL BASIC, because it is included in all versions of EXCEL.

Python

You will need to install Anaconda, a free and open-source distribution of the Python and R programming languages that also comprises the Jupyter Notebook by default. The examples in this book were obtained with Python 3.7 in Jupyter. You are advised to use both a book and internet courses to broaden your training systematically. Make your choice after having gone through Exercises 2.4 and 2.5.

1.4 Tim, Alac, and Mag

You will soon meet two types of students and a tutor who will accompany us throughout this book. The character named *Tim* (which stands for “timidus” or “timida”, meaning shy) represents those students who are somewhat hesitant, fearing that they may fall short of the requirements, although they study hard. The character named *Alac* (which stands for “alacer”, meaning alacritous, high-flying) is typical of those vehemently self-confident students (men are generally over-represented) who believe that they already have a superior overview and do not have to deal with what they consider mere bits and pieces. *Mag* (for Magister/Magistra, i.e., the tutor who runs the course) tries to engage with both characters, encouraging Tim and cautioning Alac, and clarifying that both approaches are valuable and that every Physics student should venture into the Computer world.

Tim: Computers are not my thing

► **Tim** I see how well some fellow students are juggling programming tools, but I'd rather stand back. I prefer to learn the stuff from textbooks.

► **Mag** This course is not intended to turn you into a computer nerd. You will not learn any cool tricks. We restrict ourselves to some basic techniques that are practiced again and again. The computational techniques do not stand by themselves, but are always taught in connection with physical problems.

► **Tim** But I have often heard that programming is a black art for which you have to be specially talented.

► **Mag** Here, you will learn the most basic computer techniques that every scientist, engineer, and science teacher must master to succeed in their profession.

► **Alac** Why EXCEL and Python?

► **Mag** All algorithm-oriented computer languages have a similar structure. Knowledge of specific commands is not the most important thing. You have to learn how to translate physical and technical problems into a computational structure. Furthermore, the mistakes that beginners make are always the same in all computer languages. The most important thing is to track, correct, and, finally, avoid them. Anyhow, as we shall have spreadsheet solutions and Python programs in parallel, you will be sensitized more towards common structures than the peculiarities of specific software.

Alac: How do I become a master programmer?

► **Mag** A master can be recognized by how he/she deals with errors. Any unnoticed error in spreadsheet formulas and programs can lead to disaster. It is essential that you gain experience with data structures and programming constructs.

► **Alac** And that is what this course will accomplish?

► **Mag** Yes! By using data structures in spreadsheets and Python programs and setting up graphical representations that are comprehensible, even when you look at them after some time. And by developing simple procedures that control the program flow.

► **Tim** Data structures, procedures, controlling; that sounds pretty challenging. How can I learn all of that?

► **Mag** Let's compare this course with learning a foreign language. How do you learn foreign languages?

► **Alac** Learning? For foreign languages, academic learning is useless in the long run. You simply have to go abroad; the rest follows by itself.

► **Tim** Oh, I couldn't learn like that. I couldn't form a proper sentence in a foreign language without profound foreknowledge. I would have to learn the correct grammar and vocabulary first before I would dare to speak.

A good balance

► **Mag** We are trying to find a good balance. You will learn the most straightforward “sentence” structures, but will also be “sent abroad” right off, and you will have to make your way there. If you pass this test, you can be confident of being able to learn the more complex “grammar” if necessary.

► **Tim** Is that thorough enough?

► **Alac** Will I learn the more tricky constructions too late?

► **Mag** Don't worry! Working through this book will make you fit for a computer-oriented world, be it for spreadsheet calculations in business or scientific computing in research. This can be tedious, but it will be worthwhile, whether it be as early as learning at school or working for a Bachelor's or Master's, or even as late as working on a Ph.D. thesis.

► **Tim** Can I manage this in addition to my studies in Physics?

► **Mag** I think so. Anyway, this course is about physics and will help you to pass your exams.

1.5 Didactic Concept

Workshop atmosphere

Having cleared up the doubts harbored by Tim and Alac, we now explain the didactic concept of this book.

In the courses at the University of Duisburg-Essen on *Physics with Excel and visual basic*, learning was mostly done in a workshop, such as in physics labs for beginners. The students dealt with the tasks alone or in pairs while in a computer lab, ideally also helping each other out across groups and consulting the supervisor when needed. Students could continue to work on their tasks outside of attendance time so that everyone could work according to their learning progress.

Experience shows that the students enjoy the tasks, and the learning progress is fastest when all three aspects—programming, physics, and mathematics—are combined. The systematic practice of various isolated spreadsheet and programming techniques is often perceived as too dull. The combination of calculations and graphs, realized in nearly all exercises in this book, proved to be particularly instructive.

Courses with 30 attendance hours

At the University of Duisburg-Essen, two EXCEL-based courses were offered, each with 30 attendance hours:

- a basic course for beginners, in which two tasks from each of the six Chaps. 2, 3, 4, 8, 9 and 10 were worked on and had to be presented to the supervisor;
- an advanced course with two tasks each from Chaps. 5, 6 and 7, and one task that had not yet been worked on from the chapters of the beginner’s course. Sometimes, two short exercises were combined into one task.

Subjects from a one-year physics course

The exercises rely on the subject matter from the first year of undergraduate physics. We do not intend to repeat physics that can be found in standard textbooks. Therefore, the introductions to the tasks are concise, but the solutions are presented in great detail. Experience shows that this creates the risk that the students might work through the exercises mechanically without caring about the physics context. To counteract this tendency, simple questions regarding physics and programming are asked in the middle of the text and answered in footnotes.

In Chaps. 8, 9 and 10, statistical concepts are illustrated in greater detail through simulations, because many students lack basic knowledge in this area. Although no theorems are logically derived, their structure should become clear, because the simulations follow the mathematical ideas.

Simple solutions preferred

The material is presented in nine chapters, each featuring about five detailed exercises. The aim is to pursue clear and simple solutions in which the physical justification for each step is traceable. To achieve this, suboptimal solutions, suboptimal with respect to computational efficiency and numerical precision, are often presented instead of solutions that are perfect from the outset. It has proven to be didactically more efficient to point out the shortcomings of this first approach and give the reader tools for improvements.

Broom rules

To many beginners, spreadsheet calculations and, especially, computer programming seem like witchcraft. We like to address this idea by setting up “broom rules” that the students hopefully will not forget so easily. Some examples: “ Ψ *Half, half, full; the halves count twice*” (Runge–Kutta of the 4th order) or “ Ψ *Mostly, not always*. (“fundamental rule” of statistical reasoning, no statement is 100% sure).

In addition, *Mag* puts stumbling blocks along the learning path, in talks with the two student characters, *Tim*, who learns the material from the beginner’s course diligently, and *Alac*, who does not hesitate to implement premature ad-hoc solutions. It is important to emphasize that both attitudes have their advantages and shortcomings, and neither student should feel denigrated. It is just that some students have to be encouraged to venture into the programming world, whereas others have to be cautioned against rushing too quickly into coding.

Exam questions

At the end of every chapter, a collection of rehearsals and tasks is presented, typically requested in written and oral examinations.

1.6 Subject Matter

Block A, Fundamentals (Chaps. 2, 3 and 4)

Ψ The dollar makes it absolute.

The student will learn how to organize spreadsheets, design charts reasonably, and implement simple programming procedures. The necessary computational techniques are embedded in Physics tasks and trained with clearly arranged formula calculations, presentations, interpretations of curves, and simple mathematics. The reader should consult, in parallel, systematic introductions to Excel and Python for help.

Block B, Physics and Mathematics

Ψ Half, half, whole, the halves count twice.

In Chap. 5, the reader will find exercises for analysis and vector and matrix calculations in the form of a spreadsheet-specific introduction to mathematics with parallel Python programs.

In Chap. 6, the knowledge gained in Chaps. 2 to 5 is applied to the kinematic superposition of movements, including animated charts.

In Chap. 7, we deal with various methods for solving Newton's equation of motion and apply them to one-dimensional motions, such as a jump from the stratosphere, Exercise 7.4, or a bungee jump, Exercise 7.6.

Block C, Simulation and analysis of experiments

Ψ If in doubt, count!

Evaluation of measurements is regarded as a critical skill to be exercised at the beginning of studies in Physics. Therefore, this block is particularly detailed and illustrated with simulations based on chance, because, as experience shows, many students' most significant knowledge gaps are in the field of probability and statistics. Furthermore, statistics is the branch of mathematics that is most important outside of technical professions.

We intend to develop a good understanding of concepts through statistical experiments with random numbers without going deep into formal mathematics. Statistical rules are intended to be illustrated and checked through multiple repetitions of simulations designed to test the hit rate ("Does the error range capture the true value?"). For this purpose, random number generators are introduced in Chap. 8, e.g., for normal distributions.

The student will learn how to analyze and graphically represent measurements (Chap. 9), emphasizing the precise meaning of error ranges ("C-spec errors" related to confidence levels). Before this can be done, the measurement process must be

simulated realistically to obtain data that can be evaluated. Our tools for simulation are random numbers generated according to the desired distribution.

With linear regression, mathematical functions are fitted to sets of measured values to get trend lines through data points (Chap. 10). Furthermore, an introduction to the important technique of non-linear regression with solver functions will be given, again, in both EXCEL and Python.

► **Follow-up book** A follow-up book, “Physics with Excel and Python, Using the Same Data Structure. Applications”, is being prepared, dealing, in the same style, with advanced topics, structured according to physical and mathematical aspects, such as:

- properties of oscillations,
- motions in the plane,
- the steady-state Schrödinger equation,
- partial differential equations,
- Monte Carlo methods,
- wave optics,
- statistical physics, and
- variational calculus.

1.7 Getting Started with Excel

1.7.1 Start Menu

In Fig. 1.1, you see the start menu of EXCEL 2019, where the main tab FORMULAS has been activated, and the cursor has been positioned over the group

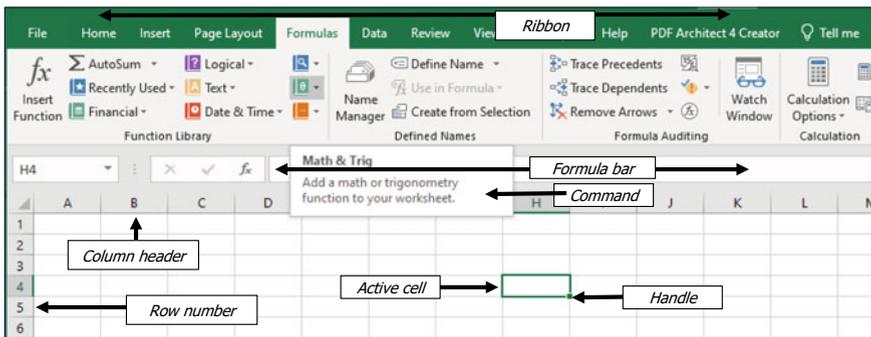


Fig. 1.1 The start menu of EXCEL, with the main tab FORMULAS activated

FUNCTION LIBRARY to show the command MATH&TRIG. Arrows indicate the different elements of the start menu, namely ribbon, tab, group, formula bar, and command, as well as elements of the working area, column header, row number, active cell, and handle of a cell. To indicate a “click path” in the text, we write a sequence TAB/GROUP/COMMAND/FUNCTION, e.g., FORMULAS/FUNCTION LIBRARY/MATH&TRIG/COS to call the cosine function.

Throughout this book, we will take screenshots from EXCEL 2019. Experience has shown that students can work with these instructions in every version of EXCEL without major difficulties.

1.7.2 Spreadsheet Presentation

Γ structure of a spreadsheet

In Fig. 1.2 (S), a spreadsheet organization in the Γ structure, often employed in our exercises, is shown. With Γ (“gamma”), we refer to the straight lines above C14:G14 and to the left of C14: C174.

Above Γ:

- the parameters of the task are defined in the range C2:C6,
- these cells get the names in B2:B6, with which they can be called in formulas,
- the most informative parameters of the exercise are integrated into a text, here, in cell E4 (with the formula in E5) that can be taken as a legend in a figure.

Left of Γ:

- the values for the independent variable t are in B14: B174.

	A	B	C	D	E	F	G	H
1	Prespecifications							
2	Amplitude of pendulum	Ap	1.50					
3	Period of pendulum	Tp	1.20					
4	Period of rotation	Tr	9.00		Tp=1.2; Tr=9			
5	Time interval	dt	0.0173		= "Tp="&Tp&"; Tr="&Tr			
6	Suspension point vs. rot. axis	xSh	0.00					
7	Calculated therefrom							
8	Angular frequency pendulum	wP	5.24 =2*Pi()/Tp					
9	Angular frequency rotating disc	wR	-0.70 =-2*Pi()/Td					
10								
11								
12								
13								
14		t	xP	xT	yT	xSt	ySt	
15		0.000	1.50	1.50	0.00	1.50	0.00	
16		0.017	1.49	1.49	-0.02	1.50	-0.02	
174		2.768	-0.52	0.18	0.49	-0.53	-1.40	

Fig. 1.2 (S) Typical Γ structure of a spreadsheet, here, for the calculation of the trace of a Foucault pendulum; rows 16–173 are hidden

1 Sub Protoc()	Range("C3") = Tp	8
2 r2 = 13	Cells(r2, 10) = Tp	9
3 Cells(r2, 10) = "Tp"	Cells(r2, 11) = Range("D174")	10
4 Cells(r2, 11) = "xT"	Cells(r2, 12) = Range("E174")	11
5 Cells(r2, 12) = "yT"	r2 = r2 + 1	12
6 r2 = r2 + 1	Next Tp	13
7 For Tp = 1 To 9	End Sub	14

Fig. 1.3 (P) Log procedure, changes the period of the rotational motion in Fig. 1.2 (S) and logs the values of x_T and y_T at the last instant of the calculation period

Below Γ :

- values are calculated from the parameters and independent variables,
- the range C14:G174 contains five columnar vectors of length 171 with the names in row 13,
- row 12 includes, in oblique orientation and in italics, the text of the formulas in the bold-printed cells of the column below. If no cell is printed in bold, the formula applies to the entire column.

Nomenclature

Python-typical terms are printed in the *Courier* font.

When EXCEL-typical terms are referred to in the text, e.g., function names, they are set in SMALL CAPS; examples: IF(CONDITION; THEN; ELSE) . Spreadsheet formulas are given in the form B15 = [=B14 + dt]. The expression in rectangular brackets corresponds exactly to the entry in the cell, including the equal sign. The equal sign specifies that it is a formula that is in the cell.

Three types of figure are distinguished, two of which are denoted by suffixes, (S) for spreadsheets, e.g., Fig. 1.2 (S), and (P) for the code of Visual Basic programs, e.g., Fig. 1.3 (P). Figures without a suffix are line drawings or screenshots, e.g., Fig. 1.1.

Names given by the programmer are printed in the text in italics, e.g., *f*, *d*. Sometimes in EXCEL, names are used that contain a dot, e.g., "T.1" or "x.2". This is because T1 and X2 are reserved for cell addresses. The associated variables are referenced in the text without a dot, but with subscripts, i.e., as T_1 and x_2 .

Physical units

Sometimes, no physical units are specified in the axis labels of the figures. They can then be deduced from the physical units of the parameters.

1.8 Getting Started with Python

You first have to install Anaconda with Python. There are many instructions on the Internet as to how to achieve that, e.g., <https://docs.anaconda.com/anaconda/install/windows/> or <https://www.jcchouinard.com/install-python-with-anaconda-on-windows/> (2020-09-02).

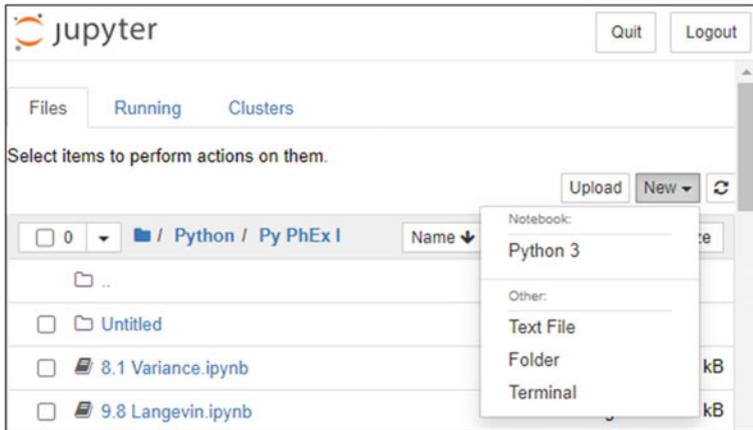


Fig. 1.4 Window opened to create a new program file

When the Jupyter Notebook is opened, an overview of the files and single files on the localhost is shown. The programs used for this book are in a sub-filer “Py PhExI” of the main filer “Python”. When we click *Python/Py PhEx I*, the window in Fig. 1.4 pops up. To edit an already existing file, we have to click on that file.

To create a new file, we open the list “New” and click on “Python 3”. A new window pops up, opening a new file “Untitled22” with an empty program cell “In []”. The version in Fig. 1.5 is displayed after a small program has been written into that cell. “In [5]” indicates that the 5th version of the code is shown. This program is executed by clicking the button “Run”. The result of the instruction `print[x]` is displayed in the output cell created automatically below the program cell.

1.9 Skills to Be Trained

The different programming techniques are distributed over various exercises. For the purpose of learning about them and how to revise them, the following lists of keywords and broom rules have been compiled. They are meant to assist the readers with the revision of subjects and, of course, their preparation for examinations.

Spreadsheet operations (Exercise 2.3)

- Using cell addressing, absolute, relative, indirect
- Ψ *The dollar makes it absolute.*
- Naming cell ranges and using the names in formulas

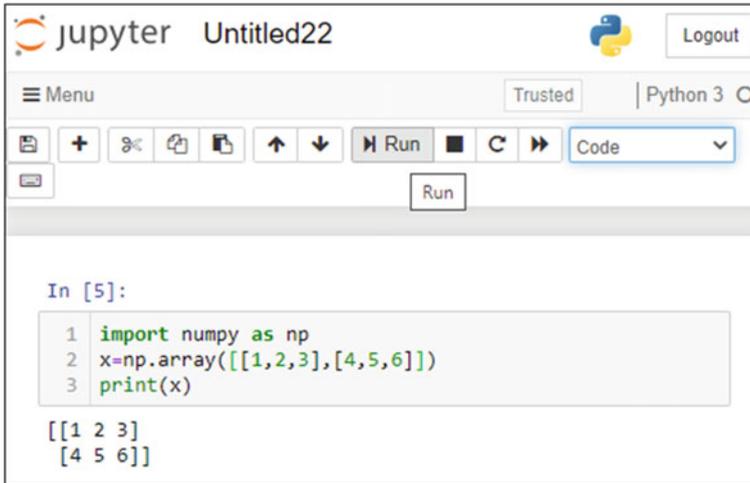


Fig. 1.5 A program creating an output just below the program cell

- Using sliders to change cell contents
- Scaling and formatting XY scatter diagrams
- Creating smart legends by linking text and variables, Ψ “Text” & Variables.
- Gamma structure of tables (Sect. 1.7.2)
- Ψ Empty lines separate curves.
- Ψ $Ctrl + Shift + Enter$. Magic “chord” to complete the entry of matrix functions in EXCEL (Exercise 2.6).

Python constructs (explained mainly in Exercises 2.4 and 2.5)

- Use of numpy arrays
- Ab-initio constructors `np.arange`, `np.linspace`
- Creating one and two-dimensional arrays with `np.array` (row vectors, column vectors, matrices)
- Broadcasting row vectors, column vectors, and matrices together in algebraic operations
- Slicing of lists
- List comprehension
- Creating smart legends by linking text and variables
- Applying a standard function to create scatter diagrams
- Creating animated figures (Exercise 6.2).

VBA-macros and Python instructions (Chap. 4)

The terms ‘routines’, ‘programs’, and ‘procedures’ are all used synonymously here. The term ‘macro’ is for VBA only.

- *For, if, Sub/def*, basic structures of programming: loops, logical queries, subroutines (Exercises 4.4 and 4.5)
- *Loop2i*, loops with a loop index and a running index incremented within the loop (Exercises 4.2 and 4.8)
- Systematically modifying parameters and recording the results of the spreadsheet calculations with *rep-log procedures*
- Processing and decoding texts for evaluation of the protocols of measuring instruments (Exercises 4.8 and 4.9)
- Writing formulas into cells with procedures (Exercises 4.1 and 4.2)
- Creating user-defined functions and using them in spreadsheet calculations (Exercise 4.9)
- Linking macros to control elements (command buttons, sliders) in spreadsheets (Sect. 4.3.3).

Mathematical techniques

Ψ *Imaging equation for lenses with plus and minus!* (Exercise 3.2).

- Using the line equation constructively (Exercise 3.2)
- Calculating with vectors in the plane and displaying them in diagrams (Exercises 5.5, 5.6, 5.7)
- Calculating with matrices (Exercise 5.9)
- Converting polar coordinates and Cartesian coordinates into one another (Chap. 6)
- Ψ *Doppler effect with plus and minus* (Exercise 3.3)
- Differentiating (Exercise 5.3) and integrating (Exercise 5.4) numerically
- Weighted sum (Exercises 5.8 and 6.5) and weighted mean (Exercise 6.5).

Functions

- Properties of the exponential function (Exercise 3.4)
- Ψ *First, the tangent at $x = 0!$* (Exercise 3.4)
- Ψ *Plus 1 becomes times e .* (Exercise 3.4)
- Use of the logarithm function for different computing tasks (Exercise 9.3)
- Addition of sines and cosines: overtones and beats (Exercise 2.7)
- Ψ *Cos plus Cos equals mean value times half the difference.* (Sum rule of cosines, Exercise 2.7).

Integration of Newton's equation of motion (Chap. 7)

- Ψ *Approximated average value instead of exact integral*
- Four numerical methods (Sect. 7.1.2, Exercise 7.2):
- Euler
- Progress with look-ahead (our standard procedure in a spreadsheet calculation)
- Half-step
- Runge-Kutta of fourth order, Ψ *half, half, whole, the halves count twice.* (our standard procedure as a Python function).

Statistics (mainly Chap. 8)

- Ψ *Mostly, not always.* Fundamental rule of statistical reasoning
- Ψ *If in doubt, count!*
- Ψ *Come to a decision! You may be wrong.* (Exercise 9.6)
- Generating random numbers with specified distribution (Exercises 8.5, 8.6, 8.7)
- Ψ *Chance is blind and checkered.* (Exercise 8.3)
- Empirical frequency distribution (Exercise 8.2), in EXCEL: Ψ *Always one more! Yes, but of what and than what?* In Python: Ψ *Always one less!*
- χ^2 test for comparing theoretical and empirical frequency distributions (Exercises 8.2, 8.8)
- Multiple repetitions of random experiments to test for uniform distribution of the results of χ^2 tests (Exercise 8.2).

Evaluation of measurements (mainly Chaps. 9, 10)

- Ψ *We know everything and play stupid.*
- Simulating measuring processes and evaluating the generated data sets statistically (Exercises 9.2, 9.9).
- Mean, standard deviation, Ψ *Two within, one out of (the standard error range)* (Exercises 9.4 to 9.8)
- Specifying measurement uncertainty (Exercises 9.2, 9.7 and 9.8)
- Multiple repetition of random experiments to test for error rate (Exercises 9.4, 9.5 and 10.3)
- For only a few repetitions of measurements, taking the t -value into account (Exercise 9.5).
- Ψ *Twice as good with four times the effort* (Exercise 9.4).
- Error propagation (Exercises 9.7, 9.8), Ψ *Calculate with variances, report the C-spec error!*
- Ψ *From variance to confidence with Student's t value* (Exercise 9.8).
- Reducing measurement uncertainty by combining measurement series (Exercise 9.6).
- Ψ *Worse makes good even better.*
- Linear regression, trend lines, coefficients with uncertainty (Sect. 10.2, 10.3 and 10.4).
- Applying non-linear regression using SOLVER in EXCEL and `curve_fit` in Python (Exercises 10.5, 10.6 and 10.7).
- Decoding textual logs of measuring instruments and writing the relevant data into tables using VBA macros or Python programs (Exercise 4.8).



Data Structures, Excel and Python Basics

2

This chapter aims to develop computational solutions for physics problems, parallel in EXCEL and Python based on the same data structure and the same type of list processing. In this way, EXCEL may serve as a low-threshold entry into scientific computation with a smooth transition to professional platforms. We proceed in three steps: (1) Basic spreadsheet techniques are introduced: absolute and relative addressing and naming of cells, creating charts, and using sliders (scroll bars), with the didactic goal of addressing variables in functions by their names, just as in mathematical formulas. (2) Python basics are explained, with emphasis on the manipulation of Numpy arrays essential for scientific computation. (3) Matrix operations are introduced in EXCEL, equivalent to those in Python. Finally, in one exercise, a set of four parabolas and, in another exercise, a group of four cosines (to simulate acoustic phenomena) are calculated and displayed in parallel in both applications.

2.1 Introduction: Named Ranges in Excel, Arrays in Numpy

Solutions of Exercises 2.3 (Excel), 2.4 (Python), 2.5 (Excel), and 2.6 (Python) can be found at the internet address: go.sn.pub/9Rtzxi.

Spreadsheet technology

This chapter is about how cells are addressed, figures are created and formatted, and sliders are used to change cell contents. This will be easy for you if you are already familiar with EXCEL and know how to write formulas into cells. If you are less experienced, you will first have to go through the basic exercise step by step. If necessary, consult the EXCEL help guide, and, finally, after having gone through the

basic exercise, find a textbook about EXCEL techniques that is best suited for your learning style.

Required and practiced EXCEL techniques are:

- relative and absolute cell addressing,
- direct and indirect cell addressing,
- naming of cells and cell ranges,
- creation of diagrams,
- application of sliders.

We first exercise different types of cell addressing. Our goal, however, is to write formulas as mathematically as possible, i.e., with letters representing variables. Then, they will be identical to formulas in Python. For this purpose, individual cells, ranges of cells within rows (row vectors) or columns (column vectors), and two-dimensional cell ranges (matrices) are to be designated by names. All of these techniques will be introduced step by step in the individual exercises and summarized again in Sect. 2.3.5. The systematic use of vectors and matrices is the reason why spreadsheet calculations can be translated nearly literally into Python.

Figures representing spreadsheets are characterized by the supplement (S), e.g., Fig. 2.2 (S).

Formulas in spreadsheets

Formulas in spreadsheets are reported in italic and often in oblique orientation, valid for a cell in the neighborhood in bold font. They will be written in the text in brackets; e.g., for the content of cell A11, we write $A11 = [=A10 + dx]$. We have to distinguish whether or not an equality sign is written in the cell. For the expressions $A9 = [x]$ and $A10 = [3]$, no equal sign is written in the cell; $[x]$ is thus interpreted as text and $[3]$ as a number.

Python constructs

We will learn Python programming by working with the program cell structure of the Jupyter notebook, first dealing with list processing and then focusing on operations on arrays in Numpy. The explanations are less technically detailed than for EXCEL, because list processing is the core business of Python and has been well described in numerous textbooks and online courses. However, our examples are designed so that the essence of the definitions and procedures should become obvious.

The Python constructs for list generation (`np.linspace` and `np.arange` in the `numpy` library) will be used to define vectors that are later transformed with standard functions into other vectors using list comprehension and broadcasting. To mimic the column vectors in spreadsheets, a two-dimensional list with only one row has to be introduced and transposed.

Matrix operations

Having provided EXCEL with the necessary matrix formulas, we can finally demonstrate broadcasting for algebraic operations and some operations of linear algebra parallel in EXCEL and Python (Exercise 2.5).

Applications

We practice our newly acquired knowledge in an exercise on four parabolas and their upper envelope. The chapter concludes with a physically meaningful exercise treating the sum of four cosines so as to demonstrate overtones, beats, and the addition theorem of cosines.

2.2 Characteristics of a Parabola

Starting from its vertex form, we set up a data structure to tabulate and plot a parabola, together with its characteristic features *focus* and *directrix*. The data structure set up here is to be used in Exercise 2.3 for a single parabola and in Exercise 2.6 for a set of 4 parabolas.

2.2.1 Different Definitions of a Parabola

Parabola from vertex

A parabola is to be presented in a diagram. Its standard form is defined as

$$y = a + b \cdot x + c \cdot x^2 \quad (2.1)$$

It is, however, more intuitive to start from its vertex form

$$y = y_V + c \cdot (x - x_V)^2 \quad (2.2)$$

because its shape is immediately clear: its vertex is at (x_V, y_V) , and its curvature is proportional to c (positive or negative). Transforming into the standard form yields

$$a = y_V + c \cdot x_V^2 \text{ and } b = -2 \cdot c \cdot x_V \quad (2.3)$$

In Fig. 2.1a, a parabola is shown with its vertex marked with a diamond.

Question

What are the coordinates (x_V, y_V) of the vertex and the value of c in Fig. 2.1a?¹

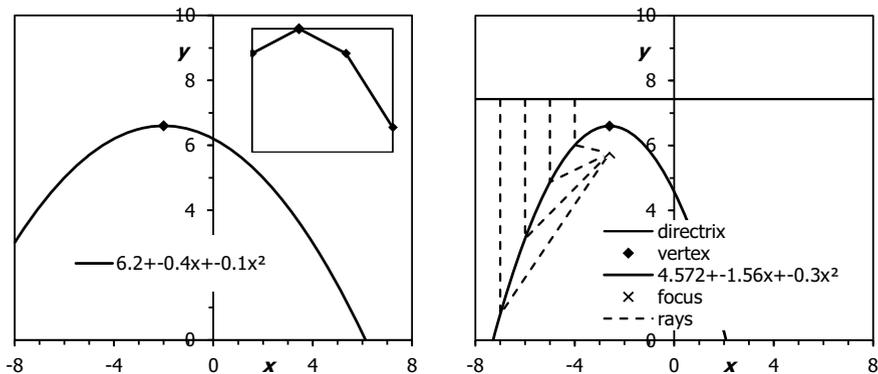


Fig. 2.1 a (left) A parabola with its maximum at $(-2.6, 6.6)$. b (right) Connecting the parabola with its focus and its directrix; compare Fig. 2.10

Focus and directrix

A parabola may also be defined as the locus with equal distance to the focus (x_F, y_F) and to the directrix $(y = y_D)$, both of which are depicted in Fig. 2.1b. The coordinates of the focus and the directrix are:

$$x_F = x_V \quad y_F = y_V + \frac{c}{4} \quad y_D = y_V - \frac{c}{4} \quad (2.4)$$

Ignoring reflection, all rays incident parallel to the symmetry axis of the parabola have the same length up to the directrix, e.g., when calculated from the line $y = 0$. By definition, they all have the same length up to the focus when reflected at the parabola. This is why light incident parallel to the optical axis is focused in the focus of a parabolic mirror.

Calculation points of a curve

Equations 2.1 and 2.2 represent continuous functions. In numerical calculations, however, the function values y are calculated on only a finite number of discrete x -values, x_i (see the inset picture in Fig. 2.1a). The points (x_i, y_i) are called the calculation points of the function. They are the vertices of the polyline representing the graph. In most of our exercises, the adjacent calculation points are equidistant on the x axis, i.e., they all have the same distance to their respective neighbors, usually specified in a parameter dx .

¹ $x_V \approx -2$, $y_V \approx 6.5$ from visual inspection in the coordinate system; $c = -0.1$ from the legend in the figure.

2.2.2 Data Structure and Nomenclature

(x_V, y_V)	coordinates of the vertex of the parabola
a, b, c	coefficients of the parabola in standard form
y_D	ordinate value of the directrix of the parabola
(x_F, y_F)	focus of the parabola
dx	distance between adjacent x values of calculation points
x	sequence of x values
y_P	values of the polynomial for x
y_A, y_B, y_C, y_D	values of the same polynomial but calculated in Excel with column names
y_{Max}	upper envelope of y_A, y_B, y_C, y_D .

2.3 Basic Exercise in Spreadsheet Calculation

With the example of tabulating a parabola and displaying it in a diagram, we train the basic spreadsheet techniques: absolute and relative cell addressing, providing cells with names, connecting text and numeric variables to get informative legends for diagrams, applying scroll bars to change cell contents without typing numbers. After performing these exercises, the reader should be able to select a more detailed textbook on EXCEL techniques that is best suited to her/his taste and needs. At the end of the exercise, an analogous Python program with the same data structure is presented.

2.3.1 Cell Addressing

Spreadsheet layout

A spreadsheet layout for generating a parabola $y_P = f(x)$ is displayed in Fig. 2.2 (S). We are going to review three regions successively: A1:B6 (to explain cell addressing), D1:H3 (to name cells and apply sliders), and A8:E169 (to name cell ranges). All relevant cells and the ranges x , y_P , and y_A get names from the beginning. The formulas in cells are printed in neighboring cells in italic to keep track of the calculations.

Relative and absolute cell addressing

The coefficients of the vertex form of the parabola, x_V , y_V , and c , are defined in B1:B3 and recalculated in B4:B5 into b and a of the standard form, with the formulas in C4:C5. The value in B6 specifies the horizontal distance dx between the calculation points.

	A	B	C	D	E	F	G	H
1	xV	-2 vertex		$= (E1-40)/5$	30			
2	yV	6.6 vertex		$= E2/10$	66			
3	c_	-0.1		$= (E3-50)/10$	49			
4	b	$-0.4 = -2*B3*B1$		$= -2*c_*xV$				
5	a	$6.2 = B2+B3*B1^2$		$= yV+c_*xV^2$				
6	dx	0.1		$6.2+0.4x+0.1x^2$				$= a&"+"&b&"x+"&c_&"x^2"$
7	$=A9+\$B\6		$=\$B\$5+\$B\$4*A10+\$B\$3*A10^2$		$=a+b*x+c_*x^2$			
8	x	yP	yA					
9	-8.0	3.0	3.0					
10	-7.9	3.1	3.1					
169	8.0	-3.4	-3.4					

Fig. 2.2 (S) Tabulation of a parabola with the parameters c, b, a in B3:B5; F1:H3, sliders to determine the parameters x_V, y_V, c ; E8:E169, alternative tabulation of the parabola using the named cells $a, b, c_$ and the column range x

The independent variables x (left of Γ) are in A9:A169. They are obtained by entering the initial value, here -8.0 , into the first cell A9 of the range and the formula $A10 = A9 + \$B\6 or $A10 = [A9 + dx]$ into the next cell and copying this formula into the whole range. Copying is done by seizing the handle of A9 (see the small black square at the lower right edge in the subpicture in column C) with the pointer and dragging it down to A169.

► **Task** Change the contents of cell B6, named “dx”! All x values in cells A10:A169 should adjust themselves immediately.

The values for y_P are obtained by entering the formula reported in B7 into B10 and dragging it up to B9 and down to B169. The formula is most conveniently obtained by first entering [=], and then by clicking on the corresponding cells and continuing with the operators $*$ for multiplication and $^$ for potentiation, resulting in $B10 = [=B5 + B4*A10 + B3*A10^2]$. In the last term, the variable taken from A10 must be squared. This is done with the power operator $^$. You have to press the button with the $^$ -sign and then the desired power, “2” in our case. Only after the second step does the operator [2] appear in the cell.

When this formula is copied into another cell, the cell addresses change accordingly. Copying into C11 would yield $C11 = [=C6 + C5*B11-C4*B11^2]$, realizing *relative* addressing but not giving the desired result, because we would like to keep the cells with the coefficients constant. This is achieved by making the references to B3:B5 *absolute*, with dollars as prefixes: $\$B\$5, \$B\$4, \$B\3 , either by introducing the \$ sign explicitly before the column letter or the row number or by pressing the function key F4, resulting in the formula reported in B7.

Having now copied this into C11, we would get $C11 = [=\$B\$5 + \$B\$4*B10-\$B\$3*B10^2]$, with the values for x copied incorrectly (A10 becomes B10), because they are still relatively addressed. Making the address of column A absolute is achieved with $\$A10$, $B10 = [=\$B\$5 + \$B\$4*\$A10-\$B\$3*\$A10^2]$. If we now copy this formula into another cell, only the row number 10 changes, e.g., into 12 when copied into any column in row 12.

Making a cell reference absolute can also be achieved by pressing the function key F4 several times. Key words for the EXCEL help: Absolute, relative, and mixed cell references.

There is a more elegant way to copy a formula down a column: clicking onto the “fill handle” (the bottom right corner) of the cell that contains the formula. Then, the cell contents are immediately continued down to the 169th row, i.e., for all cells for which there is an entry in the neighboring column, here column A, until the first empty neighbor cell is encountered.

Questions

Questions concerning Fig. 2.2 (S):

From A9 to A10, x increases by $dx = 0.1$. Why is the increase 15.7 for the next jump from -7.9 to $+8.0$?²

Where is dx defined?³

Why is the name $c_$ in E7 provided with an underscore?⁴

Having gotten this far, we have programmed our first function. Changing the values of x_V , y_V , and $c_$ in B1, B2, B3, the function values in column B adapt immediately. We may now proceed to the section “Graphical representation” to see the resulting curve, but should come back to learn about naming cells.

Naming cells

In the range B1:B6 of Fig. 2.2 (S), parameters are defined that are accessed in various parts of the worksheet. To call them like variables in mathematical equations, we provide the cells with the names written to the left of them. This is done by activating the range A1:B6 comprising names and values and clicking through (EXCEL 2019): FORMULAS/DEFINED NAMES/CREATE FROM SELECTION.

A prompt appears, “CREATE NAMES FROM VALUES IN THE LEFT COLUMN?”. The answer is YES, that the agent has correctly detected, and we confirm this by clicking OK. For more about the NAME MANAGER, see Sect. 2.3.5. We can now refer to these cells by their names, anywhere in the spreadsheet and, indeed, throughout the whole book.

When writing, e.g., $[=a]$ into a cell somewhere in the spreadsheet and pressing ENTER, this cell immediately gets the numerical value corresponding to a , in our

² Rows 11 to 168 are hidden. The jump is over 159 advances of dx ; $-7.9 + 15.9 = 8.0$.

³ The value for dx is set in cell B6, which is given the name in A6.

⁴ The name c is protected for EXCEL-internal use. A name c in a cell intended to become an identifier is automatically extended by an underscore.

example, 6.2. When the content of cell B5 is changed, the value of all cells with [=a] changes as well.

We can even name cell ranges, e.g., A9:A169 with the name x and B9:B169 with yP . When activating A8:B169 and proceeding as above, the Name Manager prompts us: "CREATE NAMES FROM VALUES IN THE TOP ROW?", and we confirm this by clicking OK.

We may now write $= a + b*x + c_*x^2$ into E9, more elegant and clearer than the formula in B7, with absolute and relative cell addressing, and copy down to D 169 to get the same values as for yP . For x , the formula in a cell takes the value in the same row in column A. The name manager has changed c into c_* because the letter c conflicts with a protected name in EXCEL.

2.3.2 Graphical Representation of a Function

After having created the function table of a parabola so beautifully, we would like to visualize the curve. To do so, we set the cursor into an empty cell, away from the filled cells, and click in the INSERT tab on INSERT/ CHARTS/, and on SCATTER within the CHARTS section. A blank chart is inserted.

Upon our activation of DESIGN/SELECT DATA (see Fig. 2.5b), a SELECT DATA SOURCE window opens. We click ADD, and a window as shown in Fig. 2.3 opens. For SERIES NAME, we click on cell B8 of Fig. 2.2, and for SERIES X VALUES, we activate A9:A169 and hit return. For SERIES Y VALUES, we activate B9:B169 and hit return. The empty chart changes to that shown in Fig. 2.4.

As the legend "yP" has been taken from a cell in the spreadsheet, it will adapt immediately when the cell entry is changed. If a legend in a chart is identical to that in the spreadsheet indicating the data, it helps in keeping the overview.

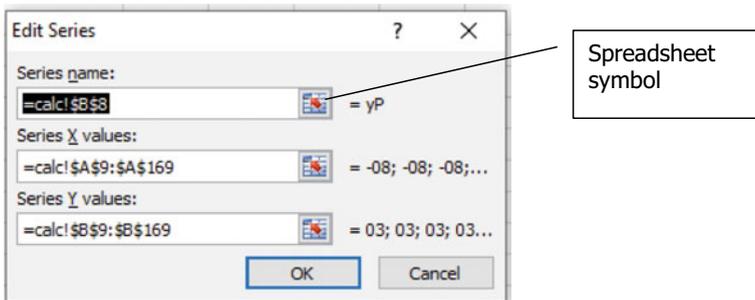


Fig. 2.3 Insertion of a data series into a chart; the series name is best taken from the worksheet by activating the relevant range, *not by entering it as text*

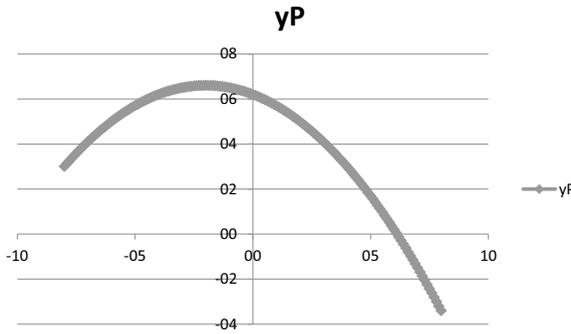


Fig. 2.4 Scatter plot after completing Fig. 2.3

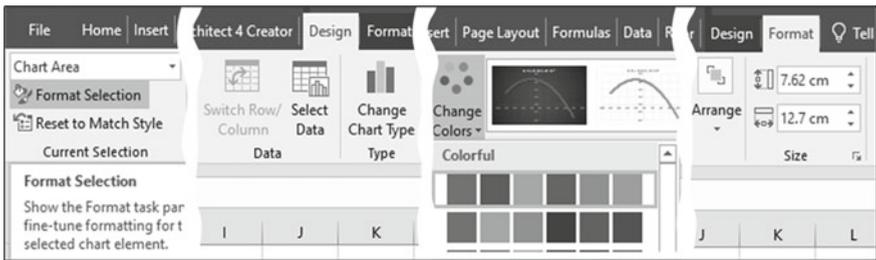


Fig. 2.5 Tabs, which are important for diagrams, after INSERT/CHARTS/SCATTER/WITH ONLY MARKERS in the Start menu (EXCEL 2019, EXCEL 2010 similar) or after activating an existing chart, **a** (left) FORMAT/CURRENT SELECTION, to the left of the start bar, to format an element of the diagram, **b** (center left) DESIGN tab / DATA group /SELECT DATA to select data to be entered into the diagram, **c** (center right) DESIGN tab /CHART STYLES, **d** (right) FORMAT/SIZE, appears after activating a diagram, to the right of the start bar, to specify the size of the diagram

Formatting the chart

We reshape the diagram according to our taste, e.g., as in Fig. 2.1a. After clicking on the diagram and HOME/FORMAT, the following components change:

- Size (7 cm high, 8 cm wide) (in the start bar to the far right, see Fig. 2.5d).

Also, after clicking on the relevant element of the diagram or selecting it from the leftmost register in the FORMAT tab (Fig. 2.5a, at first, only CHART AREA appears, but after opening the list by clicking on ▼, all items of the diagram appear), we choose before clicking FORMAT SELECTION:

- Format/Chart Area/Border /No line
- Format/Plot Area/Border Color/Solid line/Color/Black and Width/1pt

- Format/Data Series/Marker Options / Built in/Diamond, Size 4 and Marker Fill/Solid Fill/ Black
- Format/Horizontal (Value) Axis/Axis Options/Minimum /-8, /Maximum /8, /Major Unit/4.

► **Task** Change the value of the parameters x_V , y_V , $c_$ in Fig. 2.2 (S) and observe how the spreadsheet entries and the chart change!

► **Alac** That’s cool! The diagram is alive!

► **Tim** Once created, it’s always up-to-date!

2.3.3 Smart Legends in Figures

Smart legends are created by concatenating variables and text. In Fig. 2.4, we have specified the parabola with its name in the spreadsheet, in simple text. In Fig. 2.1a, however, the legend contains the actual parabola equation generated in D6 of Fig. 2.2 (S) with Ψ “Text” &variable.

= a&” + ”&b&”x + ”&c_&”x” in D6 yielding $6.2 + -0.4x + -0.1x$

If text is to be inserted, it must be enclosed in quotation marks, e.g., as above, “x + “. The concatenation operator is “&”. When the parameters a , b , or $c_$ change, the text in D6 and the legend in the chart will immediately follow.

► EXCEL Ψ “Text“& Variable concatenation with &

► Python Ψ „Text “ + Str(Variable) concatenation with +

Often, float values have to be rounded. When, e.g., $B3 = [= 1/3]$, then only 0.33 is displayed in the cell. However, if B3 is inserted into a legend, 0.333333333 appears. Setting ROUND(B3, 2), 0.33 is returned: [= “c_ = “&ROUND(B3; 2)] results in the cell content [c_ = 0.33].

2.3.4 Scroll Bars

Cell contents are changed with a slider (a scroll bar)

We can play even more impressively with the curves when we change their parameters with sliders. To introduce sliders, we select DEVELOPER/CONTROLS/INSERT/ACTIVE X CONTROLS to get the tabs in Fig. 2.6a.

If the tab does not appear in your toolbar, you have to supplement the toolbar by making a tick in FILE/OPTIONS/CUSTOMIZE RIBBON in the box before DEVELOPER.

We need the slider to be an ACTIVE X CONTROL ELEMENT, and we click on the icon for the slider in the top line on the far right, with the DESIGN MODE turned on,

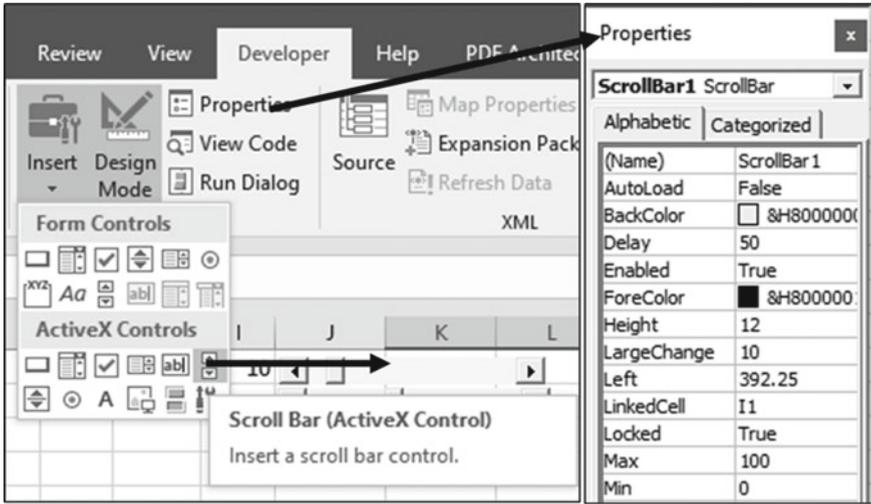


Fig. 2.6 (EXCEL 2019, EXCEL 2010 similar) **a** (left) Tab after going through DEVELOPER/CONTROLS/INSERT/ACTIVEX CONTROLS. A slider is listed in the upper row on the right, at the foot of the arrow. **b** (center) In J1:L1, a slider has been installed by clicking on the control element and pulling it up in said spreadsheet area. **c** (right) Menu for defining the properties of the slider (SCROLLBAR). It appears after we click on PROPERTIES in **a**; important parameters: LINKEDCELL, MINIMUM and MAXIMUM values

and then pull up a rectangle with the mouse at the desired place in the spreadsheet. In Fig. 2.6b, this is done in cells J1:L1. Now, with the DESIGN MODE still on, we can configure the slider. In Fig. 2.6c, the PROPERTIES list of the activated scrollbar is shown. We specify that it is cell I1 (LINKEDCELL) into which the number is to be written, and that the number should be between 0 (MIN) and 100 (MAX). We then turn the DESIGN MODE off by clicking on this icon again (see Fig. 2.6a) and move the slider’s thumb with the mouse. Immediately, a number appears in I1.

DESIGN MODE is activated and deactivated by clicking on the icon. When activated, existing control elements can be modified, or new ones added. When deactivated, the control elements can be operated.

When we grab the thumb (the rectangular bar in the middle of the slider) with the cursor and move it, the output in the linked cell changes. SMALLCHANGE specifies the jumps (to the left or the right) of the numbers when we click on the (left or right) edge of the slider. LARGECHANGE sets the jumps’ size when we click *within the slider bar* left or right of the thumb. Try it out!

Conversely, if we change the contents of a LINKEDCELL, the new value is entered into the slider’s memory, and its thumb will move.

We can use a slider (SCROLL BAR) to enter integers between 0 and 32,767 (= $2^{15} - 1$) into a cell (LINKEDCELL). Settings are specified in the PROPERTIES group:

For LINKEDCELL, specify the address of the cell to be written into. MIN and MAX can limit the value range.

Sliders in Fig. 2.2 (S)

In Fig. 2.2 (S), three sliders are introduced with linked cells E1:E3, wherefrom the parameters a , b , and dx in B1:B3 are calculated. When one of the sliders is operated, the cells' value and the curve in Fig. 2.1a adapt immediately.

A slider can produce only positive integers. If other numbers are needed, plug the value of the LINKEDCELL into a formula in another cell. In Fig. 2.2 (S), the value of the LINKEDCELL (E3), set between 0 and 100, is changed with $B3 = [(E3-50)/10]$ to the range -5 to 5 , with intervals of 0.1 .

Questions

Which numbers can appear in I_1 (range and minimum distance to each other), according to the information in Fig. 2.6c?⁵

What is the range of the numbers for x_V in Fig. 2.2 (S), provided that $\text{Min} = 0$ and $\text{Max} = 80$ for the slider in F1:H1. What is the minimum step size?⁶

Connecting a VBA macro to a control element

The following will be important in later chapters: We can connect a VBA routine to the slider (an example is given in Exercise 10.5). A routine `SUB SCROLLBAR1_CHANGE() ... END SUB` is executed each time we move the SCROLLBAR1 slider. To enter a code, click on VIEW CODE (Fig. 2.6b) and select the considered slider's name in the left-hand drop-down window that has popped up, then click CHANGE (or some other action to trigger the routine) in the right-hand drop-down window.

2.3.5 Summary: Cell References and Name Manager

► Ψ *The dollar makes it absolute.*

Cell references, relative, absolute, indirect

The following formulas in a cell, $C3 = [A4]$, $C3 = [\$A4]$, $C3 = [A\$4]$, $C3 = [\$A\$4]$, yield the same result: the value of cell A4 is written into the active cell C3. However, if these formulas are copied into another cell, they change. For example, in D4, there will be: $[B5]$, $[\$A5]$, $[B\$4]$, $[\$A\$4]$. A \$ sign before a column label or a row number has the effect that the label or number are held fixed when copied. A dollar makes a cell reference *absolute*. If the \$ is missing, the label is incremented

⁵ $\text{MIN} = 0$; $\text{MAX} = 100$; integers between 0 and 100 can appear in I_1 , distance = 1.

⁶ $B1 = [(E1-40)/5]$ (see D1), range -8 (for $E1 = 0$) to +8 (for $E1 = 80$) in steps of $1/5 = 0.2$.

by the column or line spacing between the old and new cells; this is a *relative* cell reference.

The spreadsheet function `INDIRECT(CELL)` expects a cell address as an argument. It writes the contents of the cell with this address into the current cell. For example, with $A4 = [=INDIRECT(A5)]$; $A5 = [X7]$; $X7 = [3.4]$ the value in A4 will be 3.4.

Assigning names to cell ranges

$[=a*x + b]$ instead of $[=B\$2*\$A6 + B\$3]$

Individual cells, ranges in a column (“column vectors”) or a row (“row vectors”), and rectangular ranges (“matrices”) can be named and then inserted with their names into formulas and worksheet functions as arguments. Naming is done with the `NAME MANAGER` by going through `FORMULAS/NAME MANAGER`. For details on the `NAME MANAGER`, see below. Make extensive use of this feature! Doing so, you can write formulas and functions in mathematical language, e.g., $[= A*\text{SIN}(k*x)]$ instead of $[=A\$1*\text{SIN}(\$A5*B\$2)]$! Such a formula is valid for a set of sine functions whose amplitudes A and wave number vectors k are stored in two row vectors with names “A” and “k” and where the independent variable x is stored in a column vector with the name “x”.

Name Manager of Excel

As an example to demonstrate the properties of the `NAME MANAGER`, we calculate the electric field E_x in x -direction of a point charge at $(x_1, 0)$ in the xy -plane:

$$E_x = \frac{x - x_1}{\sqrt{(x - x_1)^2 + y^2}} \quad (2.5)$$

The definition of variables and constants and the calculation are distributed over two sheets, “Dist” and “E.x”. The values of $r = \sqrt{(x - x_1)^2 + y^2}$, representing the planar distance to point $(x_1, 0)$ for x and y from -2 to 2, are calculated in Fig. 2.7a (S) (Sheet “Dist”).

The x values are in B3 to F3, the y values in A4 to A8. The value for x_1 is specified in B1. The calculation can be performed with mixed cell references, as in cell B5:

$$B5 = [= \text{SQRT}((B\$3 - \text{Dist}!\$B\$1)^2 + A\$5^2)] \quad (2.6)$$

The formula is, however, more intuitive when cell names are used, as in D6, displayed in D2:

$$D6 = [= \text{SQRT}((x - x.1)^2 + y^2)] \quad (2.7)$$

To achieve this, we have to designate, e.g., range B3:F3 with the name x , already present in cell G3. We activate B3:G3 and follow the menu `FORMULAS/DEFINED NAMES/CREATE FROM SELECTION`. To “activate” the range means

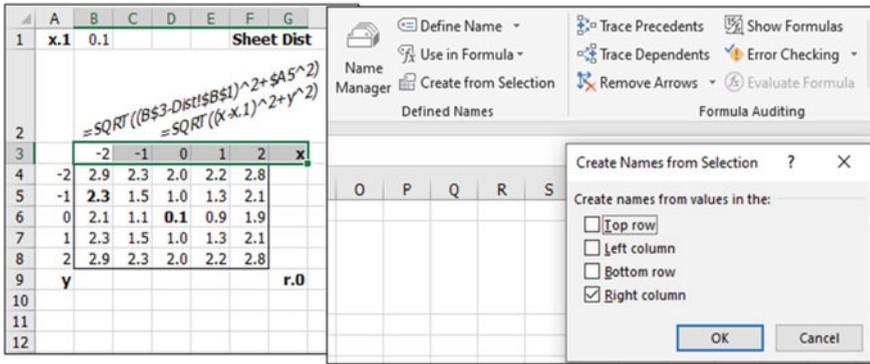


Fig. 2.7 (S) **a** (left) Sheet “Dist”; the values in the matrix B4:F8 are calculated with the values in the horizontal vector $x = B3:F3$ and the vertical vector $y = A4:A8$, together with the value of x_1 in cell \$B\$1, using mixed cell references in B5 and variable names in D6. **b** (right) After activating B3:G3 in **a**, the dialogue box of FORMULAS/ DEFINED NAMES suggests providing B3:F3 with the name in G3 (RIGHT COLUMN)

that the cells are marked with the left mouse button pressed. A window like the one shown in Fig. 2.7b (S) pops up. The assistant has already recognized a potential name in the immediate neighborhood of the activated range, namely, in the right-most column of the activated range. This name corresponds to our intention, and we click OK.

The matrix range B4:F8 is named by activating it and selecting FORMULAS/DEFINED NAMES/DEFINE NAME. A window marked NEW NAME pops up with the REFERS TO field already filled in, because a cell range was activated before the selection. We have to fill in the NAME field, in our case, with “r.0”.

The FORMULAS/DEFINED NAMES/NAME MANAGER window, displayed in Fig. 2.8 (S), gives us an overview of all of the named ranges.

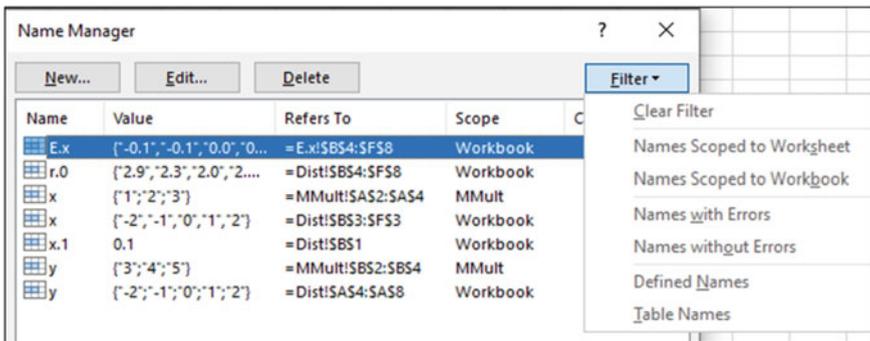


Fig. 2.8 (S) The NAME MANAGER lists all names and ranges (“refers to”) and the scopes for which they are valid

	A	B	C	D	E	F	G		I	J	K	L	M	N	O	
1	Sheet E.x								14							
2	=(x-x.1)/r.0^3									{=(x-x.1)/r.0^3}						
3	###	###	###	###	###	###	#VALUE!	15	as matrix formula							
4	-0.1	-0.1	0.0	0.1	0.1		#VALUE!	16	-0.1	-0.1	0.0	0.1	0.1	#N/A	#N/A	
5	-0.2	-0.3	-0.1	0.4	0.2		#VALUE!	17	-0.2	-0.3	-0.1	0.4	0.2	#N/A	#N/A	
6	-0.2	-0.8	-100	1.2	0.3		#VALUE!	18	-0.2	-0.8	-100.0	1.2	0.3	#N/A	#N/A	
7	-0.2	-0.3	-0.1	0.4	0.2		#VALUE!	19	-0.2	-0.3	-0.1	0.4	0.2	#N/A	#N/A	
8	-0.1	-0.1	0.0	0.1	0.1		#VALUE!	20	-0.1	-0.1	0.0	0.1	0.1	#N/A	#N/A	
9								21	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	
								22								

Fig. 2.9 a (left, S) Sheet “E.x”. The x component of the electric field of a point charge at $(x_1, 0)$ is calculated. In row 3 and column G, the index is outside of the permitted range. **b** (right, S) Alternative calculation in an arbitrary position with a matrix formula

The names are valid in the entire workbook when they are first defined. The names “ x ” and “ y ” appear twice. When they were first defined in the “Dist” spreadsheet, they were valid throughout the workbook, as shown in the SCOPE column. When they are defined a second time in another sheet, here, “MMult”, their scope is limited to this spreadsheet, and the previous definitions of x and y do not apply here, although they still do in the rest of the workbook.

Calculating an electric field

In sheet “E.x” [Fig. 2.9 (S)], the x -component E_x of the electric field is calculated:

$$E_x(x, y) = \frac{x - x_1}{r_0^3}$$

- ▶ **Tim** Which definitions for x , x_1 , and r_0 are valid in sheet “E.x”?
- ▶ **Mag** You can find this out by trial and error or by using FILTER in the NAME MANAGER.⁷

Referring to names in a cell

The ranges (vectors x and y , matrix r_0 , constant x_1) defined in the two sheets can be called in each sheet cell-wise, however, they cannot be so within the complete range of the sheet, but only within a range that matches the range in the sheet in which the name is defined, for example, the matrix r_0 only within the range B4:F8. Outside of this range, errors are reported (see Fig. 2.9a). The horizontal vector x may, for the current definition of its coordinates, only be called in columns B to F, the vertical vector y only in rows 4 to 8. The constant x_1 designates only a single cell and may be called in the whole file without restriction.

⁷ The named ranges in sheet “Dist” are valid throughout the whole workbook, except for “MMult”, so they are also so in “E.x”.

Referring to names in a matrix function

The matrix r_0 can be called within any range of a spreadsheet when entering the formulas as a matrix function, e.g., in I16:M20 in Fig. 2.9b (S). To do so, activate the desired range, write the desired formula into the formula bar, in our case $\{=(x-x.1)/r.0^3\}$, and finish with the “magic chord”! The matrix formula is enclosed in curly brackets: $\{=(x - x.1)/r.0^3\}$.

- ▶ Magic chord for completing matrix functions: Ψ *CTRL* + *SHIFT* + *ENTER*.

2.3.6 What Have We Learned so Far, and How to Proceed Further?

- ▶ **Alac** That’s all really super easy. Worksheet calculation seems to be a children’s game. That wasn’t clear to me until now.

- ▶ **Tim** Well. Might we have acquired knowledge in only a narrow section for particular tasks?

- ▶ **Mag** We have traveled quickly across a wide area on a narrow path. This is actually a fast track to success, at least for the tasks we intend to tackle.

- ▶ **Tim** Is it not better to learn thoroughly so that one does not become lost when the tasks are set a little differently?

- ▶ **Alac** You can always tackle modified tasks through trial and error.

- ▶ **Mag** Yes, trial and error is a possibility. You should do this anyway with all of the programming constructs with which you are not already familiar. Nevertheless, you should also go to a bookstore or into the internet and find books that instruct you in EXCEL. Browse along the learning path you have just gotten to know. It will not take long for you to figure out which of the books explains the computational procedures in a way that you can understand. *You should buy that book!*

The same advice holds for Python, which you will get to know in the next section.

2.3.7 Python Program

Table 2.1 presents a Python program corresponding to Fig. 2.2 (S) for completeness. The reader may study it after having gone through Exercise 2.4.

The program consists of a list of simple assignments of type $x_V = -2$, with the name x_V being called the identifier, or simple formulas assigned to an identifier,

Table 2.1 Specifications corresponding to Fig. 2.2 (S)

1	<code>xV=-2</code>	<code>#Coordinates of vertex</code>
2	<code>yV=6.6</code>	
3	<code>c=-0.1</code>	<code>#Curvature</code>
4	<code>b=-2*c*xV</code>	<code>#Coefficients of standard form</code>
5	<code>a=yV+c*xV**2</code>	
6	<code>dx=0.1</code>	<code>#Horizontal distance between x</code>
7	<code>x=np.arange(-8,8+dx,dx)</code>	<code>#Array of x values</code>
8	<code>yP=a+b*x+c*x**2</code>	<code>#Array of y values</code>
9	<code>print('a ={:5.2f}'.format(a),'; b ={:5.2f}'.format(b))</code>	
a = 6.20 ; b =-0.40		

Table 2.2 Calculating the coordinates (for Fig. 2.1b) of the focus and the directrix

10	<code>xF=xV</code>	<code>#Coordinates of the focus</code>
11	<code>yF=yV+1/4/c</code>	
12	<code>yD=yV-1/4/c</code>	<code>#y Coordinate of directrix</code>

e.g., $b = -2*c*x_V$ recurring to variables specified earlier. The calculated values of a and b (printed into the second cell) are the same as in Fig. 2.2 (S).

The term `'a = {:5.2f}'` indicates that the printout starts with the text string `'a = '` and that the value of the variable a in `format(a)` is to be printed right aligned (:) in float format (f) with length 5 and 2 decimal places (5.2).

The coordinates of the focus and the directrix are calculated in Table 2.2 with the same formulas as in Fig. 2.2 (S). A plot corresponding to Fig. 2.1b is produced with the standard program *FigStd* explained in Sect. 2.4.5.

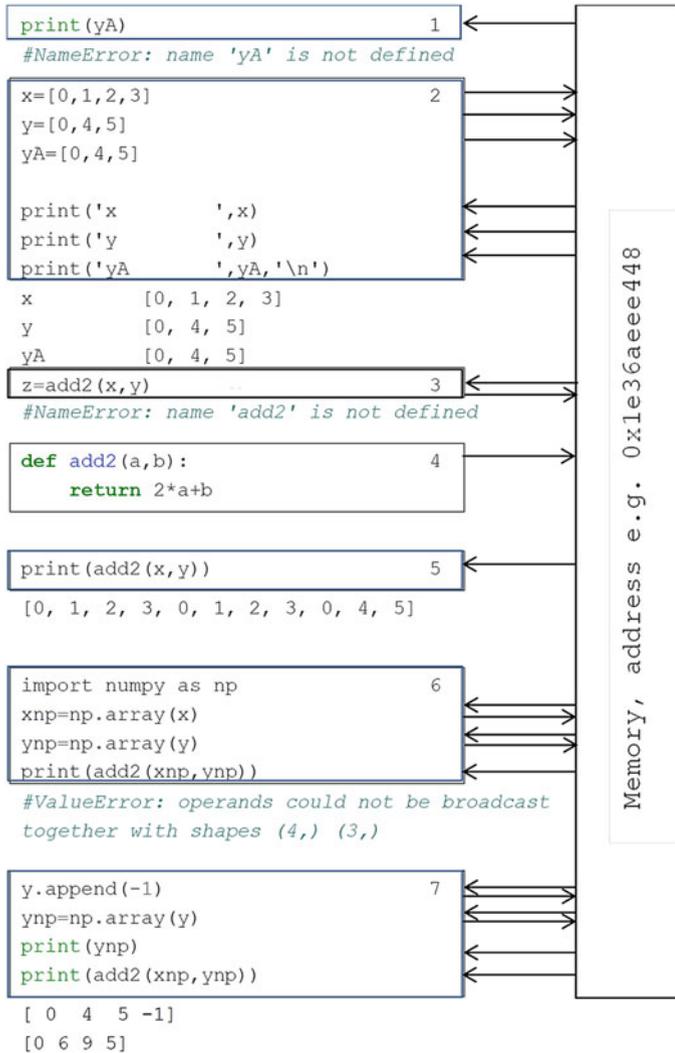
2.4 Python and NumPy Basics

We will learn how to work with program cells in the Jupyter notebook, and get acquainted with arrays in `numpy`. For more detailed information about Python specifics, the reader is referred to *Stewart, J. (2014). Python for Scientists*. Cambridge: Cambridge University Press. <https://doi.org/10.1017/CBO9781107447875>, Chap. 3 (*A Short Python Tutorial*), Chap. 4 (*NumPy*), and Chap. 5 (*Two-Dimensional Graphics*) or other textbooks treating the same subjects.

2.4.1 Basic Exercise

Table 2.3 displays a simple Python program developed in the Jupyter notebook.

Table 2.3 Cell structure of Python in the Jupyter notebook



We can start running the program with any cell. However, if referring to variables or functions, we have to define them ahead of time. Starting Cell 1 before Cell 2, or Cell 3 before Cell 4, leads to a *#NameError*.

We are now running the program cell by cell in sequence.

Cell 1: The print statement tries to get y_A from the memory. As y_A is not yet defined, a *#NameError* message is returned.

Cell 2: Three lists, characterized by square brackets, are defined, and their contents are printed out. When we now run Cell 1 again, no error message appears, but the contents of y_A are printed out.

Cell 3: The statement tries to assign a value to variable z by calling a function $add2$ that is not yet defined, so that a name error occurs.

Cell 4: The function $add2$ is defined. Now, running Cell 3 does not result in an error message.

Cell 5: Function $add2$ is applied to variables x and y , and the result is printed out. When applied to lists, the formula $2*a + b$ is not an algebraic function; list x is concatenated twice and then list y once to yield the list $[x, x, y]$.

Cell 6: The `numpy` library is imported under the abbreviation `np`. Two `numpy` arrays are created from the lists x and y . The function $add2$ is applied to x_{np} and y_{np} , yielding an error message. When applied to arrays, it is the algebraic operations addition and multiplication that are performed element-wise. As the two arrays have different size, this does not work, and a `#ValueError` message results: “Operands could not be broadcast together with shapes (4,) and (3,)”.

Cell 7: List y is extended with a new element to have the same length as x_{np} . The function $add2$ can now be applied, with the result now being an array of the same length and a linear combination of the two initial arrays. The formula $2*a + b$ is now interpreted as an algebraic operation and performed element-wise on the arrays.

Copy

Parts of the memory may get various names, e.g., the statement $AddTwo = add2$ assigns the additional identifier $AddTwo$ to the function $add2$ defined in Cell 4 of Table 2.3, so that this function may also be called by $AddTwo$. Such assignments are different from making a copy, as is demonstrated in Table 2.4. The statement $YA = Y[3:5]$ produces only a name (an identifier) for part of list Y , whereas with $YC = np.copy(Y[3:5])$, a new object with its own memory space is created. YC is an object of its own and is not affected by subsequent changes of Y .

Table 2.4 Identifier of a subarray vs. copying a subarray into a new object; only one print statement is reported in the first cell; the other print statements are similar. The results of all print statements are reported in the second and the fourth cells

1	<code>Y=np.array([0,1,2,3,4,5,6,7,8,9])</code>	
2	<code>YA=Y[3:5]</code>	
3	<code>YC=np.copy(Y[3:5])</code>	
4	<code>print("Y ", Y)</code>	
5	<code>Y</code>	<code>[0 1 2 3 4 5 6 7 8 9]</code>
6	<code>YA</code>	<code>[3 4]</code>
7	<code>YC</code>	<code>[3 4]</code>
8		
9	<code>Y[3]=8</code>	<code>#Another program line</code>
Y	<code>[0 1 2 8 4 5 6 7 8 9]</code>	
YA	<code>[8 4]</code>	
YC	<code>[3 4]</code>	<code>#YC is independent of Y</code>

2.4.2 Data Structures

Data types

The following data types are available in Python:

- Int (integer, unlimited size),
- float (8 Bytes),
- bool (boolean),
- string (text),
- complex (only in numpy).

Lists

Lists contain one or more items. They:

- may contain items of different type: `z = [1, 3.14, 'abc' . 2 + 4]`,
- are mutable, e.g., can be changed by `append` or `delete`;
`x.append('new') → [x, 'new']`,
- can be concatenated by `+`: `[x] + [y] → [x, y]`,
- can be concatenated by `*`: `3*[x] → [x, x, x]`,
- can be sliced, i.e., subarrays can be addressed with new identifiers: `x2 = [3:-1]`, from the fourth (indices 0,1,2,3,...) to the penultimate element of `x`.

Append, *add*, and *multiply* result in longer lists. The new identifier `x2` points to a sublist of `x` from the 3rd to the penultimate (`-1`) element. The method `append`, as well as the operators `+` and `*`, are demonstrated in Table 2.3 (*def add2*). Lists can be multidimensional, but in our exercises, only one- and two-dimensional lists are used.

Sets {}

Sets contain unordered collections of unique elements to which standard mathematical set operations can be applied. These are *intersection*, *union*, *difference*, and *symmetric difference*. Examples are given in Exercise 8.4. Sets do not record element position and, as a consequence, do not support indexing, slicing, or other sequence-like behavior.

Dictionaries {}

Dictionaries comprise pairs of an identifier (data type string) and an object (arbitrary data type). The objects are addressed by their identifiers:

```
- lis={"second": 1.234, "first": 5.678}
  print(lis['first']) → 5.678
```

We do not use dictionaries in this book.

2.4.3 Python Libraries

Python libraries are collections of pre-compiled functions. They are open-source, supported by a community of programmers, who are always there to answer questions, e.g., on *stackoverflow.com*. We shall sometimes directly refer to such advice.

NumPy

NumPy is a fast and efficient array-processing package designed for numerical computing that provides functionalities comparable to MATLAB. We are using it as our standard. It is usually imported under the name *np*: `import numpy as np`, see Cell 6 in Table 2.3.

NumPy.random

Random number routines produce pseudo-random numbers. We import `NumPy.random` as `npr` and make use of two functions:

- `npr.rand(N)` generates an array of N random numbers between 0 and 1, (Exercises 8.2 and 8.3).
- `npr.randn(N)` generates an array of N normally distributed random numbers (Exercise 8.5).

Matplotlib

Matplotlib is a package for designing a variety of charts or even arrays of charts. We import `matplotlib.pyplot` as `plt`. We restrict ourselves to producing simple scatterplots with a user-defined function (Sect. 2.4.5) that is able to display nearly all results of our exercises, similar to EXCEL charts. In order to plot arrows, we introduce a second user-defined function (Sect. 3.2.7).

Scipy

Scipy is designed for scientific computing and is especially suited for machine learning. We need only certain functions for optimization [`minimize` (Exercise 10.6), `fsolve` (Exercise 10.5), `curve_fit` (Exercise 10.7)], linear algebra [`solve` (Exercise 5.9)], and statistics (`ChiSquare` from `Scipy.stats`).

Pandas

Pandas mimics spreadsheet calculation within Python, enabling input to and output from EXCEL files and text files. We use it only occasionally, e.g., in Exercise 4.8, to make the reader aware that such things exist and are useful.

Table 2.5 Data types in numpy demonstrated with arrays built with lists from Cell 2 of Table 2.3

1	import numpy as np	
2	xInt = np.array(x, dtype=int)	
3	yFloat = np.array(y, dtype=float)	
4	yAfloat = np.array(yA, dtype=float)	
5	xBool = np.array(x, dtype=bool)	
6	xComplex = np.array(x, dtype=complex)	
7	xStr = np.array(x, dtype=str)	
xInt	[0 1 2 3]	shape (4,)
yFloat	[0.00 4.00 5.00 -1.00]	shape (4,)
yAfloat	[0.00 4.00 5.00]	shape (3,)
xBool	[0 1 1 1]	
xComplex	[0.+0.j 1.+0.j 2.+0.j 3.+0.j]	
xStr	['0' '1' '2' '3']	
xInt*yFloat	[0. 4. 10. -3.]	
xInt*yAfloat	ValueError: operands could not be broadcast together with shapes (4,) (3,)	

2.4.4 Numpy Constructions

Numpy is usually imported under the name *np*: `import numpy as np`.

Ndarrays

Ndarrays are, in general, *n*-dimensional arrays. One- and two-dimensional ones have analogies in spreadsheets. For scientific computing, they have an advantage over spreadsheets when the data becomes large.

They:

- are immutable, with size and datatype specified when the object is introduced,
- are operated element-wise in algebraic operations: $x + y \rightarrow [x + y]$,
- can be arguments in mathematical functions.

In Table 2.5, one-dimensional arrays of various data types are built with the function `np.array(.)` expecting a list as an argument, here taken from Table 2.3. Two numerical arrays can be multiplied element-wise when they have the same shape, e.g., `xInt` and `yFloat`, but not `xInt` and `yAfloat`.

Question

How do the entries of `xBool` in Table 2.5 arise?⁸

⁸ Numerical 0 becomes False, $\neq 0$ becomes True. False is output as 0, True as 1.

Ab-initio constructors

The following five functions construct an array from the specifications in the argument list.

- `np.ones(N)` 1D-array of N ones
- `np.zeros(N)` 1D-array of N zeros
- `np.linspace(start, stop, number of steps, endpoint = True)`
`np.linspace(1.5, 3.5, 3) → [1.5 2.5 3.5]`, True is default.
`np.linspace(1.5, 3.5, 2, endpoint = False) → [1.5 2.5]`
- `np.logspace`
`np.logspace(1, 3, 3, base = 10) → [10. 100. 1000.]`
- `np.arange(start, stop, step)`
`np.arange(0.0, 4.5, 1.5) → [0.0 1.5 3.0]`

In `np.arange`, the stop value is not included. This is favorable when stacking `np.arange`s together:

```

_      np.hstack([np.arange(1, 4, 1), np.arange(4, 10, 2)])
_      → [1 2 3 4 6 8]

```

We can start the second `np.arange` with the stop of the first `np.arange` and have the stop/start value only once.

The following two functions construct arrays of the same shape as *Array* in the argument:

- `np.ones_like(Array)` 1D-array of ones
- `np.zeros_like(Array)` 1D-array of zeros.

From the `numpy.random` library, imported as `npr`, we shall use:

- `npr.rand(N)` generating an array of N random numbers between 0 and 1.
- `npr.randn(N)` generating an array of N standard normally distributed random numbers.

Identifiers to elements of 2-dimensional arrays

In Table 2.6, we construct 1-dimensional arrays with `np.linspace` and stack them together to form a 2-dimensional array consisting of two rows. Furthermore, identifiers to single rows, elements, columns, and 2D subarrays are specified.

Table 2.6 Constructing a 2D array with two 1D arrays obtained with `linspace`; identifiers to single rows, elements, columns, and 2D subarrays; only one print statement reported in Cell 1

1	<code>X=np.linspace(0,3,4)</code>
2	<code>Y=np.linspace(10,13,4)</code>
3	<code>Lis=np.array([X,Y])</code>
4	<code>print("np.shape(Lis) ", np.shape(Lis))</code>
<pre> np.shape(Lis) (2, 4) #2 Rows with 4 entries each Lis [[0.00 1.00 2.00 3.00] [10.00 11.00 12.00 13.00]] Lis[0] [0.00 1.00 2.00 3.00] Lis[0][2] 2.0 Lis[:,1] [1.00 11.00] Lis[0][2:3] [2.00] Lis[1][1:3] [11.00 12.00] Lis[0:1][1:3] [[1.00 2.00] [11.00 12.00]] </pre>	

- Recommendation: Use this type of presentation, printing out the shape and content of matrices, to get an overview of your data structure!

`Lis[0][2]` means: from the first row (index 0), take the third element (index 2).
Alternative interpretation: take the matrix element from row = 0, column = 2.

`Lis[1][1:3]` means: from the second row (index 1), take the second (index 1) to the third (index 2, index 3 exclusive) elements.

`Lis[0][1:-1]` means: from the first row (index 0), take the second (index 1) to the penultimate (index -1) elements.

`Lis[:,1]` means: take the second column (index 1).

Information on arrays

- `np.size(array)` returns the total number of elements.
- `np.shape(array)` returns the shape, for 2-dimensional arrays in the form (r,c) , number r ; c of rows and columns, respectively.

Stacking

- `np.hstack([list of arrays])` concatenates the 1D-arrays horizontally into a long 1D-array, is defined more generally for multidimensional arrays
- `np.stack([list of 1D-arrays])` stacks the arrays (all of the same size by necessity) in the list as rows into a 2D-array, is defined more generally for multidimensional arrays.

Functions

- `np.sin`, `np.cos`, `np.tan`, `np.atan2`
- `np.power`
- `np.dot`.

The function `np.dot(A, B)` performs a matrix multiplication of the two matrices `A` and `B`. The same can be achieved with the operator `@`: `A@B`. Matrix multiplication is dealt with in Exercise 2.5 in detail.

`Atan2` is *arcus tangens* calculating the angle from the x -axis given the point (x, y) . The order of arguments is `atan2(y;x)`, different from EXCEL:

```
np.arctan2(y, x) (Python) = ATAN2(X; Y) (EXCEL)
np.arctan2(3, 2) = 0.980.98 = ATAN2(2; 3).
np.arctan2(2, 3) = 0.590.59 = ATAN2(3; 2).
np.arctan2(4, 1) = 1.331.33 = ATAN2(1; 4).
```

2.4.5 Standard Plot Program

We create charts by using the library `matplotlib.pyplot` imported under the name `plt`. As we deal almost exclusively with scatter diagrams for functions of the form $y = f(x)$, we do not need the full versatility of `matplotlib`. To profit from this restriction, we have devised a function to plot a standard figure, `FigStd`, in Table 2.7, where all necessary style information is coded within the function

Table 2.7 User-defined function for creating a scatter plot; this cell is run in all programs at the beginning

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 np.set_printoptions(precision=2, threshold=10, edgeitems=3,
4                     formatter={'float': '{: 7.2f}'.format})
5
6 def FigStd(xlabel, xmin, xmax, dx,
7           ylabel, ymin, ymax, dy,
8           xlength=4, ylength=4):
9     plt.figure(figsize=(xlength, ylength))
10    plt.axis([xmin, xmax, ymin, ymax])
11    plt.rcParams.update({'font.size': 10,
12                        'font.style': 'italic'})
13    plt.xlabel(xlabel)
14    plt.xticks(np.arange(xmin, xmax+dx, dx))
15    plt.ylabel(ylabel)
16    plt.yticks(np.arange(ymin, ymax+dy, dy))
17    plt.plot([xmin, xmax], [0, 0], 'k-', lw=1) #x Axis through 0
18    plt.plot([0, 0], [ymin, ymax], 'k-', lw=1)
```

Fig. 2.10 Parabola (the same as in Fig. 2.1b) specified in Tables 2.1 and 2.2 and plotted with Table 2.8

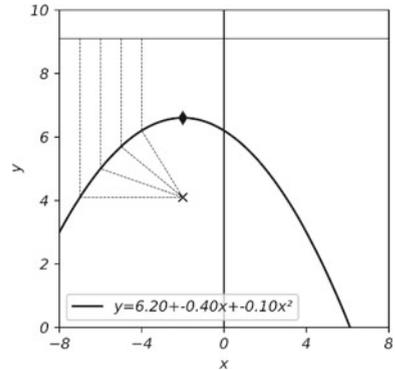


Table 2.8 Applying our standard function for scatter plots

```

1 FigStd('x', -8, 8, 4, 'y', 0, 10, 2)
2 lblP="y=%4.2f+%4.2fx+%4.2fx^2"%(a,b,c) #Label for parabola
3 plt.plot(x,yP,'k-',label=lblP)      #'k-' Black full line
4 plt.plot(xV,yV,'kd')                #'kd' Black diamond
5 plt.plot([-8,8],[yD,yD],'k-',lw=0.5)
6                                     #'lw' Line width
7 plt.plot(xF,yF,'kx')
8 for i in range(-7,-3,1):
9     plt.plot([i,i,xF],[yD,a+b*i+c*i**2,yF],
10             'k--',lw=0.5)          #'k--' Black dashed line
11 plt.legend()#Plots the labels for the curves within the fig

```

body, and the information concerning the x -axis and the y -axis has to be specified in positional arguments. The axes' lengths in the figure have default values ($xlength = 4$, $ylength = 4$) that can optionally be specified otherwise.

The plot in Fig. 2.10, corresponding to Fig. 2.1b, is produced with the program in Table 2.8 (continuation of Table 2.1 and Table 2.2).

In the instruction $[yD, a + b*i + c*i**2, yF]$, the values are created implicitly within the list.

Extensions

In Sect. 3.4.7, we introduce two extensions of our standard figure: a secondary y -axis, a logarithmic scaling of the y -axis with the statement `plt.yscale(value = "log")`.

Table 2.9 Formatted output with %

1	<code>a1, a2, a3=1.2345, 3.4567, -4.5678</code>
2	<code>label="y=%4.2f*x**%4.2f + %6.1f"%(a1, a2, a3)</code>
3	<code>print(label)</code>
<code>y=1.23*x**3.46 + -4.6</code>	

2.4.6 Formatted Output

The example in Table 2.9 shows how a formatted output can be achieved. There is a format string with three text variables “`y = $*x**$ + $`”. The \$ here is a proxy; it starts with a “%” and is followed by a format, e.g., `4.2f` for a floating number 4 characters long and to be displayed with 2 decimal places. Following the string is a % sign and the name of the variables as a tuple, here `%(a1, a2, a3)` with the three entries replacing the three % in the format string.

2.5 Matrix Calculations in Excel and Python

We define row and column vectors and 2-dimensional matrices, and explain operations on them, parallel in EXCEL and Python. We apply broadcasting in EXCEL and Python to adapt the shape of operands so as to fit with each other in the intended operation. Finally, we get to know and apply the matrix operations transposition and inversion of linear algebra.

2.5.1 Data Structure and Nomenclature

Data structure

The data structure in scientific computing is based on arrays. Python, especially its `numpy` library, is designed for vectorized code, so that the programmer is forced to work with it from the first line of a code. This is its core business, described in all introductory textbooks, so we do not need to explain it here in detail.

Spreadsheet software, in contrast, is primarily designed for business calculations. In order to make it suitable for scientific computing, we have to create said vectorized data structure. This is achieved by identifying cell ranges with names so that most of the operations on vectors and matrices known from Python programs can also be applied in spreadsheet calculations. This is the essence of our approach to making spreadsheet calculations suitable for elementary scientific computing.

The treatment in this book is restricted to 1-dimensional arrays called vectors and 2-dimensional arrays called matrices. Vectors are of two types, row and column. We designate the shape of vectors and matrices with a tuple (r, c) , where r and c are the number of rows and columns, respectively. In Python, a row vector is regarded as

a 1-dimensional array, e.g., of shape (2,), whereas a column vector is represented as a 2-dimensional array with only one column, e.g., of shape (3, 1).

Nomenclature

We not only strive to use the same notation of variables in Python and EXCEL, but also to use a similar one in mathematical equations in the text. For this exercise, we have chosen the following nomenclature:

<u>U</u> , <u>V</u>	column vectors, shape (3, 1) 3 rows, 1 column
<u>W</u> , <u>X</u>	row vectors, shape (2,) 2 entries in the array
<u>L</u> , <u>M</u>	matrices, shape (3, 2) 3 rows, 2 columns
<u>N</u> , <u>O</u>	square matrices, shape (2, 2).

In the text, vectors are characterized by an underline, matrices by a double underline. Additionally, we apply subscripts and underlines to make the variable names more similar to the usual mathematical notation.

2.5.2 Operations on Arrays

Element-wise operation

Functions with scalar arguments, e.g., $\cos(x)$, are applied element-wise on vectors and matrices, resulting in an output of the same shape as the argument. The same holds for the algebraic operations addition and multiplication on arrays of the same shape. Function names and operations are usually the same in EXCEL and Python, with one important exception: *arcus tangens*, as already mentioned in Sect. 2.4.4.

Broadcasting

Broadcasting is a technical term for an essential tool in Python, but the underlying operations are also available in EXCEL. If two operands of an operation are of different but compatible shapes, they can be broadcast together. For a row vector of size n and a column vector of size m , this is done by repeating the row vector vertically and the column vector horizontally so as to get the same shape as an (n, m) -matrix. When such vectors are, e.g., multiplied, the result is a matrix of shape (n, m) , with the elements being the element-wise product of the broadcast matrices.

Also, other operations, such as the multiplication of two vectors or the exponentiation of a vector with another vector, are performed as if each of the operands were repeated, so that matrices with a common shape are obtained that are then processed element-wise.

Linear Algebra

Functions applied in linear algebra comprise *determinant*, *inverse*, and *matrix multiplication*. Their names are listed in Table 2.10.

Table 2.10 Functions of linear algebra

EXCEL	Python
TRANSPOSE(<u>M</u>)	<code>np.transpose(M)</code>
INDEX(<u>M</u> ; <i>r</i> ; <i>c</i>)	<code><u>M</u>[<i>r</i>, <i>c</i>] or <u><u>M</u></u>[<i>r</i>][<i>c</i>]</code>
Indexing starts in EXCEL with 1, and in Python with 0 in square brackets:	
EXCEL: INDEX(<u>W</u> ;1)	Python <code><u>W</u>[0]</code>
MDETERM(<u>N</u>)	<code>np.linalg.det(<u>N</u>)</code>
MINVERSE(<u>N</u>)	<code>np.linalg.inv(<u>N</u>)</code>
MMULT(<u>M</u> ₁ , <u>M</u> ₂)	<code><u>M</u>₁ @ <u>M</u>₂ or np.dot(<u>M</u>₁,<u>M</u>₂)</code>

In Python, a matrix is constructed as a list of row vectors. So, indexing `M[2][0]` points to the first element [0] in the third row `M[2]`. The abbreviation stands for `numpy.linalg` that is imported under that name.

Spreadsheet first

We start our overview of matrix calculations with spreadsheet constructs because the data structure there is immediately visible on the screen.

2.5.3 Matrices in Spreadsheets

Row and column vectors, matrices

In the upper half of Fig. 2.11 (S), two column vectors U, V of shape (3,1), two row vectors W, X of shape (2,), and two matrices L, M of shape (3,2) are defined, using the notation for shapes in Python. In the lower half of the figure, elements of the defined entities are singled out, with spreadsheet functions INDEX(VECTOR; INDEX) and INDEX(MATRIX; ROW INDEX; COLUMN INDEX). In the last two rows, the scalar product of the vectors W and X is calculated in Cartesian and polar coordinates.

The length *l* of a vector is determined with the equation $l = \sqrt{\sum x_i^2}$, obtained by nesting two functions: `SQRT(SUMSQ(W))`. A 2-dimensional vector can be interpreted as a straight line in the *xy*-plane directed from (0,0) to (*x*,*y*). Its angle to the horizontal is determined with the *arcus tangens* function, = `ATAN2(x,y)`, which, for vector W, reads as = `ATAN2(INDEX(W;1);INDEX(W;2))`.

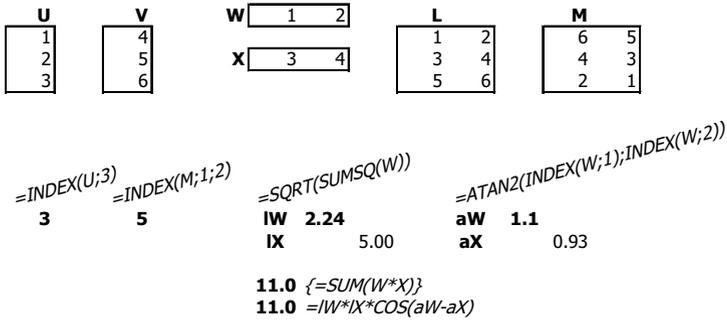


Fig. 2.11 (S) Column vectors U , V , row vectors W , X , and matrices L , M are defined as named ranges, and some operations are applied to them in the lower half of the figure

Broadcasting

Addition and multiplication of a row vector and a column vector, and of two matrices, is demonstrated in Fig. 2.12 (S).

In the upper half of Fig. 2.12 (S), two vectors are added with the operator $+$ or multiplied with the operator $*$. When the operations are performed on two vectors of the same type, row or column, they must be of the same size. The operations are then performed element-wise, and the result is a vector of the same shape. Activating a larger range leads to a repetition of the 1-dimensional result [see $U*V$ in Fig. 2.12 (S)]. Algebraic operations on matrices of the same shape result in a matrix of the same shape (see $L*M$).

All operations in Fig. 2.12 (S) are of the spreadsheet matrix type, as indicated by the fact that they are enclosed in curly brackets. To recall: You have to activate a range of size suitable for the result, enter the formula, and finish with the magic chord: Ψ *Ctrl*, *Shift*, *Enter*.

Multiplying or adding a column vector (r), (e.g., V) and a row vector (c), (e.g., W) results in a matrix of shape $(r;c)$ (see the results for $W*V$ and $V+W$). The operations are performed as if each of the operands is repeated, so that matrices with a common shape are obtained that are then multiplied or added element-wise.

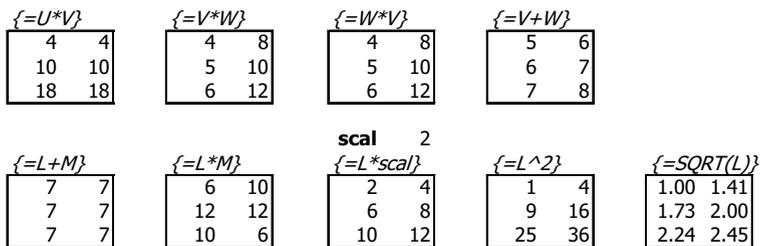


Fig. 2.12 (S) Arithmetic operations on vectors and matrices defined in Fig. 2.11 (S)

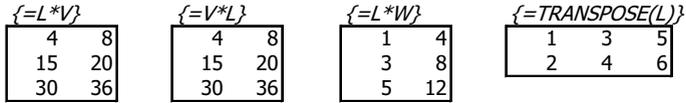


Fig. 2.13 (S) Operations on vectors and matrices with different shapes, defined in Fig. 2.11 (S) after broadcasting

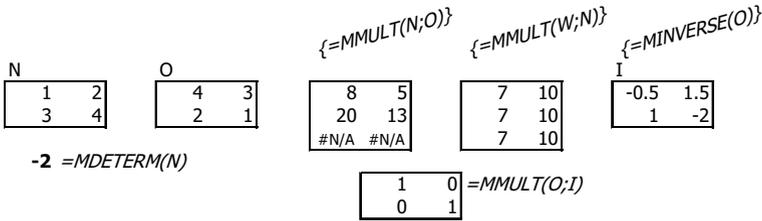


Fig. 2.14 (S) Mathematical matrix operations on the square matrices *N* and *O*

The same holds for the multiplication of a vector or a matrix with a scalar (see $\underline{L} * \text{scal}$). Functions applied to vectors or matrices are also performed element-wise and yield a range of the same shape as the argument (see $\{=\underline{L}^2\}$ or $\{=\text{SQRT}(\underline{L})\}$).

In Fig. 2.13 (S), operations combining a vector and a matrix are performed. Before an element-wise operation, the vectors are repeated so as to obtain the same shape as the matrix that also becomes the result's shape.

A matrix is transposed with the function $\{=\text{TRANSPOSE}(\text{Matrix})\}$. In Fig. 2.13 (S), this is applied to \underline{L} .

Operations of Linear Algebra

Operations and functions on mathematical matrices are shown in Fig. 2.14 (S). On square matrices, the determinant can be obtained with MDETERM and its inverse matrix with MINVERSE.

Two matrices of suitable shapes, (n_1, n_2) and (n_2, n_3) , can be multiplied with MMULT (*Matrix1*, *Matrix2*); the result is of shape (n_1, n_3) . Multiplying the original matrix *O* with its inverse *I* yields the unit matrix.

2.5.4 Matrices in Python

Before operating with `numpy`, we have to import this library (see line 1 in Table 2.11, where we also set the `printoptions` for arrays that are to be printed, here, in the float format 0.2f with 2 decimal places and one blank between the printed numbers. The “0” in 0.2f indicates that the number of characters is not fixed but adapted to the current number. Additional separating blanks are obtained by inserting them between ‘ and {, or after {: as in line 3. In the following, print

Table 2.11 Importing numpy and setting a print option for arrays

<pre> 1 import numpy as np 2 np.set_printoptions(formatter={'float': '{: 0.2f}'.format}) 3 x=np.linspace(0,10,4) 4 print('x \n',x) #\n Makes a line feed </pre>
<pre> x [0.00 3.33 6.67 10.00] </pre>

statements similar to that in line 5 of Table 2.11 are not explicitly reported, but only the results, as in the bottom cell of Table 2.11. The key word “\n” induces a line feed.

Row vectors

In Table 2.12, row vectors \underline{W} and \underline{X} are created as an array with the same elements as in Fig. 2.11. The length of \underline{W} , calculated correspondingly with $\text{np.sqrt}(\text{np.sum}(\underline{W}*\underline{W}))$, is the same. The scalar product of two vectors can be calculated with the function np.dot or with the operator $@$ for matrix multiplication: $\underline{W} @ \underline{W}$.

Column vectors

Row vectors can be transformed into column vectors by transposition, most simply with the function $\text{np.transpose}()$, but also with the extension $\text{.transpose}(1, 0)$ fixed to the identifier of a vector (or, more generally, of a matrix) (see Table 2.13). The original vector \underline{U}_R is not defined as a row vector, but as a matrix with only one row, characterized by enclosing the list within double square brackets, $[[\dots]]$. This (1, 3)-matrix can be transposed with $\text{np.transpose}(\underline{U}_R)$ or $\underline{U}_R.\text{transpose}(1, 0)$ to become a (3,1)-matrix equivalent to a column vector. The transpose operation can also be applied to more-dimensional matrices.

In said table, we also demonstrate the element-wise multiplication of two column vectors \underline{U} , \underline{V} and the broadcasting that occurs when a row vector (\underline{W} from Table 2.12) and a column vector \underline{V} are multiplied. An instruction $[1,2,3]*[1,2]$ results in an error message: “Operands could not be broadcast together”. The operation $*$ is commutative: $\underline{W}*\underline{V} = \underline{V}*\underline{W}$.

Table 2.12 Creating row vectors in Python and determining their scalar product; print statements are not reported in Cell 1, and only their results are reported in Cell 2

<pre> 1 W = np.array([1, 2]) 2 X = np.array([3, 4]) 3 lW = np.sqrt(np.sum(W*W)) 4 ld = np.dot(W, W) </pre>	<pre> W, np.shape(W) [1 2] (2,) lW 2.24 ld 5 1 W @ W 5 </pre>
--	--

Table 2.13 Creating column vectors in Python, multiplication of two column vectors and multiplying column vector \underline{V} with row vector \underline{W} (from Table 2.12). The entries in the lower cells are obtained by print statements in the top cell that are not reported there, e.g., `print("np.shape(UR)", np.shape(UR))`

<pre> 1 UR = np.array([[1, 2, 3]]) 2 U1 = np.transpose(UR) 3 U = UR.transpose(1, 0) 4 V = np.array([[4, 5, 6]]).transpose(1,0) </pre>	
<pre> np.shape(UR) (1, 3) UR [[1 2 3]] </pre>	<pre> V [[4] [5] [6]] </pre>
<pre> np.shape(U1) (3, 1) U1 [[1] [2] [3]] </pre>	<pre> U*V [[4] [10] [18]] </pre>
<pre> np.shape(U) (3, 1) U [[1] [2] [3]] </pre>	<pre> V*W [[4 8] [5 10] [6 12]] </pre>
<pre> U[0][0] [1] </pre>	<pre> np.sum(U*V) 32 </pre>

Matrices

In Table 2.14, we see how 2-dimensional matrices are created with the function `np.array(...)`. The argument of `np.array` is a list of 1-dimensional lists all of the same size. Algebraic operations such as $\underline{L} * \underline{M}$ on matrices with equal shape are again performed element-wise.

In Table 2.15, multiplication of \underline{L} [shape (3,2)] with \underline{V} [shape (3,1)] and \underline{W} [shape (2,)] is demonstrated. These are some further examples of broadcasting. The effect of matrix transposition, `np.transpose(L)`, is also reported. Shape (3,2) indicates 3 rows with 2 elements each, and may also be interpreted as a matrix with 3 rows and 2 columns.

Linear Algebra

In Table 2.16, we learn about linear algebra operations on square matrices (shape (n, n)) using the functions `inv` and `det` of the `linalg` sublibrary of `numpy` that is often imported as `np`. `np.linalg.det` returns the determinant and `np.linalg.inv()` the inverse of a square matrix.

Table 2.14 Creating 2-dimensional matrices in Python and performing algebraic operations on them; the entries in the bottom cells are obtained by print statements in the top cell that are not reported there, e.g., `print("L\n", L)`

<pre>1 L = np.array([[1, 2], [3, 4], [5, 6]]) 2 M = np.array([[6, 5], [4, 3], [2, 1]])</pre>	
<pre>np.shape(L) (3, 2)</pre>	<pre>L*M [[6 10] [12 12] [10 6]]</pre>
<pre>L [[1 2] [3 4] [5 6]]</pre>	<pre>L**2 [[1 4] [9 16] [25 36]]</pre>
<pre>M [[6 5] [4 3] [2 1]]</pre>	<pre>np.sqrt(L) [[1.00 1.41] [1.73 2.00] [2.24 2.45]]</pre>

Table 2.15 Product of a 2-dimensional matrix (L defined in Table 2.14) with 1-dimensional vectors V, W defined in Tables 2.13 and 2.12, respectively; the entries are obtained with, e.g., `print("L*V\n", L*V)`

<pre>L [[1 2] [3 4] [5 6]]</pre>	<pre>L*V [[4 8] [15 20] [30 36]]</pre>
<pre>V [[4] [5] [6]]</pre>	<pre>L*W [[1 4] [3 8] [5 12]]</pre>
<pre>W [1 2]</pre>	<pre>np.transpose(L) [[1 3 5] [2 4 6]]</pre>

The determinant of matrix N is $1 \cdot 4 - 2 \cdot 3 = -2$. Python deviates from that value in the 16th decimal. This is because the calculations are performed in the binary system.

The operator `@` stands for matrix multiplication. The matrix product of a matrix with its inverse yields the unit matrix (see lines 18 to 20 in Table 2.16). The difference to the algebraic multiplication with the operator `*`, operating element-wise, is demonstrated in Table 2.17, $L*W$ vs. $L@W$.

Table 2.16 Operations of linear algebra on square matrices

1	<code>N=np.array([[1, 2], [3, 4]])</code>
2	<code>O=np.array([[4, 3], [2, 1]])</code>
3	<code>I=np.linalg.inv(N)</code>
N	<code>np.linalg.det(N)</code> -2.0000000000000004
	<code>[[1 2]</code> <code>[3 4]]</code>
O	<code>N @ I</code> <i>#Mathematical matrix multiplication</i> <code>[[1.00 0.00]</code> <code>[0.00 1.00]]</code>
	<code>[[4 3]</code> <code>[2 1]]</code>
I	<code>[[-2.00 1.00]</code> <code>[1.50 -0.50]]</code>
	<code>[[-2.00 1.00]</code> <code>[1.50 -0.50]]</code>

Table 2.17 Array multiplication with * versus mathematical matrix multiplication with @

L	L*W
<code>[[1 2]</code> <code>[3 4]</code> <code>[5 6]]</code>	<code>[[1 4]</code> <code>[3 8]</code> <code>[5 12]]</code>
W	L @ W
<code>[1 2]</code>	<code>[5 11 17]</code>

Questions concerning Tables 2.15, 2.16, and 2.17

- What are the shapes of L , W , $L*W$, `np.transpose(L)`? ⁹
- What is the shape and the type of $N@I$?¹⁰
- What are the shapes of L , W , $L@W$?¹¹

2.6 Four Parabolas and Their Upper Envelope

Four parabolas are represented in a figure, together with their upper envelope, first produced by a spreadsheet calculation and then by an analogous Python program intended to serve as a basic exercise in Python programming and developed step by step in great detail.

⁹ `Shape(L) = (3, 2)`, `shape(W) = (2,)`, `shape(L*W) = (3, 2)`, `shape(np.transpose(L)) = (2, 3)`.
¹⁰ $N@I$ is a diagonal matrix, `shape = (2, 2)`.
¹¹ `Shape(L) = (3, 2)`, `shape(W) = (2,)`, `shape(L@W) = (3,)`.

2.6.1 Graphical Representation

Figure 2.15 shows four parabolas with their vertices marked with diamonds and their upper envelope always running on the, at the respective position, top parabola.

The spreadsheet for producing this figure is given in Fig. 2.16 (S). It is briefly described in the following section on the basis of Exercise 2.2. The corresponding

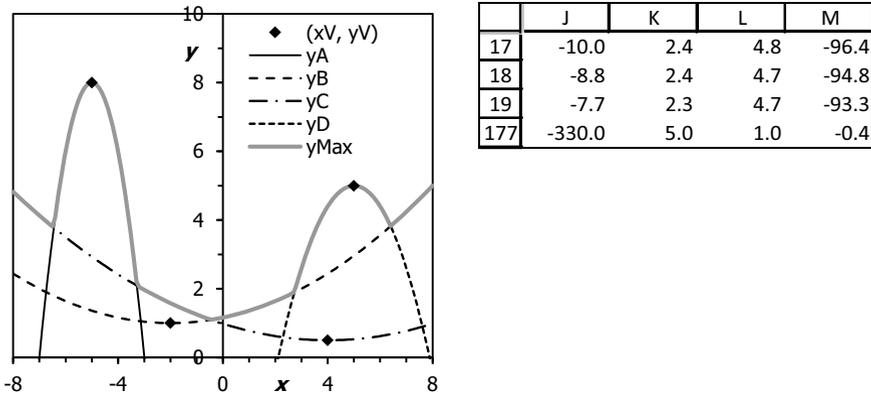


Fig. 2.15 a (left) Four parabolas for the parameters specified in Fig. 2.16 (S). b (S, right) Coordinates for the four parabolas obtained with a matrix formula, J17:M177 = $\{[-a + b*x + c_*x^2]\}$

	A	B	C	D	E	F	G	H
1	xV	-5	-2	4	5	(xV, yV)		
2	yV	8	1	0.5	5			
3	c_	-2	0.04	0.03	-0.6			
4	b	-20	0.16	-0.24	6			
5	a	-42	1.16	0.98	-10			
6	dx	0.1						
7	$=A9+dx$	$=B\$5+B\$4*\$A9+B\$3*\$A9^2$ $=C\$5+C\$4*\$A10+C\$3*\$A10^2$ $=D\$5+D\$4*\$A11+D\$3*\$A11^2$ $=E\$5+E\$4*\$A10+E\$3*\$A10^2$ $=MAX(B10:E10)$						
8	x	yA	yB	yC	yD	yMax		
9	-8.0	-10.0	2.4	4.8	-96.4	4.8		
10	-7.9	-8.8	2.4	4.7	-94.8	4.7		
11	-7.8	-7.7	2.3	4.7	-93.3	4.7		
169	8.0	-330.0	5.0	1.0	-0.4	5.0		
170								

Fig. 2.16 (S) Four parabolas defined by the parameters in row vectors in rows 1 to 5; the formula reported in G7 is a better alternative for the formulas in B7:E7

Python program is developed in detail in Sect. 2.6.4, serving as the basic exercise in Python programming.

2.6.2 Data Structure and Nomenclature

x_V, y_V	arrays of the coordinates of the vertices of 4 parabolas
a, b, c	arrays of the coefficients of 4 parabolas $y = a + bx + cx^2$
x	array of x values (here, 161 from -8 to 8), separated by dx
y_A, y_B, y_C, y_D	four parabolas defined over x
G	matrix comprising y_A, y_B, y_C , and y_D as columns
y_{Max}	maximum of y_A, y_B, y_C , and y_D at the x values.

2.6.3 Spreadsheet Calculation

Figure 2.16 (S) shows a spreadsheet tabulating four parabolas for the values of x in A9:A169 and the parameters in rows 1 to 5, valid for the y -values of the parabolas in the respective column.

Question

What are the formulas for b in B4:E4 and a in B5:E5 of Fig. 2.16 (S)?¹²

The task is to write a formula into cell B9 with relative and absolute addresses so that it can be copied into the whole range B9:E169 to produce the values of the four parabolas y_A, y_B, y_C, y_D with the parameters c_-, b , and a in range B3:E5. Parameter vector c_- is specified directly by entering the desired values into B3:E3. Parameter vectors b and a are obtained from the coordinates of x_V and y_V of the vertex with the same formula as in Fig. 2.2 (S), realizing Eq. 2.3. In column F, the maximum y_{Max} of the four parabolas is built, giving their upper envelope. The result is shown in Fig. 2.15a.

The column vector x is defined in A9:A169 in 161 steps of $dx = 0.1$. The values of the four parabolas are then generated by typing the equation reported in B7 into B9 and copying right and down to E169, the range spanned by x and the row vectors a, b, c_- . The resulting curves are shown in Fig. 2.15a, together with the vertices (x_V, y_V) and the curve y_{Max} calculated in column F.

Naming cell ranges

Row and column ranges can be provided with names (see Sect. 2.3.5). Activating A1:E5 and going through FORMULAS/CREATE FROM SELECTION/CREATE NAMES

¹² $b = -c \cdot x_V$; $a = y_V + c \cdot x_V^2$, see Fig. 2.2 (S) or Table 2.1.

Table 2.18 Specification of the coefficients of 4 parabolas and composition of their labels

1	import numpy as np
2	#Specify position (xV,yV) of vertices!
3	xV=np.array([-5.0,-2.0,4.0,5.0])
4	yV=np.array([8.0, 1.0,0.5,5.0])
5	#Define coefficients of $y=a+bx+cx^3$
6	c=np.array([-2.0,0.04,0.03,-0.6])
7	b=-2*c*xV
8	a=yV+c*xV**2
9	#Compose labels for the figure
10	lbl_1=str(a[0])+" "+str(b[0])+"·x"+str(c[0])+'*x ² '
11	lbl_2=str(a[1])+" "+str(b[1])+"·x"+str(c[1])+'*x ² '
12	lbl_3=str(a[2])+" "+str(b[2])+"·x"+str(c[2])+'*x ² '
13	lbl_4=str(a[3])+" "+str(b[3])+"·x"+str(c[3])+'*x ² '
14	print(lbl_1)
15	-42.0+-20.0·x+-2.0·x ²

FROM VALUES IN THE/LEFT COLUMN does the job. Proceeding correspondingly with A8:A169 gives us a column vector named x . The formula can now be B9 = [=a + b*x + c_*x^2] instead of B9 = [B\$5 + ...], and remains the same when copied into the whole range B9:E169.

Application as a matrix formula

The formula [=a + b*x + c_*x^2] can be applied as a matrix formula within any range of suitable size. In Fig. 2.15b, the range J17:M177 has been activated, the formula entered and the process finished with the magic chord: Ψ Str + Alt + Enter. The same numbers appear as in B9:E169 of Fig. 2.16 (S).

2.6.4 Python Program

Now, how to do all this in Python?

In the first cell of Table 2.18, we import the numpy library. We run the cell so that the features and functions of numpy are available in the following cells.

Questions

What are the formulas to get a and b from the coordinates of the vertex?¹³

What are the shapes of c , b , and a in Table 2.18?¹⁴

¹³ $a = y_v + c \cdot x_v^2$ and $b = -2 \cdot c \cdot x_v$.

¹⁴ Shape(c) = (4,), c with 4 explicit entries; b and a have the same shape, because they are constructed with c .

Table 2.19 Creation of a column vector x

<pre> 1 xS=np.linspace(-8,8,161) #Is a vector = 1D array 2 xR=np.array([xS]) #Is a matrix = 2D array with one row 3 x=xR.transpose(1,0) #Creates a column vector 4 5 np.set_printoptions(edgeitems=2, 6 formatter={'float': '{: 6.1f}'.format}) 7 print('\xS ',xS) 8 print('\xR ',xR,'\n') 9 print(" x\n",x,'\n')</pre>	
<pre> xS [-8.0 -7.9 ... 7.9 8.0] xR [[-8.0 -7.9 ... 7.9 8.0]]</pre>	<pre> x [[-8.0] [-7.9] ... [7.9] [8.0]]</pre>

The next cell contains that part of the main program that reproduces the definitions and the data structure of the spreadsheet solution. The parameters x_V , y_V and a are specified exactly as before in numerical row vectors with four elements. The parameters b and c are obtained with the formulas mentioned above (in Sect. 2.2.1), also as row vectors.

In the third cell, the labels of the four curves in Fig. 2.15 are composed. Remember: the concatenation operator in Python is $+$ (contrary to $\&$ in EXCEL), and numerical values have to be converted explicitly into a string, e.g., `str(a[0])`. They are used as the legend in the chart.

In Table 2.19, the column vector x , representing the independent variable, is constructed. We could achieve that in one statement:

```
x = np.array([np.linspace(0.0,1.0,11)]).transpose(1,0)
```

but use the three lines in the first cell instead, so as to make the construction clearer. First, a simple list x_S with 161 items equally spaced between -8 and 8 , including 8 as the last element, is generated. In the output (bottom) cell, it shows up as a 1-dimensional row vector. To get a column vector, we first have to make it a matrix (2-dimensional array) x_R by defining an `np.array` with just one row showing up in the output cell as a list within double square brackets `[...]`. This matrix is transposed, with `x = xR.transpose(1,0)`, indicating that the axes 0 and 1 are interchanged, now yielding a matrix x with just one column. This is the desired column vector.

In Table 2.20, a two-dimensional array named G is constructed with $G = a + b*x + c*x**2$. It has the same shape and the same content as the range B9:E169 in Fig. 2.16 (S). The four columns of array G get additional names y_A to y_D , as in the spreadsheet. Remember: Python is zero-addressing; first index is 0 .

In column F of Fig. 2.16 (S), we have calculated the maximum of the four parabolas for each value of x . The same is achieved in Python with the statement

Table 2.20 Creating the y values of all 4 parabolas; G[:,0] indicates first column

<pre> 1 G=a+b*x+c*x**2 2 yA=G[:,0] #1st col. 3 yB=G[:,1] #2nd col. 4 yC,yD=G[:,2],G[:,3] </pre>	<pre> 5 print(G, '\n') 6 print(yA) 7 yMax=G.max(axis=1) #Max. across the columns for every x 8 print(yMax) </pre>
<pre> G [[-10.0 2.4 4.8 -96.4] [-8.8 2.4 4.7 -94.8] ... [-324.8 4.9 1.0 -0.0] [-330.0 5.0 1.0 -0.4]] yA [-10.0 -8.8 ... -324.8 -330.0] #First column yMax [4.82 4.74 ... 4.92 5.0] #For every x </pre>	

$yMax = G.max(axis = 1)$. G is a two-dimensional array, and we have to specify the axis along which the maximum value has to be found. $Axis = 1$ indicates that it is across columns for every entry in a row. The resulting curve is shown in Fig. 2.17 as a bold gray line.

Figure 2.17 is similar to Fig. 2.15 obtained with EXCEL, but now created by the Python program *FigStd* described in Sect. 2.4.5.

Fig. 2.17 Similar to Fig. 2.15, but produced with the Python program in Table 2.21

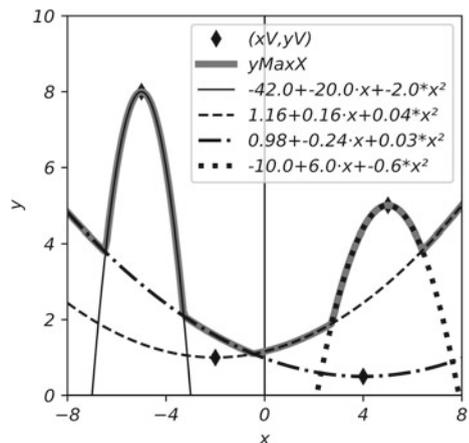


Table 2.21 Program for plotting Fig. 2.17

```

1 import matplotlib.pyplot as plt
2 FigStd('x',-8,8,4,'y',0,10,2)
3 plt.plot(xV,yV,'kd')
4         #'kd' means points as black diamonds
5 plt.plot(x,yMax,color='gray',lw=4,label='yMax')
6 plt.plot(x,yA,'k-',lw=1,label=lbl_1) #'k-' Black Line
7 plt.plot(x,yB,'k--',lw=1.5,label=lbl_2) #'--' Dashed line
8 plt.plot(x,yC,'k-.',lw=2,label=lbl_3) #'-.' Dash-dotted
9 plt.plot(x,yD,'k:',lw=3,label=lbl_4) #' :' Dotted
10 plt.legend()
11 plt.savefig('PhEx2-2 parabolas.png',dpi=1200)

```

Plotting four parabolas

The program for plotting Fig. 2.17 is given in Table 2.21, with the function `FigStd` called in the second line. It produces a figure of size 4 cm \times 4 cm with scaling and labeling of the axes as specified in the arguments. The figure is made with the entries *label*, *minimum*, *maximum*, and *distance* between the ticks, each for the *x*-axis and the *y*-axis.

We add five curves with the statement `plt.plot()`. The plot statements all get the pre-syllable `plt`, the short form under which the `matplotlib` library has been imported. The *x*-values and the *y*-values of the curve to be plotted are positional arguments and have to be the two first arguments of `plt.plot`. The third argument specifies the color and style of the curve. The keyword '*ks*' in line 2 specifies that the points (x_{Ext} , y_{Ext}) are to be marked with black ('k') squares ('s'). The next plot statements contain more keyword arguments, *lw* for linewidth, and *label* for the curve's label to be displayed in the legend. Additionally, in the last plot statement, the curve's color is explicitly specified with `color='gray'`. For more information on plot styles and options, consult Python Help!

The legend with all labels is plotted by calling the function `plt.legend()`. Generally, you can specify its position in the figure. The default is '`loc`' = 0, indicating that the program should choose an optimum position. Specifying '`loc`' = 2 would place the legend at the top-left part of the figure. Various style specifications can also be incorporated with keyword arguments, e.g., `fontsize` = 10. With the last statement, the figure is saved under the indicated name and with the indicated resolution.

Question

The statement `plt.legend` will not plot the legend. Why?¹⁵

¹⁵ The statement `plt.legend()` calls a function, and function identifiers have to be supplemented with parentheses.

Table 2.22 Extrema of matrix G along different axes

1	<code>yMax=G.max(axis=1)</code>	<code>#Across columns, for every x</code>
2		
3	<code>yMaxCol=G.max(axis=0)</code>	<code>#Down the rows, for every column</code>
4	<code>iMaxCol=G.argmax(axis=0)</code>	<code>#Index of Max in the row</code>
5	<code>xMaxCol=x[iMaxCol]</code>	<code>#Corresponding x value</code>
6		
7	<code>yMinCol=G.min(axis=0)</code>	
8	<code>xMinCol=x[G.argmin(axis=0)]</code>	<code>#Two statements in one</code>
9	<code>np.set_printoptions(precision=2)</code>	
<code>xMaxCol</code>	<code>yMax</code>	<code>[4.82 4.75 ... 4.92 5.]</code>
<code>[[-5.]]</code>		<code>#1 Max. for every column:</code>
<code>[8.]</code>	<code>yMaxCol</code>	<code>[8. 5. 4.82 5.]</code>
<code>[-8.]</code>	<code>xMaxCol[:,0]</code>	<code>[-5. 8. -8. 5.]</code>
<code>[5.]]</code>		<code>#1 Min. for every column:</code>
	<code>yMinCol</code>	<code>[-330. 1. 0.5 -96.4]</code>
	<code>xMinCol[:,0]</code>	<code>[8. -2. 4. -8.]</code>

2.6.5 Extrema Along Different Axes

In Table 2.22, we determine the extrema of the 2-dimensional matrix G , defined in Table 2.20 and containing the y values of the parabolas. The variable y_{Max} contains the maxima for axis 1, i.e., across the columns (index 1), for every entry in the rows, i.e., for the same value of x . This is the upper envelope of the four parabolas.

Next, we will find the individual extrema (minima and maxima) of the four parabolas. To do so, we have to build the maxima y_{MaxCol} and minima y_{MinCol} for every single curve. This is done down the rows (index 0), for `axis = 0`. The arrays y_{MaxCol} and y_{MinCol} contain 4 entries each. The indices i_{MaxCol} at which the maxima occur in the columns of G are found with `iMax = G.argmax(axis = 0)`; `xMaxCol = x[iMaxCol]` gives the corresponding x values. For the coordinates y_{minCol} and x_{minCol} of the minima, we proceed accordingly with `argmin`.

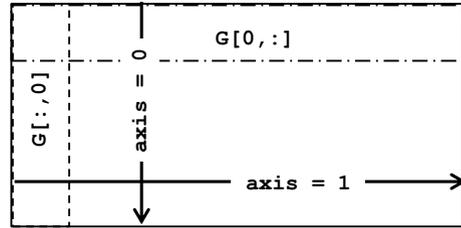
In the printout cell, we realize that x_{Max} is a column vector, i.e., a matrix with only one column. The slice `xMaxCol[:,0]` extracts the column within the matrix and displays it as a row.

Questions

Are the values for (x_{Min}, y_{Min}) and (x_{Max}, y_{Max}) consistent with the prespecified values of the vertices (x_V, y_V) ? Compare with Fig. 2.17!¹⁶

¹⁶ The four vertices show up as the 1st and 4th point in (x_{Max}, y_{Max}) and the 2nd and 3rd point in (x_{Min}, y_{Min}) . The other points are at the boundaries of the x range.

Fig. 2.18 Illustration to demonstrate indexing of matrices



The indexing of rows, columns, and axes seems not to be very intuitive. Therefore, we illustrate it in Fig. 2.18. $G[0,:]$ addresses the first row: $\text{axis} = 0$ means *across* the rows yielding a number of values corresponding to the number of columns. $G[:,0]$ addresses the first column, $\text{axis} = 1$ means *across* the columns yielding a number of values corresponding to the number of rows, in our example for every value of x .

2.7 Sum of Four Cosine Functions

We sum up four cosine functions with different angular frequencies ω_i . When the frequencies are multiples (overtones) of a fundamental frequency, such sums mimic the time signals of sound. Beats are generated when the frequencies are equally spaced within a small frequency range. The formula for the addition of cosines is illustrated by setting each two frequencies as equal, with the result being described by the broom rule Ψ “*Cos plus cos yields mean times half the difference*”.

2.7.1 Sound and a Cosine Identity

Vibrations of a string

- ▶ **Mag** Take a look at the microphone signal in Fig. 2.19a! How would you describe the signal?
- ▶ **Alac** Well, a peak is repeated periodically, and in between, there is a lot of fidgeting.
- ▶ **Mag** Yes and no. Yes, there is a fundamental frequency of repetition, and no, sound is not fidgeting; it is composed of harmonics.

Figure 2.19a is the record of the sound of a guitar string. What is the fundamental frequency in this case?

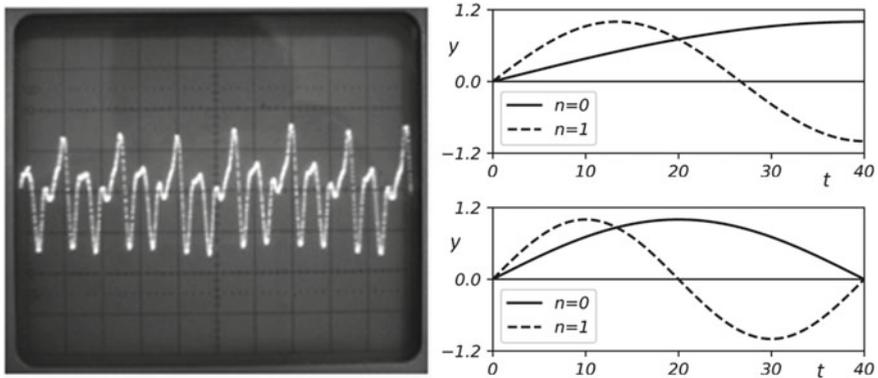


Fig. 2.19 **a** (left) Oscilloscope image of a microphone signal of a vibrating guitar string as a function of time (courtesy of Norbert Renner, University of Duisburg-Essen), time unit of the grid = 5 ms., **b** (right) Fundamental) and first harmonic oscillations (first overtones) of a reed of length l fixed on one end (top), $l = (\lambda_n/4) \cdot (2n + 1)$ and a string of length l fixed on both ends (bottom), $l = (\lambda_n/2) \cdot (n + 1)$; n indicates the number of internal nodes

- ▶ **Tim** The period is one-and-a-half grid distance, 7.5 ms, corresponding to a frequency of 133 Hz. But what are harmonics?
- ▶ **Mag** Let's consider the vibrations of a guitar string. How does a string vibrate?
- ▶ **Tim** It is sinusoidally excited, as in the bottom part of Fig. 2.19b.
- ▶ **Mag** Yes.
Figure 2.19b suggests that there are only discrete values of the wavelength with which a string can vibrate. You can determine them by considering the boundary conditions.
- ▶ **Tim** The sine must go through zero, where the string is clamped.
- ▶ **Alac** In between, it may also go through zero.
- ▶ **Mag** Exactly. The zero-crossings are called nodes. The boundary conditions require that the string length l be a multiple of half the wavelength, $l = (n + 1) \cdot \lambda_n/2$, $n = 0, 1, 2, \dots$ The possible frequencies are multiples of the fundamental frequency $c/(2l)$, where c is the velocity of sound on the string. The corresponding vibrations are called harmonics. The mode with $n = 0$ is called the fundamental mode. Now, let me repeat my question: How do you now describe the microphone signal?

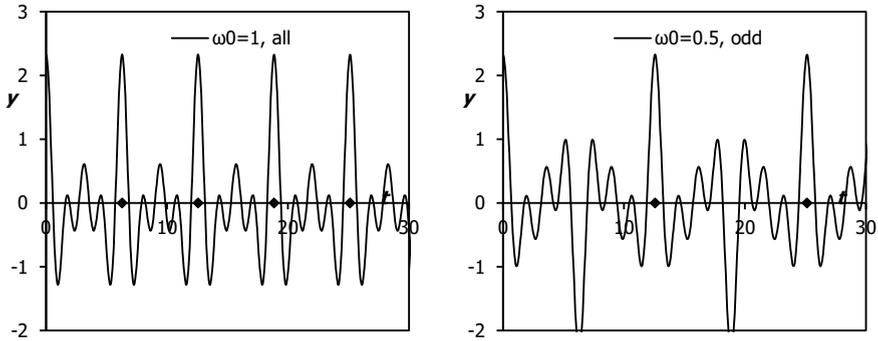


Fig. 2.20 **a** (left) Sum of a fundamental tone with $\omega_0 = 1$ and three overtones at $2x$, $3x$, $4x \omega_0$. The time unit is 1 s when the values for ω are given in 1/s. **b** (right): Sum of a fundamental tone with $\omega_0 = 0.5$ and three overtones with $3x$, $5x$, $7x \omega_0$. Time unit = 1 s

- **Alac** I guess it is the sum of the harmonics.
- **Mag** Right, it is the sum of the allowed vibrations with individual amplitudes.

In the case of a string clamped on both ends, the frequencies are multiples (1, 2, 3, ...) of the fundamental frequency, $f_n = f_0 \cdot (n + 1)$, or correspondingly for the circular frequencies ω_n . In cases in which one end of the vibrating medium is free, such as, for example, in a saxophone reed clamped on one side, the frequencies are odd multiples (1, 3, 5, ...) of the fundamental frequency, $f_n = f_0 \cdot (2n + 1)$. Examples are given in Fig. 2.20. There, you see the sum of four cosines with a fundamental frequency $\omega_0 = 1$ and multiples $2\omega_0$, $3\omega_0$, $4\omega_0$ (Fig. 2.20a) or multiples $3\omega_0$, $5\omega_0$, $7\omega_0$ (Fig. 2.20b).

Task in this exercise

- **Tim** Now, are we going to emulate the microphone signal?
- **Mag** Yes, but in the more general context of summing up four cosine functions whose frequencies satisfy certain conditions.

In this exercise, we shall use the form $y = A \cdot \cos(\omega \cdot t + \phi)$ with the parameters amplitude A , circular frequency ω , and zero phase ϕ . We set up a calculation model with four cosines, with their circular frequencies specified by a fundamental frequency and either three multiples thereof (to get harmonics) or three more frequencies at a distance of $d\omega$ (to get beats or demonstrate the addition theorem of cosines).

Beats and the uncertainty relation

Beats are shown in Fig. 2.21a. The four frequencies are again equidistant, but packed together in a small frequency range $\Delta\omega$. As control parameters for the calculation, to be systematically varied later, we take the lowest frequency ω_1 and the width of the frequency range $\Delta\omega$, i.e., the difference between the highest and the lowest

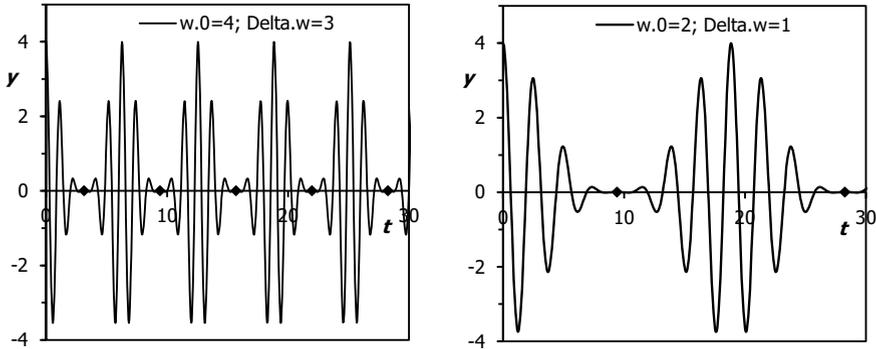


Fig. 2.21 **a** (left) Beats arising from four cosine functions; the initial angular frequency $\omega_0 = 4$, the width in the frequency range $\Delta\omega = 3$, **b** (right) another beat with initial angular frequency $\omega_0 = 2$, and width in the frequency range $\Delta\omega = 1$

frequencies. As a result, the signal clusters together into wave packets. In Fig. 2.21, they are separated by black markings. We shall determine the width of such wave packets based on an uncertainty rule.

► **Mag** The amplitudes of the cosine components of a beat are not freely chosen, but rather derived from binomial coefficients. How do you get such coefficients?

► **Alac** Well, with Pascal's triangle. Why so precise rules for the amplitudes? Can't we choose them as we like?

► **Mag** With said choice of amplitudes, we get a clear picture of the beat in the time domain, with the envelope being similar to a bell curve. Initially, you are to change only the lowest frequency ω_0 and the spectral width $\Delta\omega$ and observe the function's behavior in the time domain. Later, you may select the amplitudes at will and see whether the observed regularity is preserved.

► **Task** You are to insert points into the diagrams at the position of the nodes of the oscillations. To do so, you may create formulas that specify the marker points' coordinates when the width of a packet Δt and an initial time offset t_0 are specified. In a spreadsheet, it would be best for you to use sliders that change the coordinates so that the points in the diagram lie precisely on the nodes of the beat.

► **Mag** Have you figured out the rule for the width of a wave packet?

► **Tim** The time interval Δt for a packet is proportional to the inverse of the spectral width $\Delta t = 3\pi/\Delta\omega = 1.5/\Delta f$. The width is certainly smaller, because the signal goes to zero well before the marker points.

► **Mag** With that statement, you have found an uncertainty relation: $\Delta f \cdot \Delta t = 3/2 \geq 1$.

► **Tim** Heisenberg's uncertainty relation?

► **Mag** Yes, it is related to the commutation relation of time t and energy $E = hf$, $\Delta E \cdot \Delta t = \frac{3}{2}h \geq h$.

Sum of cosines

► **Mag** How do you write $\cos(x) + \cos(y)$ as a product of two trigonometric functions?

► **Alac** Is it necessary to know such things? You can look it up in handbooks.

► **Tim** I've memorized it: "the mean times half the difference."

► Ψ *Cos plus Cos yields the mean, times half the difference.*

► **Mag** This broom rule is a useful mnemonic if you can reconstruct the full form from it:

$$\cos(x) + \cos(y) = 2 \cdot \cos\left(\frac{x+y}{2}\right) \cos\left(\frac{x-y}{2}\right) \quad (2.8)$$

► **Alac** I'm certainly never going to forget such a crazy saying. The first cosine in the product has the average of the two primary cosines, the second half the difference.

► **Mag** Exactly, and if $x = 0$ and $y = 0$, then $1 + 1 = 2$ must be the result. This explains the pre-factor on the right-hand side of Eq. 2.8. We can test the sum formula for two cosines with our calculation model by making each two of our four frequencies equal so that only two different frequencies remain and setting all amplitudes to 0.5. The result of such a calculation can be seen as the thick gray curve in Fig. 2.22.

The black dotted curve corresponds to the function.

$$2 \cos\left(\left(w_0 + \frac{dw}{2}\right)t\right) \text{ (the mean frequency)}$$

and the dashed curve to.

$$2 \cdot \cos\left(\frac{dw}{2} \cdot t\right) \text{ (half the difference of the frequencies).}$$

We see that the cosine with half the difference frequency (dashed line) envelopes the sum. The curve oscillates with the mean frequency (dotted line). The product of the two curves experiences a phase shift π after each zero crossing of the envelope.

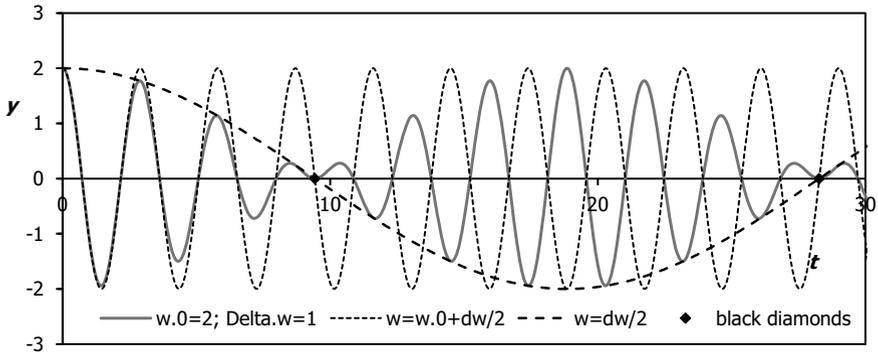


Fig. 2.22 Thick gray curve: the sum of two cosine functions with the frequencies ω_1 and $(\omega_1 + \Delta\omega)$; black dotted curve: cosine function with the mean frequency; black solid curve: envelope with half the difference of the frequencies

2.7.2 Data Structure and Nomenclature

dt	distance between adjacent points of time
t	vector of 801 equidistant points of time
c_1, c_2, c_3, c_4	cosine functions at t
c	matrix $[c_1, c_2, c_3, c_4]$
A	array of 4 amplitudes
ω	array of 4 circular frequencies
ω_0	lowest frequency
$\Delta\omega$	width of the frequency range of a beat
$d\omega$	distance between frequencies
ϕ	array of 4 phase shifts.
$sumC$	sum $c_1 + c_2 + c_3 + c_4$

2.7.3 Spreadsheet Layout

From the very beginning: a clear layout!

We set up a worksheet calculation to sum up four cosine functions, with their amplitudes A , angular frequencies ω , and zero phases ϕ to be chosen freely. The functions are to be calculated from $t = 0$ to $t = 32$ s for 801 sampling points. The basic Γ -structure of a suitable spreadsheet set-up can be seen in Fig. 2.23 (S). The names c_1, c_2, c_3, c_4 of the cosine functions are written in the spreadsheet with a dot separating the letter and the number, c.1, etc., because C1, etc., are cell addresses.

Overtones

The problem of generating overtones (higher harmonics) is a special case of the general task. The functions are to be calculated within a matrix range of, in our

	A	B	C	D	E	F	G	H	I	J
1		=w.0	=2*w.0	=3*w.0	=4*w.0		, all			
2	A	0.23	0.75	0.63	0.72					
3	w	1.00	2.00	3.00	4.00		w.0	1.00	ω_0	
4	phi	0	0	0	0		$\omega_0=1$, all	=J3&"=&w.0&G1		
5	0.04									
6	=A8+\$A\$5	=A*COS(w*t+phi)	=A*COS(w*t+phi)	=A*COS(w*t+phi)	=A*COS(w*t+phi)	=SUM(B9:E9)				
7	t	c.1	c.2	c.3	c.4	sumC				
8	0.00	0.23	0.75	0.63	0.72	2.33		Write formula into B8!		
9	0.04	0.23	0.75	0.63	0.71	2.31		Dragg into B8:E808!		
808	32.00	0.19	0.29	-0.11	-0.50	-0.13				

Fig. 2.23 (S) Four cosine functions c_1 to c_4 are calculated in the matrix range B8:E808 (below Γ), the cells of which always contain the same formula. The four functions are summed up in column F. The formulas in the columns are displayed in row 6 in oblique orientation. The time t (the independent variable) is defined as column vector A8:A808 (left of Γ) with the name t . The amplitudes A , the angular frequencies ω , and the zero phases ϕ are defined as row vectors B2:E2, B3:E3, B4:E4, respectively (above Γ). Here, the angular frequencies are multiples of the fundamental frequency ω_0

particular case, width 4 (number of functions) and height 801 (number of data points). We write the desired values for amplitudes A , circular frequencies ω , and zero phase ϕ in rows above the matrix range and the independent variable t in a column to the left of the matrix range.

You are to organize the worksheet calculation such that you need to write a formula into only one cell that is then copied to the entire calculation area (here B8:E808) for the four functions by dragging down and to the right.

Beats

The parameters for beats are specified in H2:I4 of Fig. 2.24 (S), namely, the first frequency ω_1 and the frequency range $\Delta\omega$, from which $d\omega$, the distance between neighboring frequencies, is obtained. The amplitudes A are binomial coefficients (up to a factor of 2) and can be determined using Pascal’s triangle.

Cosine identity

Figure 2.25 (S) presents a spreadsheet layout for obtaining the cosine identity shown in Fig. 2.22, the same as for beats. Here, however, $c_1 = c_3$; $c_2 = c_4$, and all amplitudes

	A	B	C	D	E	F	G	H	I	J
1		=w.0	=B3+dw	=C3+dw	=D3+dw				=Delta.w/3	
2	A	0.5	1.5	1.5	0.5		Delta.w	1.00		
3	w	2.00	2.33	2.67	3.00		dw	0.33		
4	phi	0	0	0	0		w.0	2		
5	0.04						w.0=2; Delta.w=1			

Fig. 2.24 (S) Parameters for a beat; ω (“omega”) is coded as w

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		=w.0	=B3+dW	=w.0	=D3+dW									
2	A	0.5	0.5	0.5	0.5			Delta.w	1.00		Mean value			
3	w	2.00	2.33	2.00	2.33			dw	0.33		half the difference			
4	phi	0	0	0	0			w.0	2		$w=w.0+dw/2$			
5	0.04							w.0=2; Delta.w=1			$w=dw/2$			
6	=A8+\$A\$5	=A*COS(w*t+phi)	=A*COS(w*t+phi)	=A*COS(w*t+phi)	=A*COS(w*t+phi)	=SUM(B9:E9)					=F\$8*COS((w.0+dw/2)*t)			
7	t	c.1	c.2	c.3	c.4	SumC								
8	0	0.50	0.50	0.50	0.50	2.00					2.00	2.00		
9	0.04	0.50	0.50	0.50	0.50	1.99					1.99	2.00		
808	32	0.20	0.37	0.20	0.37	1.14					1.95	1.16		

Fig. 2.25 (S) Parameter set for a sum of two cosine functions; $c_1 = c_3$; $c_2 = c_4$; H5 displays the legend for one of the curves in Fig. 2.22, while there are formulas in K6:L6 for the angular frequencies of the fast oscillation and the envelope according to the sum formula, Eq. 2.8, applied in K8:K808 and L8:L808; the amplitudes are taken from cell \$F\$8. K4 and L5 contain the legend for the other two curves in Fig. 2.22

are equal (= 0.5). The legends for the fast oscillation (with medium frequency) and the envelope (with half the difference frequency) are assembled in K1:L5.

2.7.4 Python Program

Program flow in Jupyter

The program flow in the cell structure of Jupyter is shown in Fig. 2.26. The parameters for five subtasks are specified in five program cells. The identifiers of the parameters are the same in all five cells shown in Table 2.26 and Table 2.27. The main program gets the parameter values from the cell that has been run immediately before the main program is started. To keep track of the program flow, labels that contain all relevant parameters are created in all subtasks and displayed in the main program as legends in figures.

Fig. 2.26 Program flow: the parameters for five subtasks are specified in five cells and alternatively fed into the main program

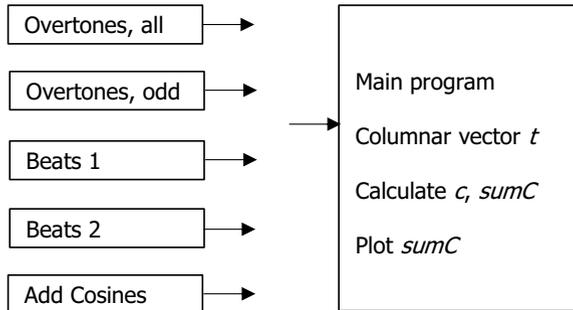


Table 2.23 Main program for calculating 4 cosine functions and plotting their sum $sumC$; y_{min} , y_{max} , and $FigName$ are specified in Tables 2.25–2.27

```

1  t0=np.linspace(0,32,801)
2  t1=np.array([t0])
3  t=t1.transpose(1,0)
4  c=A*np.cos(w*t+phi)
5  sumC=np.sum(c,axis=1)
6
7  FigStd('t',0,30,10,'y',ymin,ymax,dy)
8  plt.plot(t,sumC,'k-',label=lbl_0)
9  plt.legend(loc=0,fontsize=11)    #loc = 0, best free place
10 plt.savefig(FigName)           #Store figure as a file!
```

Main program

The main program for calculating the four cosine functions and plotting their sum is given in Table 2.23. We assume that we have imported the two libraries `numpy` and `matplotlib.pyplot` under the shortcuts `np` and `plt`, as well as the function `FigStd` described in Sect. 2.4.5. This shall be done in all future programs and shall not be reported explicitly. The coefficients A , ω , ϕ , as well as the label `lbl_0` for the curve and the name `FigName` of the file name under which the figure is saved, have to be specified ahead of time. Figures 2.20, 2.21, and 2.22 all have different scaling of the y axis, so that the corresponding axis parameters y_{min} , y_{max} , dy must also be specified in the subtask cells before calling `FigStd`.

This is done in the program cells for the different situations: harmonics (Table 2.25), beats (Table 2.26), and the addition of cosines (Table 2.27).

In order to get the same data structure as in the spreadsheet solution, we have to construct t as a column vector. We first define our discrete time points according to column A in Fig. 2.23 (S). This is done with `t0 = np.linspace(0, 32, 801)` specifying that the range from 0 to 32 is scanned with 801 equidistant points. The endpoint 32 is included by default. If we do not want that, we have to include “`endpoint = False`” as an entry behind the three positional arguments: `np.linspace(0, 32, 801, endpoint = False)`, but we won’t do that here.

The variable t_0 is now a row vector. To make it a column vector, we first transform it into a two-dimensional array by including the array t_0 within square brackets as the argument for `t1 = np.array()`, with one row only (shape = (1, 801); axis 0 has one element, a list with 801 elements. We then transpose the array between the two axes 1 and 0 by `t1.transpose(1, 0)`. The shape and the first and last elements of the three vectors are reported in the second cell of Table 2.24. The column vector t has the same shape and contains the same numbers as the variable t in A8:A808 of Fig. 2.23 (S). The same holds for the matrix c

Table 2.24 Second cell: data structure of the variables of the main program, printed with instructions similar to those in the first cell. Third cell: matrix of the y values of four cosines

<pre>1 print("t0, shape:", np.shape(t0)) 2 print(t0)</pre>	
<pre>t0, shape: (801,) [0.00 0.04 ... 31.96 32.00] t1, shape: (1, 801) [[0.00 0.04 ... 31.96 32.00]] t, shape: (801, 1) [[0.00] [0.04] ... [31.96] [32.00]]</pre>	<pre>c, shape: (801, 4) [[0.23 0.75 0.63 0.72] [0.23 0.75 0.63 0.71] ... [0.20 0.35 -0.04 -0.41] [0.19 0.29 -0.11 -0.50]] sumC, shape: (801,) [2.33 2.31 ... 0.10 - 0.13]</pre>

calculated as $c = A * \text{np.cos}(w * t + \text{phi})$ and for *sumC* reported in the third cell of Table 2.24.

Question

Which parameters relevant in the main program in Table 2.23 have to be specified in the sub-programs executed immediately before the main program?¹⁷

To calculate the four cosines, we apply the same formula, an operation on three row vectors and one column vector as in range B8:E808 in the spreadsheet (Fig. 2.23 (S)).

EXCEL: [RANGE] = [=A * COS(W * T + PHI)]

Python: $c = A * \text{np.cos}(w * t + \text{phi})$

Both operations yield a 2D array, EXCEL by dragging into a 2D range, Python by automatic broadcasting.

You should have noticed that our spreadsheet layout has been translated line-by-line into Python. The striking similarity between the two platforms when using vector notation is the reason why mathematical-physical calculations can be sensibly performed in EXCEL. Training such calculations is a good introductory exercise for computational physics.

There is a difference between the spreadsheet and the Python solution presented here:

¹⁷ (1) Labels that contain information on the characteristic frequency and frequency range. (2) Scalings of the y-axis. (3) Name under which the resulting figure is stored. (4) Parameters A, w, phi.

- In EXCEL, the four column ranges get the names *c.1*, *c.2*, *c.3*, *c.4*, with which they can be called.
- In Python, all four curves are stored in the columns of the two-dimensional list *c* and can be called by *c[:,i]* with *i* = 0, 1, 2, 3.

Summing up the four cosines is done in EXCEL with the formula in F9 = [=Sum(B9:E9)] of Fig. 2.23 (S) and in Python with `np.sum(c,axis = 1)`. The resulting variable *sumC* is a 1D vector. It is the sum across the columns (axis = 1) for a specific value of *t*, and has the rows' length. Attention: The choice of axes does not seem intuitive, but becomes apparent when *sumC* is plotted versus *t*; both have to be of the same length. For an overview, see Fig. 2.18.

Same data structure in Excel and Python

In spreadsheets, the formulas behind the values in the cells are usually hidden. To report them in our figures, we have copied their text in italic into neighboring cells. Contrastingly, in Python, the formulas are evident, but the data are hidden. To make them visible, we print them out in a well-structured manner (see Table 2.24). A comparison with Fig. 2.23 (S) shows that we have reached our goal to implement the same data structure in EXCEL and Python.

Harmonics

In Table 2.25, we specify the harmonics' parameters in the same way as in the spreadsheet, in the first cell for *all* multiples (of $\omega_0 = 1$) and in the second cell for *odd* multiples of the lowest frequency $\omega_0 = 0.5$. The values for amplitude *A*, circular frequency ω , and zero phase ϕ are specified as vectors, in Python realized as lists (characterized by square [] brackets). From these parameters, we compose a label *lbl_0*, later to be reported in the figure showing the result of our calculation. Furthermore, we specify in *FigName* the name under which the corresponding chart is to be stored. In lines 5 and 11, we specify the scale of the y-axis according to Fig. 2.20.

Table 2.25 Specifications of the parameters for harmonics

1	<code>A=[0.23,0.75,0.63,0.72]</code>	
2	<code>w=[1,2,3,4]</code>	<i>#Overtones, all</i>
3	<code>phi=[0,0,0,0]</code>	
4	<code>lbl_0="ω0="+str(w[0])+" , all"</code>	
5	<code>ymin,ymax,dy=-2,3,1</code>	<i>#Scaling the y-axis</i>
6	<code>FigName="PhExI 7-4 Overtones all"</code>	<i>#File name for later use</i>
7	<code>A=[0.23,0.75,0.63,0.72]</code>	
8	<code>w=[0.5,1.5,2.5,3.5]</code>	<i>#Overtones, odd</i>
9	<code>phi=[0,0,0,0]</code>	
10	<code>lbl_0="ω0="+str(w[0])+" (odd) "</code>	
11	<code>ymin,ymax,dy=-3,3,1</code>	
12	<code>FigName="PhExI 7-4 Overtones odd"</code>	<i>#In plt.savefig(FigName)</i>

Table 2.26 Specifications of the parameters for beats

```

1  #Beats 1
2  A=[0.5,1.5,1.5,0.5]
3  Delta_w=1
4  dw=Delta_w/3
5  w1=2
6  w=[w1,w1+dw,w1+2*dw,w1+3*dw]
7  phi=[0,0,0,0]
8  lbl_0=("ω1="+str(w1)+"; Δω="+str(Delta_w)) #Label for curve
9  ymin,ymax,dy=-4,4,2
10 FigName="PhExI 2-7-4 Beats1" #File name in savefig()
11 #Beats 2
12 A=[0.5,1.5,1.5,0.5]
13 Delta_w=3
14 dw=Delta_w/3
15 w1=4
16 w=[w1,w1+dw,w1+2*dw,w1+3*dw]
17 phi=[0,0,0,0]
18 lbl_0=("ω1="+str(w1)+ "; Δω="+str(Delta_w))
19 ymin,ymax,dy=-4,5,2
20 FigName="PhExI 2-7-4 Beats2" #File name

```

Table 2.27 Specification of the parameters for the addition of cosines

```

1  #Addition of cosines
2  A=[0.5,0.5,0.5,0.5]
3  Delta_w=1
4  dw=Delta_w/3
5  w1=2
6  w=[w1,w1+dw,w1,w1+dw]
7  phi=[0,0,0,0]
8  lbl_0=("ω1="+str(w1)+"; dω="+str(np.round(dw,2)))
9  ymin,ymax,dy=-3,3,1 #Scaling of y axis
10 FigName="PhExI 2-7-4 Sum Cos" #File name

```

Beats and the addition of cosines

The parameters for the other situations are set in Table 2.26 (beats) and Table 2.27 (addition theorem of cosines).

2.7.5 Producing Labels (as Strings) in Excel and Python

When producing labels, the following differences between EXCEL and Python have to be taken into account:

EXCEL (G4 in Fig. 2.23 (S)) A1 = [=”w0” & w.0 & “, all”]

The concatenation operator is & (ampersand), and numeric values are automatically converted into a string.

Python (Table 2.25):lbl_0 = “ω0” + str(w[0]) + “, all”

The concatenation operator is + ; numeric values have to be converted explicitly into a string. In both applications, text is enclosed in quotation marks.

When numbers x have to be rounded to n decimal places, we can use ROUND(X;N) in EXCEL and np.round(x, n) in numpy.

2.8 Questions

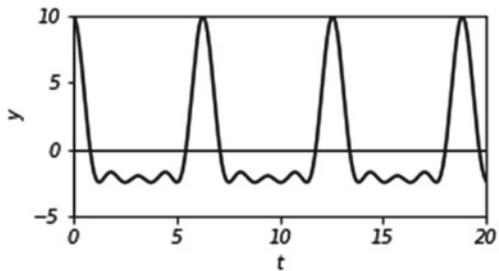
Cell references

1. What does the broom rule Ψ *The dollar makes it absolute* tell us?
2. What formula must be written in cell B5 of Fig. 2.27 (S), with absolute and relative references, so that copying this formula into the range B5:E205 creates four sine functions?
3. (Python) Specify arrays A, ω, t , so that an instruction $C = A * np.cos(\omega * t)$ generates the four cosines C_a, C_b, C_c, C_d of Fig. 2.27(S) in one matrix C . How do you replace the # in $F_{sum} = np.\#(C, axis = \#)$ to get the sum of the four cosines?

	A	B	C	D	E	F	G	H
1		1	2	3	4	Amplitude		
2		4	3	2	1	Angular frequency		
3								
4			Ca	Cb	Cc	Cd		
5		0						=A*COS(w*t)
6		1						
7		2						
205		200						

Fig. 2.27 (S) Γ structure of a spreadsheet for displaying four cosine functions

Fig. 2.28 The sum of the four cosines specified in Fig. 2.27 (S)



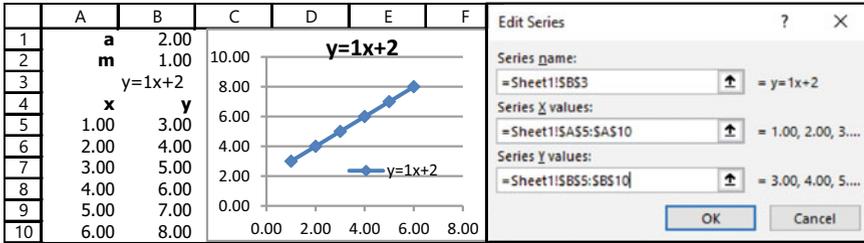


Fig. 2.29 a (left, S) A straight line is defined in the spreadsheet and displayed in the diagram. b (middle) Standard chart of the data in a. c (right) The DESIGN/SELECT DATA/EDIT dialog box, which is used to insert the data series x, y from a into the diagram

- (Python) Write a program that realizes the spreadsheet calculation in Fig. 2.27 (S), using an instruction $C=A*\text{np.cos}(w*t)$! Calculate the sum of the four cosines and plot them as shown in Fig. 2.28, with a statement `plt.plot(t,Ctot,'k-')`!

Spreadsheet function Indirect

- Look at the entries in eight cells of a table: A1 = 5; B2 = K; E1 = 5; K5 = 7; W1 = "E"&1; W2 = INDIRECT(W1); W3 = B2&A1; W4 = INDIRECT(W3). Which numbers appear in cells W2 and W4?

Arrays in Python

- Array x is specified as $x = \text{np.arange}(-8, 8 + dx, dx)$ with $dx = 1$. How many elements does x comprise and what are its first and last elements?
- What are the elements of `np.linspace(0, 3, 4)`?
- What is the shape of arrays $U = \text{np.array}([1, 2, 3])$, $UR = \text{np.array}([[1, 2, 3]])$ and $UT = UR.transpose(1, 0)$?
- What is the shape of `np.array([[1, 2], [2, 3], [3, 4]])`?
- Let $V = \text{np.array}([[1, 2, 3]])$ and $W = \text{np.array}([3, 2, 1])$. What is the shape of $U = V * W$ and $V.transpose(1, 0)$?

Diagrams

In Fig. 2.29a (S), you see data series x and y, in Fig. 2.29b, the corresponding chart, and in Fig. 2.29c, the dialog box with which the data series was inserted into this diagram.

- Which spreadsheet ranges contain SERIES NAME, SERIES X VALUES, and SERIES Y VALUES?
- How do you create the expression $y = 1x + 2$ in the spreadsheet, and how do you insert it into the chart in Fig. 2.29b as a legend?
- (Python) Below, you find a program for plotting a chart similar to the one in Fig. 2.29a, additionally with labels 'x' for the horizontal and 'y' for the vertical axis. Fill in the missing entries! The graph should be a straight line with diamonds, all black.

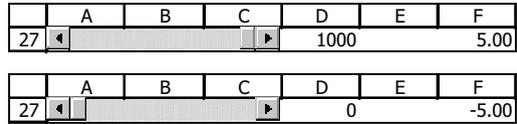


Fig. 2.30 Two settings of a slider

```
FigStd( ... )
x=[ ... ]
y=[ ... ]
plt.plot(x,y, ... )
```

14. (Python) How do you produce the string $y = lx + 2$ when $a = 2.0023$ and $m = 1.001$ are specified?
15. (Python) How do you produce a string $3.0*exp(t/-30.0)$ when $A = 3.001$ and $t_A = -30.0$ are specified?

Sliders (scroll bars)

In Fig. 2.30, you can see two settings of a slider.

16. Which is the LINKED CELL?
17. What are the minimum and maximum values of the slider?
18. The formula in F27 accesses cell D27. How does it look like?
19. The formula $= (A5-500)/100$ is used to generate decimal numbers between -5 and 5 recurring to a slider. What is the LINKED CELL, and what are MIN and MAX of this slider? What is the distance between two decimal numbers.

Polar coordinates

20. The coordinates of a circle are best given in polar coordinates with the angle ϕ and the radius r . How do you get the cartesian coordinates x and y needed for an xy diagram?

The figure in Fig. 2.31a is generated by the spreadsheet organization in Fig. 2.31b (S). Cells C11 and E11 have the names shown to their left. The column area B14:B26 gets the name *phi*.

21. How big are the numbers $dPhi$ and r_K ?

Apply names for cell ranges, if defined, in the answers to the next three questions!

22. What formulas are in cells B15 and B26?
23. Which formula is in column C below x ?

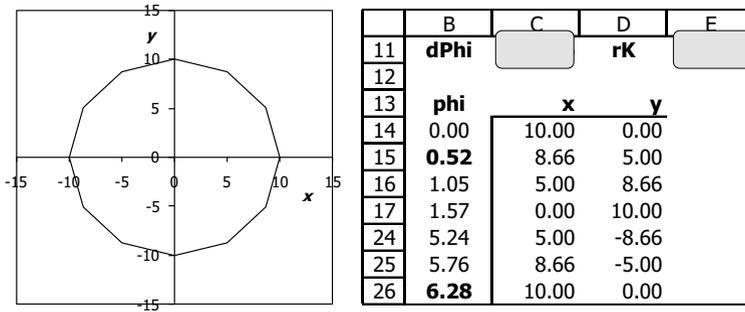


Fig. 2.31 a (left) Representation of a circle with 12 line segments; b (right, S) Coordinates for the circle in a

24. Which formula is in column D below y?
 25. (Python) Complete the following program by replacing # to get a figure similar to Fig. 2.31a:

```

rK=10
phi = np.arange(0, #, dPhi)
x=rK*#
y=rK*#
FigStd( #, #, #, #, #, #, #, #, #)
    
```

Cosine functions

Figures 2.32a and b display two cosine functions.

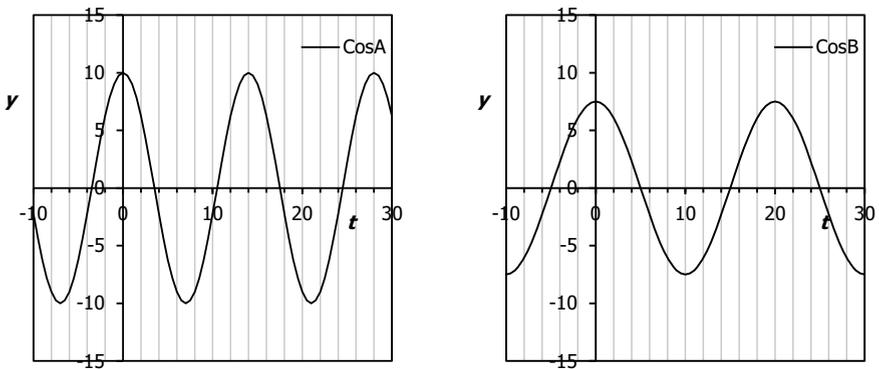


Fig. 2.32 a (left) Cosine function Cos_A . b (right) Cosine function Cos_B

-
26. What are the amplitudes and cycle times of the functions Cos_A and Cos_B shown in Fig. 2.32?
 27. What are the angular frequencies of the two functions shown in Fig. 2.32?
 28. What are the overtones to the fundamental with the frequency $f = 100$ Hz?
 29. How do you interpret the broom rule: $\Psi \text{ Cos plus Cos} = \text{mean value times half the difference}$?
 30. A second cosine function is added to a cosine function with $f = 100$ Hz. What frequency must the second cosine function have to produce a beat of 1 Hz?

Formula Networks and Linked Diagrams

3

In this chapter, we practice clearly-structured calculations with formulas. Our aim for spreadsheets—formulas in cells should be similar to mathematical formulas, a feat that is achieved by naming variables, calling them by their name, and clearly separating independent and dependent variables. As in every chapter, parallel solutions in Python are presented. For both EXCEL and Python, we aim to document intermediate results step by step from top to bottom and accompany them with charts. This way, the results are checked during the implementation of the formulas, and the calculation is easy to understand, even weeks later. With these rules in mind, we treat image construction for lenses, the Doppler effect, and exponential growth.

3.1 Introduction: Well-Structured Sheets and Programs

Solutions of Exercises 3.2 (Excel), 3.3.1 (Python), 3.3.5 (Python), and 3.4 (Excel) can be found at the internet address: go.sn.pub/McoItP.

Physical tasks with networks of formulas

Many tasks in secondary education, or in physics courses at colleges, or even in regard to minor subjects at universities, are based on simple formulas used to solve practical tasks. In this chapter, we solve such tasks through spreadsheet calculations and Python programs:

- image constructions for optical lenses,
- the Doppler effect with a general formula and for an observer off a race track, and
- exponential diode characteristics.

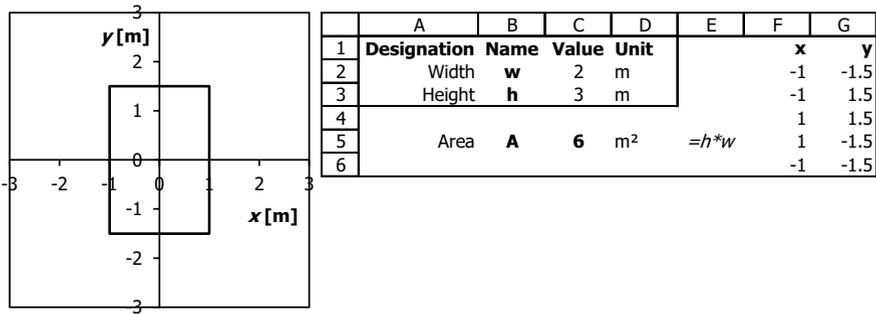


Fig. 3.1 a (left) A rectangle around the origin of the coordinate system b (right, S) Calculation of the area of a rectangle; Independent variables in B2:C3; coordinates x, y for the representation in a are calculated in columns F and G. b (right) Spreadsheet solution; coordinates for the rectangle shown in a, symmetrical to the origin of the coordinate system

For this purpose, a system of formulas has to be built using the results of other formulas. The spreadsheet set-ups and the Python programs should reflect the line of thought and be easily traceable, even weeks later.

Solve step by step and write it down in mathematical language!

The student shall learn to keep an overview by developing the solution step by step and displaying the results graphically, e.g., as optical ray construction. The diagrams should adapt automatically to any change in the parameters of the task.

Illustration: Draw a rectangle!

We consider a simple task: Specifying the width w and height h of a rectangle, calculating its area A , and drawing the rectangle as a chain of straight lines with vertices (x, y) , with its center at the origin of the coordinate system (see Fig. 3.1a).

Python program

In Python, every object has to have an identifier (a name) so that, from the outset, the implementation is similar to mathematical formulas. For Fig. 3.1a, we apply Table 3.1.

Formula network in spreadsheets

In spreadsheet calculations, we have to pay special attention in order to achieve our goal. It is necessary to call all variables by names such that the cell formulas can be written like mathematical formulas. It is often useful to present the calculation in four columns: designation of the quantity, variable name as an identifier for cells, numerical value, and physical unit. An example is given in Fig. 3.1b. All independent variables are in a block at the top left of the sheet, with names in B2:B5 and units in D2:D5.

If one of the independent variables is changed, the entire calculation and all diagrams should follow, without corrections having to be made somewhere in the

Table 3.1 Specifying the coordinates of a rectangle around (0, 0)

1	w=2	# [m]	Width
2	h=3	# [m]	Height
3	A=h*w	# [m ²]	Area
4	x=[-w/2, -w/2, +w/2, +w/2, +w/2]		
5	y=[-h/2, +h/2, +h/2, -h/2, -h/2]		

sheet. We call a spreadsheet structure corresponding to these requirements a ‘formula network’. It is often useful to change independent variables with sliders that can quickly be used to get an impression of the trend of the solution (not realized in Fig. 3.1b).

The area is calculated as $F = h * w = \text{height by width}$. If you write “= h * b” into cell C5, C5 = [= h * w], then immediately “6” appears after you hit ENTER. In columns F and G of Fig. 3.1b, the coordinates for the graphical representation in Fig. 3.1b are calculated. The formulas are the same as for x and y in Table 3.1. Cells and cell areas must be named, e.g., F2:F6 with “x”. A good formula network must not only be correct, but also clear!

Provide cells in spreadsheets with names

There are several ways to name cell areas. We already became acquainted with this in the last chapter in Exercise 2.1 and Sect. 2.3.5. You can also find out more about this in the EXCEL help under the keyword “Create a name”. We prefer to use the variant FORMULAS/DEFINED NAMES/CREATE FROM SELECTION, which has the advantage that the names given in the NAME MANAGER are visible in the spreadsheet, and thus contribute to the clarity of the calculation.

Check the solution with diagrams

For the representation of the rectangle in a diagram, here, Fig. 3.1a, the coordinates x , y are calculated from the values for width w and height h . The drawings should adapt automatically to any change in the parameters of the task. Therefore, in the spreadsheet of Fig. 3.1b, columns F and G contain formulas and not just numbers. The entries for w and h control all computations in the sheet and the figure, giving the impression of a “living spreadsheet”.

Experience has shown that it is especially appealing to many students to work on the tasks until the diagram “obeys on command”. In the case of more complex tasks, it is also easier to see whether the formula is correct.

Question

What are the formulas for x and y in Fig. 3.1b (S)?¹

¹ $x = +w/2$ or $x = -w/2$; $y = +h/2$ or $y = -h/2$, always correctly combined.

Mathematical functions

In this chapter, we apply four mathematical functions:

- straight lines for image constructions of geometrical optics,
- exponentials for diode characteristics,
- polar coordinates with sine and cosine.

3.2 Image Construction for Focusing and Diverging Lenses

We construct the image point that a lens generates from an object point with three characteristic rays, applying the general imaging equation valid for both a focusing lens and a diverging lens. Ψ *Lens equation with plus and minus.* We draw the bundle of rays through the lens, which actually contributes to the image point. After this exercise, the reader should be able to master the straight-line equation blindfolded.

3.2.1 Straight Line Equation

In this exercise, a straight line is defined by two points (x_1, y_1) and (x_2, y_2) . For a given third coordinate x_3 , a coordinate y_3 is to be calculated so that the point (x_3, y_3) lies on the straight line.

Straight line equation

Given two points (x_1, y_1) and (x_2, y_2) of a straight line, the straight-line equation is

$$y(x) = y_1 + m \cdot (x - x_1) \quad (3.1)$$

or

$$y(x) = y_2 + m \cdot (x - x_2) \quad (3.2)$$

both times with the slope

$$m = \frac{y_2 - y_1}{x_2 - x_1} \quad (3.3)$$

We call the respective points (x_1, y_1) in the first straight-line equation and (x_2, y_2) in the second the *reference points* of the straight line.

	A	B	C	D	E	F
1	x.1	-1.91	209			
2	y.1	3.00				y.1; y.2
3						
4	x.2	2.00				
5	y.2	-2	200			
6						
7	m	$-1.28 = (y.2 - y.1) / (x.2 - x.1)$				
8	x.3	1.63	563			
9	y.3	$-1.53 = y.2 + m * (x.3 - x.2)$				
10						
11	Straight line					
12		-10	$13.35 = y.2 + m * (B12 - x.2)$			
13		10	$-12.23 = y.3 + m * (B13 - x.3)$			

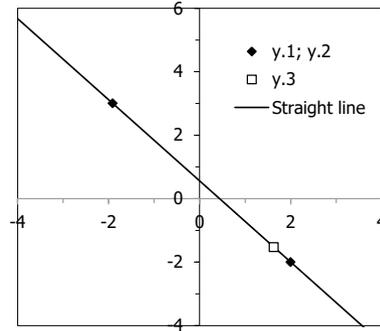


Fig. 3.2 a (left, S) Spreadsheet layout for the diagram in **b**; the values of x_1 and y_1 are obtained by means of sliders. **b** (right) The point “y.3” is to lie on a straight line given by the two points “y1; y2”

We demonstrate Eq. 3.1 through the spreadsheet in Fig. 3.2a. The y -value y_1 of the first and the x -value x_2 of the second defining point are directly written into cells B2 and B4, respectively. The associated values x_1 and y_2 in B1 and B5 are determined using the two sliders in D1:F1 and D5:F5. From these coordinates, the slope m is calculated in B7 with Eq. 3.3. The x value x_3 of the third point in B8 is selected with the slider in D8:F8, and the corresponding y -value y_3 in B9 is obtained with the straight-line equation Eq. 3.2. These three points are represented in Fig. 3.2b with diamonds.

In range B12:C13 of Fig. 3.2a (S), the straight-line coordinates are calculated for x values -10 to 10 extending beyond the range of the x -axis in Fig. 3.2b so that the straight line goes through the whole picture. The straight line is entered into the figure with SERIES X- VALUES: (B12:B13), SERIES Y- VALUES: (C12:C13).

Questions

Questions concerning Fig. 3.2a (S):

What are the linked cells for the three sliders?²

Which number range (the same for all) is presumably covered by the sliders?³

If the coordinates, to be set by sliders with MIN = 0 and MAX = 800, take on values between -4 and 4 , what are the formulas in cells B1, B5, and B8, with which the coordinates are calculated from the cells linked to the sliders?⁴

² The linked cells are C1, C5, and C8.

³ The number range of all sliders runs from 0 to 800.

⁴ B1 = $[(C1 - 400)/100]$; B5 = $[(C5 - 400)/100]$; B8 = $[(C8 - 400)/100]$.

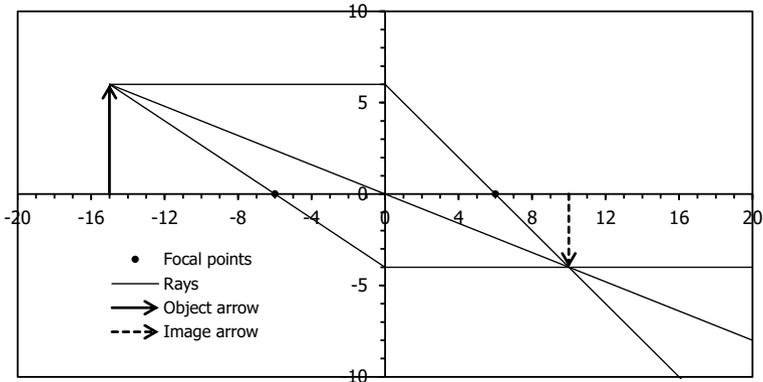


Fig. 3.3 Image construction for a focusing lens with parallel, central and focus rays

In C12 and C13, the y coordinates of the straight line are calculated with two different formulas. Why do both formulas describe the same straight line?⁵

If we have done everything correctly, the graphic presentation in Fig. 3.2a should adapt to every change of the values for y_1 , x_2 , x_3 by the three sliders, and the three points should lie on a straight line every time.

3.2.2 Geometrical Image Construction for a Thin Focusing Lens

Image construction by ray drawing

Figure 3.3 illustrates how the image of an object point is constructed with three characteristic rays in the xy -plane.

The x -axis represents the optical axis and also the axis of the circularly shaped lens. A thin lens is represented by its principal plane and its focal length. The shape of the lens does not play a role anymore. The principal plane is the plane $x = 0$ in which the y -axis is situated.

Question

How does the parallel ray in an image construction run?⁶

⁵ The slopes are the same for both straight lines. As reference points, (x_2, y_2) has been chosen for C12 and (x_3, y_3) for C13. As both lie on the straight line, the third point also lies on the same line.

⁶ The parallel ray runs from the object point parallel to the x -axis (that is, the optical axis) up to $x = 0$ and then through the image-side focus.

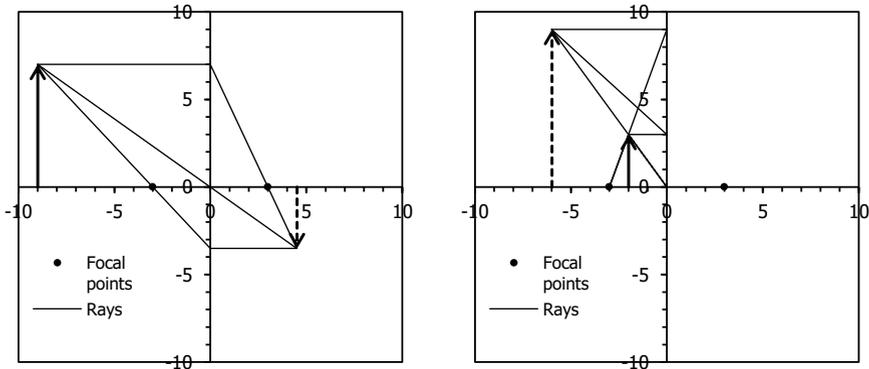


Fig. 3.4 **a** Converging lens, **a** (left) real inverted image for an object distance outside double the focal length. **b** (right) Virtual, upright image for an object distance within the focal length; corresponds to a look through a magnifying glass from the right

► **Mag** Do you remember how to determine the image point of an object point geometrically?

► **Alac** Yes, as in Fig. 3.4a. We draw two rays starting from the object point, one through the center of the lens and another parallel to the optical axis up to the principal plane and then through the focal point on the right side of the lens. The image point is where the two rays intersect.

► **Mag** So you can do it. Your construction is valid for a converging lens. However, a lens has two focal points, one on the image-side and another one on the object-side. In the image construction just described, you have exploited the fact that all rays incident parallel to the optical axis go through the *image-side focus* after having passed through the lens.

► **Tim** We have often drawn a third ray from the object point through the focal point on the left of the lens, which imagine is called the *object-side focal point*. After passing through the lens, this ray is parallel to the optical axis and then passes through the image point.

► **Mag** Yes, all three construction rays intersect at the image point, as in Fig. 3.4. That's what we want to reproduce with our exercise.

In Fig. 3.4, the image construction for a converging lens using the principal rays (parallel, central, and focal) is represented in a Cartesian coordinate system. By convention, the optical axis is the x -axis. The optical center and the lens's principal plane are respectively located in the origin of the coordinate system in the plane $x = 0$. We take it as given that the object is always to the left of the lens, i.e., *the object distance is always negative*.

We know (from physics courses) that an inverted real image is formed to the right of the lens when the object distance is bigger in magnitude than the focal length (Fig. 3.4a). If the object distance is smaller in magnitude than the focal length, the result is an erect virtual image to the left of the lens (Fig. 3.4b). The general imaging equation for optical lenses considers these relationships by sign conventions for the variables that enter the imaging equation, Eq. 3.4.

► *Ψ Lens equation with plus and minus!*

3.2.3 Imaging Equation with Correct Signs

► **Mag** Do you know the imaging equation for lenses?

► **Alac** Sure, I've already learned it at school:

$$\frac{1}{x_O} + \frac{1}{x_I} = \frac{1}{f} \quad (3.4)$$

where x_O and x_I are the object and the image distance, respectively, and f is the focal length.

► **Mag** This equation is useful only for handmade geometric constructions. For an analytical calculation, we must use a more accurate one, namely, Eq. 3.5, in which two modifications with respect to Eq. 3.4 have been introduced. Now, f_I is the image-side focal length, x_I the image distance, and x_O the object distance. The object distance is, in principle, negative, because the object is, by convention, placed to the left of the lens.

► **Alac** With the old equation, we always got the correct values for image distance and image size.

► **Mag** Yes, the absolute values are calculated correctly. However, no signs, plus or minus, are considered. Let's adopt the more general notation. The object distance x_o is negative if the object is to the left of the lens. The image is often upside down. This is automatically considered in Eq. 3.6, which calculates the image size y_I from the object size y_O .

Furthermore, Eqs. 3.5 and 3.6 are also valid for a diverging lens if a negative image-side focal length is introduced, $f_I < 0$.

General imaging equation for lenses

For the analytical computation of images of lenses, the imaging equation has to be written with signs:

$$-\frac{1}{x_O} + \frac{1}{x_I} = \frac{1}{f_I} \quad (3.5)$$

The x -axis is the optical axis. The principal plane of the lens is in the plane $x = 0$; the object distance x_O is negative. The image-side focal length f_I is *positive* for converging lenses and *negative* for diverging lenses. The image distance x_I may result positive or negative. The imaging scale is

$$\frac{y_I}{y_O} = \frac{x_I}{x_O} \quad (3.6)$$

with y_O being the object size and y_I the image size that can be positive or negative.

Converging lens

In Fig. 3.4, you see the usual image construction for a focusing lens ($f_I > 0$) employing parallel, center, and focus rays.

Diverging lens

For the image construction of a diverging lens, you can use the same spreadsheet calculation or Python program as for a converging lens. You only have to enter a negative image-side focal length f_I . Figure 3.5 shows two examples (obtained from Table 3.3).

3.2.4 Beam Through a Converging Lens that Really Contributes to the Image

- ▶ **Mag** In your geometric ray constructions, you did not draw the cross-section of the lens. What can you say about the lens?
- ▶ **Alac** In all cases, the center is thicker than the edge; otherwise, it would not be a converging lens.
- ▶ **Tim** The principal plane of the lens is located in the plane $x = 0$. The center of the lens is at the origin of the coordinate system.
- ▶ **Mag** How big should the diameter of the lens be, e.g., in Fig. 3.4?

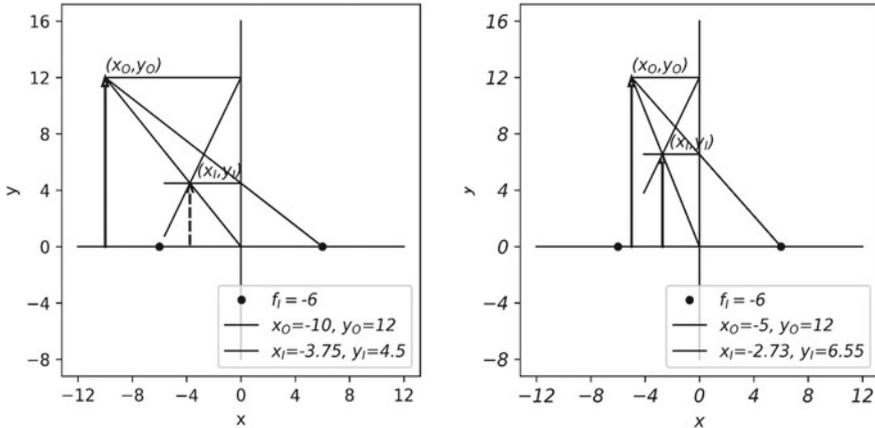


Fig. 3.5 Imaging with a diverging lens, the geometric construction being the same as for a converging lens, but with negative focal length ($f_l < 0$) (drawings obtained with the Python program in Table 3.4). **a** (left) upright image for an object distance larger than the focal length. **b** (right) As with **a**, but for an object distance smaller than the focal length

Table 3.2 Specifications for an image construction with a converging lens, resulting in a figure similar to Fig. 3.3

```

6   #Converging lens
7   fI=6.0
8   xO=-15
9   yO=6
10  xMin, xMax, Dx = -20, 20, 5 #Scaling of figure axes
11  yMin, yMax, Dy = -10, 10, 2.5
12  FigName='Converging lens' #File name in plt.savefig()

```

Table 3.3 Specifications for image constructions with a diverging lens, resulting in Fig. 3.5

```

1   #Diverging lens, Object beyond focal length
2   fI=-6
3   xO=-10
4   yO=12
5   xMin, xMax, Dx = -12, 12, 4
6   yMin, yMax, Dy = -8, 16, 4
7   FigName='Diverging lens, outside' #In plt.savefig()
8   #Diverging lens, Object within focal length
9   fI=-6
10  xO=-5
11  yO=12
12  xMin, xMax, Dx = -12, 12, 4
13  yMin, yMax, Dy = -8, 16, 4
14  FigName='Diverging lens, inside' #In plt.savefig()

```

► **Alac** I would draw the lens from $y = -7$ to $y = 7$ so that the three constructing rays pass through the lens. The diameter would be roughly the same as those of the lenses used for lecture experiments.

► **Mag** Be cautious; think of cameras! In that case, the lens diameter is much smaller than, for example, the elephant you are photographing.

► **Tim** That's right. But does it mean that the construction rays do not go through the lens?

► **Mag** They don't, indeed. They exist only in thought and on paper. Which rays actually do contribute to the image point for a camera lens?

► **Alac** Only the central ray, or perhaps other rays that really do pass through the lens.

► **Mag** Yes, the image point is formed by a bundle of rays through the lens, as we will draw now. The lens itself has not shown up in the figures presented so far. The size of the lens is irrelevant to the image construction; only the principal plane and the focal length are needed to construct the image point.

Figure 3.6 shows the ray construction of the image point, together with the cross-section of the lens and eleven rays going from the object point through positions in the lens's full span and finally focusing in the image point. The parallel ray and the focus ray run outside the lens. They do not exist in physical reality.

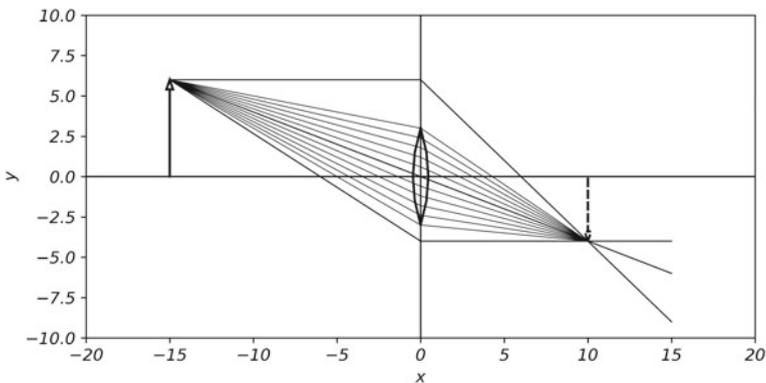


Fig. 3.6 (By program in Tables 3.4 and 3.5) Light beam contributing to the image formation in Fig. 3.3

Cross-section of the lens

We are going to add the cross-section of the lens to the drawings. To do so, we need to construct segments of a circle. We make use of the circle equation:

$$x^2 + y^2 = r^2 \quad (3.7)$$

The parameters are the coordinates x_0 of the center point on the x -axis, the radius r_K of the circle determining the lens's curvature, and the lens diameter $D_L = 2r_L$ (in front view). With $x_0 = \sqrt{r_{Lens}^2 - r_L^2}$, we calculate the distance of the center of the sphere, limiting the surface of the lens to the origin of the drawing. With

$$x = \sqrt{r_{Lens}^2 - y^2} - x_0, \quad (3.8)$$

we get the x -coordinate for a given y -coordinate on the surface of the lens.

3.2.5 Data Structure and Nomenclature

f_I	image-side focal length
(x_O, y_O)	coordinates of the object point
(x_I, y_I)	coordinates of the image point, to be calculated with the imaging equation.

Three rays for the geometrical construction of the image, without prior knowledge of the coordinates of the image point, are defined by f_I and the object point from which characteristic slopes have to be calculated.

x_{Par}, y_{Par}	3 characteristic points of the parallel ray
x_{Cen}, y_{Cen}	3 characteristic points of the central ray
x_{Foc}, y_{Foc}	3 characteristic points of the object-side focal ray
$m_{Par}, m_{Cen}, m_{Foc}$	slopes of the non-horizontal parts of the three rays
x_{Lens}, y_{Lens}	cross-section of a converging lens, calculated with Eq. 3.8.

3.2.6 Spreadsheet Calculation

Imaging equations

We calculate the ray path coordinates for imaging with a converging lens, and therewith set up an image construction that should adapt automatically whenever the parameters are changed. The coordinates of the three constructing rays' defining points are shown in Fig. 3.7 (S).

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Specifications										Rays		
2	Image-side focal length	fI	6.0								<i>Central ray</i>		
3	Object distance	xO	-15.0				slope			-15 =xO	6 =yO		
4	Object height	yO	6.0							0 =0	0 =0		
5	Imaging equation						mCen	-0.4		20		-8 =mCen*J5	
6	Image distance	xI	10.0	$= (1/fI + 1/xO)^{-1}$				$=yO/xO$			<i>Parallel ray</i>		
7	Image height	yI	-4.00	$=yO*xI/xO$							-15 =xO	6 =yO	
8											0 =0	6 =yO	
9	Focal points		6 =fI	0			mPar	-1		20		-14 =yO+mPar*J9	
10			-6 =-fI	0				$=-yO/fI$					<i>Focal ray</i>
11	Object arrow		-15 =xO	0							-15 =xO	6 =yO	
12			-15 =xO	6 =yO							-6 =-fI	0 =0	
13	Image arrow		10 =xI	0 =0			mFoc	-0.67		0		-4 =yO+mFoc*-xO	
14			10 =xI	-4 =yI				$=yO/(xO+fI)$		20		-4 =L13	

Fig. 3.7 (S) Imaging equation for a converging lens; the quantities related to the object and the image are designated with the indices O and I. $x = 20$ in column J indicates the right border of Fig. 3.3

The five parameters (focal length, object distance, and height, as well as the image distance and height that are dependent on them) are provided with the names in column B, with which they are entered into the image equations (B6:C7) and the coordinates of the construction rays (J:M).

The coordinates of the focal, central, and parallel rays are introduced as data series into the diagrams of Fig. 3.4a, b. We may enter the column range J3:J14 as SERIES X VALUES and L3:L14 as SERIES Y VALUES to get three separate straight lines, because empty rows separate their coordinates. The designations *Central ray*, etc., are in column K, not in column J or column L.

► Ψ Empty rows separate curves.

Note that, in most cells, there are formulas. So, you cannot simply copy the numbers from the spreadsheets displayed in this text. If you have done the implementation correctly, images such as those in Fig. 3.4a, b should result, automatically adapting whenever you change the parameters of focal length, object distance, and object height.

► **Alac** A fascinating experience!

The parameters in Fig. 3.7 (S) are for a converging lens ($f_I = 6.0$) and an object distance ($x_O = -15$) beyond the focal length (Fig. 3.3). With $f_I = 3.0$, $x_O = -9$, and $y_O = 7$, Fig. 3.4a results. With $f_I = 3.0$, $x_O = -2$ (within the focal length), and $y_O = 3$, Fig. 3.4b results.

For a diverging lens (Fig. 3.5), we choose $f_I = -6$ and $x_O = -10$ (a) or $x_O = -5$ (b).

1 Sub Bundle()	For i = 0 To nR - 1	11
2 xO = Range("C3")	Cells(r2, c2) = xO	12
3 yO = Range("C4")	Cells(r2, c2 + 1) = yO: r2 = r2 + 1	13
4 xI = Range("C6")	Cells(r2, c2) = 0	14
5 yI = Range("C7")	Cells(r2, c2 + 1) = -RL + i * Dy: r2 = r2 + 1	15
6 nR = 11 'Number of rays	Cells(r2, c2) = xI	16
7 RL = 3 'Lens radius	Cells(r2, c2 + 1) = yI: r2 = r2 + 1	17
8 Dy = 2 * RL / (nR - 1)	r2 = r2 + 1	18
9 r2 = 3 'First row for output	Next i	19
10 c2 = 10 'Column J	End Sub	20

Fig. 3.8 (P) Drawing rays from the object point through the lens to the image point; the coordinates of the object point and the image point are read from column C of a spreadsheet, e.g., Fig. 3.7 (S). Dy is the distance between rays at $x = 0$

Ray bundle through the lens

We use a VBA subroutine⁷ as in Fig. 3.8 (P) to draw the ray bundle that physically contributes to the image. The rays run from the object point (x_O, y_O) to a point in the lens, and finally to the image point (x_I, y_I). The coordinates of the points are in C3:C7 of a spreadsheet; they are read in the first lines of the sub-routine. The parameters of the lens are specified within the subroutine. The defining points of the rays are written into columns 6 (c_2, F) and 7 ($c_2 + 1, G$) of the spreadsheet.

3.2.7 Python Program

Specifications for three different types of images

The Python program is organized into four cells. The first three cells contain the specifications and a filename to store the resulting figure, each for

- a converging lens (Table 3.2, resulting in a figure similar to Fig. 3.3),
- a diverging lens with the object outside the focal length (Table 3.3 top, resulting in Fig. 3.5a),
- a diverging lens with the object inside the focal length (Table 3.3 bottom, resulting in Fig. 3.5b).

The fourth cell, represented in Tables 3.4 and 3.5, draws arrows representing the image and the object, together with the image construction, with rays using the specifications of one of the three initial cells that were run earlier.

The specifications in the three cells comprise not only focal length and the coordinates of the object point, but also parameters for axis scaling of the image and the name of the file wherein the image is to be stored. When the program in one of these cells is run, the resulting parameters are valid for the following image construction in Fig. 3.4. So, each of the three situations can be the basis of an image.

⁷ VBA macros are introduced in Chap. 4.

Table 3.4 Drawing object and image arrows, function *ArrowP* presented at the end of this section

```

1 FigStd('x',xMin,xMax,Dx,'y',yMin,yMax,Dy,xlength=8)
2 plt.plot((-fI,fI),(0,0),'ko',
           markersize=4,label="$f_I=$"+str(fI))
3 ArrowP((xO,0),(xO,yO),lw=1.5)           #Object
4 lbl_1=r'$x_O$='+str(xO)+r', $y_O$='+str(yO)
           #'$x_O$' becomes x_o=
5 plt.text(xO,yO+0.5,r"($x_O$, $y_O$)",fontsize=10)
6 #Calculated image
7 xI=1.0/(1.0/fI+1.0/xO)
8 yI=yO*xI/xO
9 Arrow((xI,0),(xI,yI),lw=1.5,ls='--') #Image
10 lbl_2=(r'$x_I$='+str(round(xI,2))
         +r', $y_I$='+str(round(yI,2)))    # $x_I$ as x_I in legend
11
12 plt.text(xI+0.5,yI+0.5,"($x_I$, $y_I$)",fontsize=10)

```

Table 3.5 Continuation of Table 3.4; setting up the image construction with the parameters specified in other cells without explicitly referring to the image point

```

13 #Parallel ray
14 xPar=[xO,0,1.5*xI]
15 mPar=-yO/fI                               #Slope in image space
16 yPar=[yO,yO,yO+mPar*xPar[2]]
17 plt.plot(xPar,yPar,ls='-',color='k',
           lw=1,label=lbl_1)
18
19 #Central ray
20 xCen=[xO,0,1.5*xI]
21 mCen=yO/xO                               #Slope in whole space
22 yCen=[yO,0,mCen*xCen[2]]
23 plt.plot(xCen,yCen,ls='-',
           color='k',lw=1,label=lbl_2)
24
25 #Ray through object-side focus
26 xFoc=[xO,-fI,0,1.5*xI]
27 mFoc=-yO/(xO+fI)                         #Slope in object space
28 yG0=yO+mFoc*xO
29 yFoc=[yO,0,yG0,yG0]
30 plt.plot(xFoc,yFoc,ls='-', color='k',lw=1.)
31 plt.legend(loc=4,fontsize=10)             #loc= 4 , "Lower right"
32 plt.axis('scaled')
33 plt.savefig(FigName)

```

General program for drawing the image construction

The main program is in Table 3.4; it performs the image construction according to the specifications in the cell executed earlier. Arrows representing the object and the image are drawn. The coordinates of the object follow directly from the specifications

Table 3.6 User-defined function for drawing an arrow from point P_0 to point P_1 in the xy -plane (construct explained in Chap. 4)

```

1 def Arrow(P0,P1,c="k",ls='-',lw=1,hw=0.4):
2     (x0,y0)=P0
3     (x1,y1)=P1
4     #c has to be given as c="k", not c='k'
5     plt.arrow(x0,y0,x1-x0,y1-y0,
6               length_includes_head=True,
7               head_width=hw,fill=False,
8               linestyle=ls, color=c,linewidth=lw)

```

in Table 3.2 or Table 3.3, whereas those of the image have to be calculated with the image equation (lines 7 and 8).

The program reproduced in Tables 3.4 and 3.5 calculates with the values obtained in one of the three cells in Tables 3.2 and 3.3. The ray constructions in Fig. 3.1 are obtained with the specifications in Table 3.5. The arrows are drawn with a function reproduced in Table 3.6 at the end of this section.

The three characteristic rays are drawn with the Python program in Table 3.5.

Plotting an arrow in Python

A user-defined function for drawing an arrow from point P_0 to point P_1 is shown in Table 3.6.

The arrow is plotted from P_0 to P_1 , with color c , linewidth lw , and headwidth hw entered as keyword arguments. If the function call does not specify the values of the keywords, the default values specified in the header are taken. The coordinates of the two points have to be translated into the expected entries of the function `plt.arrow` of the `Matplotlib` library.

Cross-section of the lens

The program for drawing the cross-section of a lens is given in Table 3.7.

The arrays y and x specify only one-quarter of the cross-section (see the values in the bottom cell). They run from (2.0, 0.0) up to (0.0, 6.0). For a complete cross-section, we need three more curves. They are obtained with the help of the functions

`np.flipud` (“flip up down”) reversing the order of the elements in an array, and

`np.hstack` concatenating arrays to one long array, x_{Lens} resp. y_{Lens} .

The concatenated arrays x_{Lens} and y_{Lens} are used for the drawing performed in Table 3.8, line 21. The curve begins at (0.0, 6.0), runs to (2.0,0.0), continues to (0.0, -6.0), then to (-2.0, 0.0), and closes the cross-section by running to (0.0, 6.0).

Table 3.7 Coordinates of the cross-section of a lens with radius r_L of the disk and radius r_O of curvature of the surface of the lens

1	<code>rL=6</code>
2	<code>rO = 10</code>
3	<code>xO=np.sqrt(rO**2-rL**2)</code>
4	<code>y=np.linspace(0, rL, 3)</code>
5	<code>x=np.sqrt(rO**2-y**2)-xO</code>
6	<code>xf=np.flipud(x)</code> <i>#First becomes last</i>
7	<code>yf=np.flipud(y)</code>
8	<code>FigStd('x', -20, 20, 5, 'y', -10, 10, 2.5, xlength=8)</code>
9	<code>xLens=np.hstack([xf, x, -xf, -x])</code> <i>#One long array</i>
10	<code>yLens=np.hstack([yf, -y, -yf, y])</code>
x	<code>[2.00 1.54 0.00]</code>
y	<code>[0.00 3.00 6.00]</code>

Bundle of rays through the lens

The coordinates of the $n_R = 11$ rays going through the lens are calculated in Table 3.8 with pre-specified coordinates of the object and the image point, e.g., in Table 3.4. The rays are drawn in a for-loop, starting with a y -coordinate at the bottom of the lens and increasing it by $\Delta y = 2 \cdot r_L / (n_R - 1)$. The cross-section of the lens is drawn with the plot-statement in line 21, using the coordinates x_{Lens} , y_{Lens} calculated in Table 3.7. The complete drawing, i.e., image construction, lens, and ray bundle, is shown in Fig. 3.6.

Table 3.8 Drawing a bundle of rays from the object point through the lens to the image point

11	<code>#Bundle of rays through the lens</code>
12	<code>#Object point and image point are known.</code>
13	<code>nR=11</code> <i>#Number of rays</i>
14	<code>rL=3</code> <i>#Diameter of lens</i>
15	<code>x=np.zeros(3)</code>
16	<code>y=np.zeros(3)</code>
17	<code>(x[0], y[0])=(xO, yO)</code> <i>#Object point</i>
18	<code>(x[2], y[2])=(xI, yI)</code> <i>#Image point</i>
19	<code>x[1]=0</code>
20	<code>FigStd('x', -20, 20, 5, 'y', -10, 10, 2.5, xlength=8)</code>
21	<code>plt.plot(xLens, yLens, 'k')</code>
22	<code>Arrow((xO, 0), (xO, yO), lw=1.5)</code> <i>#Object</i>
23	<code>Arrow((xI, 0), (xI, yI), lw=1.5, ls='--')</code> <i>#Image</i>
24	<code>Dy=2*rL/(nR-1)</code>
25	<code>for i in range(nR):</code> <i>#Bundle of rays</i>
26	<code> y[1]=-rL+i*Dy</code> <i>#Position in lens</i>
27	<code> plt.plot(x, y, 'k-', lw=0.5)</code>

3.3 Doppler Effect

When a sound source (a “sender”) and a receiver move relative to air, the receiver perceives a frequency that is different from the transmitted one. We set up a formula for all cases of movements of the two agents on a straight line. We determine the frequency trajectory recorded at a receiver off the sender’s track.

3.3.1 A Formula for All Cases

When a sound source (in the following, designated as sender S) and a receiver R are approaching or moving away from one another on a straight line, the receiver perceives a frequency different from that emitted. In the following, we develop a formula for the cases when sender and receiver move on the same straight line.

Formula for intuitive use (Doppler)

The relationship between frequencies f (frequency ratio) and speed v is given by the following formula:

$$\frac{f_R}{f_S} = \frac{c \pm v_R}{c \mp v_S} \quad (3.9)$$

The letters f , c , and v denote the frequency, the speed of sound, and the speed (≥ 0) of the agents relative to air. The upper sign in the formula is valid when the agents are approaching each other and the lower sign when they are moving apart. Note, as a mnemonic, that, above the fraction bars, there are quantities with index R and, below the fraction bars, quantities with index S , for both sides of the equation.

Ψ Doppler effect with plus and minus

It is best to consider which signs are to be used for every individual case. An example: $S \rightarrow R \rightarrow$; the sender moves towards the receiver, and the frequency increases ($/(c - v_S)$); the receiver moves away from the sender ($(c - v_R)$), and the frequency decreases; thus $f_R/f_S = (c - v_R)/(c - v_S)$.

Questions

How may Eq. 3.9 be simplified when the receiver is stationary and the sender is approaching him?⁸

⁸ $f_R/f_S = c/(c - v_S)$, the received frequency becomes higher.

What frequency does the receiver hear when he travels at the same speed as the sender, (a) in front of and (b) behind the sender?⁹

Sender overtakes receiver

Let us apply Eq. 3.9 to the situation in which the sender and receiver both move in the same direction, and the sender overtakes the receiver.

Before overtaking, the sender is approaching the receiver, thereby increasing the received frequency (minus sign in the denominator). The receiver is moving away from the sender, also reducing the received frequency (minus sign in the numerator):

$$\frac{f_R}{f_S} = \frac{c - |v_R|}{c - |v_S|} \quad (3.10)$$

After overtaking, it is the other way around: The sender is moving away, the receiver is approaching, and a plus sign must be inserted in both the numerator and denominator:

$$\frac{f_R}{f_S} = \frac{c + |v_R|}{c + |v_S|} \quad (3.11)$$

Remember: $|v_R|$ und $|v_S|$ are speeds (amount of the velocities).

Motion on a straight line, analytical formula (Doppler)

We formulate a general formula in which the signs are automatically correct:

$$\frac{f_R}{f_S} = \frac{c - v_R \cdot \text{sgn}(x_R - x_S)}{c - v_S \cdot \text{sgn}(x_R - x_S)} \quad (3.12)$$

with x_S , x_R being the positions of the sender and the receiver, respectively, on the x -axis. The mathematical function sgn (“signum”) is available as a spreadsheet function `SIGN` and as `np.sign` in `numpy`. To be able to better compare the formula with the previous calculations, we rewrite it with the speeds (amounts):

$$\frac{f_R}{f_S} = \frac{c - |v_R| \cdot \text{sgn}(v_R) \cdot \text{sgn}(x_R - x_S)}{c - |v_S| \cdot \text{sgn}(v_S) \cdot \text{sgn}(x_R - x_S)} \quad (3.13)$$

► **Tim** I could never develop a formula like that. I would always set the wrong sign or change the correct order.

► **Mag** Nor could I. I’ve been toying around with this, checking whether the outcome corresponds to the intuitive formula Eq. 3.9 in ten different situations.

⁹ (a) $f_R/f_S = (c - v_R)/(c - v_R) = 1$; (b) $f_R/f_S = (c + v_R)/(c + v_R) = 1$; the received frequency is, in both cases, equal to the sent frequency.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1													$=\frac{(c \cdot s - vR \cdot s \cdot vR \cdot xRel)}{(c \cdot s - v \cdot S \cdot s \cdot vS \cdot xRel)}$
2													$=SIGN(x.R - x.S)$
3						fE/fS		$s.vR$	$s.vS$	$x.S$	$x.R$	$xRel$	
4	cs	340 m/s	S-->	R*	R*	$1.05 = (cs)/(cs-vS)$		0	1	-1	1	1	1.05
5	vS	17 m/s	S*	S-->	S-->	$0.95 = (cs)/(cs+vS)$		0	1	1	-1	-1	0.95
6	vR	10 m/s	R*	R-->	R-->	$0.97 = (cs-vR)/(cs)$		1	0	-1	1	1	0.97
7			R-->	S*	S*	$1.03 = (cs+vR)/(cs)$		1	0	1	-1	-1	1.03
8			S-->	R-->	R-->	$1.02 = (cs-vR)/(cs-vS)$		1	1	-1	1	1	1.02
9			R-->	S-->	S-->	$0.98 = (cs+vR)/(cs+vS)$		1	1	1	-1	-1	0.98
10			S-->	<--R	<--R	$1.08 = (cs+vR)/(cs-vS)$		-1	1	-1	1	1	1.08
11			<--R	S-->	S-->	$0.92 = (cs-vR)/(cs+vS)$		-1	1	1	-1	-1	0.92
12			<--S	R-->	R-->	$0.92 = (cs-vR)/(cs+vS)$		1	-1	-1	1	1	0.92
13			R-->	<--S	<--S	$1.08 = (cs+vR)/(cs-vS)$		1	-1	1	-1	-1	1.08

Fig. 3.9 (S) Frequency ratio for ten cases, in column F, calculated with Eq. 3.9 and with individual considerations for each case, in column M, calculated with the general formula Eq. 3.13

- ▶ **Alac** Proof by trial and error? You can't do that in math!
- ▶ **Mag** With trial and error, proof is not possible, but serious mistakes can be uncovered. You can later rigorously prove the formula.

Check the analytical formula in a spreadsheet

In Fig. 3.9 (S), ten situations are listed (D:E) in which sender and receiver move to the left or the right or one of them is at rest, and it is also distinguished as to whether the sender is to the left or the right of the receiver. In column F, the frequency ratio is calculated according to Eq. 3.9, “for intuitive use”, considering which signs are to be used for each formula. In column M, the analytical formula Eq. 3.13 is applied relating to the values 1, 0, or -1 indicating the direction of the movements (s.vR, s.vS) and the relative position (x.S, x.R), with respect to the zero of the straight line, of receiver and sender. The quantity x_{Rel} is defined as $sign(x_R - x_S)$.

Checking in Python

In the Python program of Table 3.9, a function *Doppler* is defined realizing Eq. 3.12, taking the velocities v_S and v_R (with correct sign + or -), respectively, of the sender and the receiver, together with a keyword argument *pos* as input and returning the frequency ratio f_R/f_S . The string argument *pos* specifies the relative position of sender and receiver, ‘SR’ indicating that the receiver is to the right and ‘RS’ to the left of the sender. The same parameters as in Fig. 3.9 (S) are specified successively in a list *var1* that is passed to *Doppler* expecting three positional arguments; so, the list *var1* has to be unwrapped (*var1 in line 11). The results of *Doppler* are the same as in Fig. 3.9 (S).

Unwrapping a list with *

The function *Doppler* defined in Table 3.9 expects three positional arguments. The parameters of our lists are, however, specified in one object, the list *var1*. Calling *Doppler(var1)* results in an error message: `TypeError: Doppler() missing 1 required positional argument: 'vR'`. In the call of *Doppler* in line 11, list *var1* has, therefore, been unwrapped (*var1) so that the elements of *var1* are transferred and not the list as one object.

Table 3.9 A general formula for calculating the Doppler shift for sender and receiver moving on the same straight line

<pre> 1 def Doppler(vS,vR,pos = 'SR'): 2 if pos == 'SR': sgnX=1 3 if pos == 'RS': sgnX=-1 4 fR=c-vR*sgnX 5 fS=c-vS*sgnX 6 return fR/fS </pre>			
<pre> 7 c=340.0 8 vS=17.0 9 vR=10.0 10 var1=[vS,0.0,'SR'] 11 print(var1,'{:5.2f}'.format(Doppler(*var1))) </pre>			
[17.0, 0.0, 'SR']	1.05	[17.0, 10.0, 'RS']	0.98
[17.0, 0.0, 'RS']	0.95	[17.0, -10.0, 'SR']	1.08
[0.0, 10.0, 'SR']	0.97	[17.0, -10.0, 'RS']	0.92
[0.0, 10.0, 'RS']	1.03	[-17.0, 10.0, 'SR']	0.92
[17.0, 10.0, 'SR']	1.02	[-17.0, 10.0, 'RS']	1.08

3.3.2 A Sound Source Passes a Remote Receiver

Figure 3.10a illustrates the ride of a car on a straight road, on the line $y = 0$ from $x = -100$ m to $x = 100$ m. At a distance of $y_R = 30$ m off the road, a receiver is at position $(0, 30)$. The car constantly sends out a tone of 200 Hz. Which frequency does the receiver perceive when the car passes by?

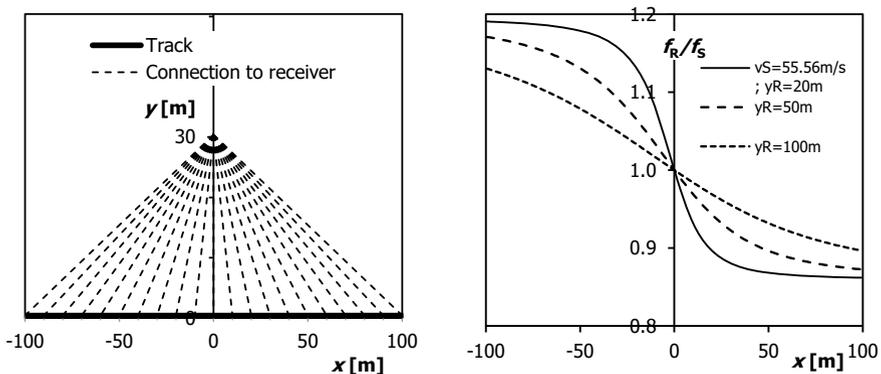


Fig. 3.10 a (left) An observer (receiver) at point $(0; 30)$ hears a car (sender) passing on the x -axis (in the figure compressed). The velocity along the current connection line (dashed) determines the perceived frequency. b (right) The perceived frequency, relative to the frequency of the source, when a sound source passes the receiver at different distances y_R at speed $v_S = 55.56$ m/s

Velocity on line connecting sender and receiver

The velocity along the current connection line (dashed) determines the perceived frequency. The quantities v_R (receiver) and v_S (sender) in Eq. 3.9 are the velocity components on the current connecting line between sender and receiver, i.e., on the dashed lines in Fig. 3.10a.

The velocity must have the correct sign. The upper signs in Eq. 3.9 apply when the car is approaching, and the lower signs when it is driving away. For the situation described above, the sender's velocity component in the direction of the stationary receiver is to be determined by differentiating the distance with respect to time. This yields a negative velocity when the car is left of $x = 0$ and a positive velocity when it is right of $x = 0$, so that the frequency ratio

$$\frac{f_R}{f_S} = \frac{1}{1 + \frac{v_S}{c}} \quad (3.14)$$

is valid for the whole track.

Frequency curves for the distances $y_R = 20, 50,$ and 100 m are displayed in Fig. 3.10b. The closer the receiver is to the track, the stronger the frequency varies with the position of the car. When the sender is at the receiver's height, the received frequency is identical to the sent frequency.

The interesting thing about the computation model in this exercise is that the time-dependent distance of the sender to the receiver is to be calculated (with Pythagoras) at each interval boundary. Then, the velocity component on the connecting line is obtained through numerical differentiation. In this way, more complicated geometries can also be treated, e.g., when the transmitter is moving at varying speed on a circular path, or if transmitter and receiver are moving on different paths.

3.3.3 Data Structure and Nomenclature

c	speed of sound in air
v_S	velocity of sender in x direction
v_R	velocity of receiver in x direction
f_S	frequency of sender
f_R	frequency at receiver
x_S	position of sender
x_R	position of receiver
x	array of 201 equidistant positions of a sender on the x -axis
$t(x)$	points of time corresponding to x
y_R	distance of the receiver to the x -axis
$dist$	current distance sender-receiver, $dist = \sqrt{x^2 + y_R^2}$.
v	relative speed along the line connecting sender and receiver

	A	B	C	D	E	F	G	H	I
1			200.00	km/h					
2	Velocity of sender	vS	55.56	m/s					
3	Frequency of sender	fS	200.00	Hz			=B2&"="&ROUND(vS;2)&"m/s; "		
4	Speed of sound	c_	340.00	m/s				&B5&"="&yR&"m"	
5	Distance of receiver	yR	20.00	m			vS=55.56m/s; yR=20m		
7			=B9+1	=C9+(x-B9)/vS	=SQRT(x^2+yR^2)	=fS/(1+v/cS)	=f/fS	=(x+B9)/2	
8		x	t	dist	v	f	fNorm	xC	
9		-100	0	101.98					
10		-99	0.02	101.00	-54.47	238.15	1.19	-99.50	
209		100	3.60	101.98	54.47	172.38	0.86	99.50	

Fig. 3.11 (S) A sound source moves on the x -axis past an observer at a distance d_R . The x position in column B is the independent variable. From that, the time (column C), the distance source-observer (column D), the velocity of the sender in the direction of the connecting line (E), and the observed frequency (F, G) are calculated with Eq. 3.14. Attention: The time in column C depends on the speed of the source! In column H, x_C is the center of the intervals

3.3.4 Spreadsheet Calculation “Remote Receiver”

A possible calculation model is shown in Fig. 3.11 (S), where we have chosen the x -coordinate of the car on the track as the independent variable, and time, distance, and velocity along the connecting line as dependent variables. You could just as easily choose time as the independent variable. The x in the formula C10 = [= C9 + (x - B9)/vS] refers to B10, the entry of the column vector x in the same row.

- ▶ **Mag** The frequency ratio is calculated numerically in columns F and G. Over which local coordinates do you plot the calculated frequencies? Perhaps over x in column B?
- ▶ **Tim** Well, since you asked it in that way, it probably isn't. I remember this much: We take the centers of the considered distance intervals because the speeds were calculated with the (t, x) coordinates of the interval boundaries.
- ▶ **Mag** Right! The centers of the intervals are calculated as x_C in column H.

3.3.5 Python Program “Remote Receiver”

A Python program that solves the task is given in Table 3.10, the main program in the upper cell, the function *FreqLine* for realizing Eq. 3.14 for the complete frequency curve in the lower cell. List slicing is used to calculate the interval centers x_C in the main program, and the velocities v when differentiating the distance *dist* in *FreqLine*.

Table 3.10 **a** (top) Specifications of the situation in which a receiver is at rest at a distance y_R off the sender's track. **b** (bottom) Function for calculating the frequency curve when y_R is given

```

1  vS=55.56           #Speed of sender
2  fS=200             #Frequency of sender
3  c=340              #Speed of sound
4
5  x=np.linspace(-100.0,100.0,201)
6  #Center xC of path segments
7  xC=(x[1:]+x[:-1])/2
8
9  #Time segments Dt
10 Dt=(x[1:]-x[:-1])/vS
11 #Time t
12 t=np.cumsum(Dt)    #Integrates over dt
13 def FreqLine(yR):
14     dist=np.sqrt(x**2+yR**2)
15     v=((dist[1:]-dist[:-1])/Dt)
16     fRS=1/(1+v/c)
17     return fRS     #Frequency ratio

```

Questions

concerning Table 3.10:

What are the arguments and the global parameters accessed in *FreqLine*?¹⁰

The local variable x_C over which the received frequency is to be plotted is calculated in the main program. Would it be more consistent to calculate x_C in *FreqLine* and return it together with the normalized frequency (def *FreqLine*(x , y_R) ... return x_C , f_{Res})?¹¹

Question

concerning Table 3.11

What does the keyword argument $lw=1.5$ in the plot function specify?¹²

What does the label in line 8 look like?¹³

How do we specify that the figure is saved with a resolution of 1200 dpi?¹⁴

The plot program in Table 3.11 calculates and displays the frequency curves for the three distances $y_R = 20$, 50, and 100 m.

¹⁰ *FreqLine*, argument: y_R , global parameters x , $dist$, c , Dt .

¹¹ Discuss!

¹² Line width $lw = 1.5$ point.

¹³ The label is “ $y_R = 20$ ”.

¹⁴ `plt.savefig('Doppler off, multiple.png', dpi=1200)`, `dpi` is a key word argument.

Table 3.11 Plotting several frequency trajectories with the parameters specified in Table 3.10

```

1 FigStd('x',-100,100,25,'f/fQ',0.8,1.2,0.1)
2 plt.plot([-100, 100],[1,1],ls='-',color='k',lw=1)
3           #Horizontal through y=1
4 yR=20           #Distance to track
5 fRS=FreqLine(yR)
6 plt.plot(xC,fRS,'k-', lw=1.5,label='yR='+str(yR))
7 yR=50; fRS=FreqLine(yR)
8 plt.plot(xC,fRS,'k--',lw=1.5,label='yR='+str(yR))
9 yR=100; fRS=FreqLine(yR)
10 plt.plot(xC,fRS,'k-.',lw=1.5,label='yR='+str(yR))
11 plt.legend(loc=0,fontsize=10)
12 #plt.savefig('Doppler off, multiple.png')

```

3.4 Exponentials

For exponential functions, apply Ψ *Plus 1 yields times e*; plus 1 in the argument yields times e in the value. An exponential function $A \cdot \exp(-t/t_0)$ is best drawn by hand (Really? Yes, also by hand!), beginning with its tangent at $t = 0$. Diode characteristics seemingly exhibit a “kink voltage” that depends on the scaling of the y -axis; more generally, exponential functions seem to explode.

3.4.1 Explosive Character of Exponentials

Rice grains on a chessboard

In a classic bet from far eastern literature, a clever chess player agreed with his king that his winnings would be paid by placing a rice grain on the first square of a chessboard and then doubling the number of grains repetitively on each of the following 63 squares.

In Fig. 3.12a (S), the doubling of the number of rice grains is simulated with a spreadsheet calculation. In A3:A66, the 64 squares of the checkerboard are numbered from $n = 0$ to 63. In B3, a rice grain is placed on the first field ($n = 0$, B3 = [1]). In the following cells, the preceding number is doubled until the huge number $2^{63} = 9.22 \times 10^{18}$ is reached in B66 ($n = 63$), on the 64th square.

- ▶ Ψ *Plus (in the argument) yields times (in the result) for exponentials.*
For $y = 2^x$, plus 1 (in the argument) yield times 2 (in the result).

	A	B	C	D	E
1	=A3+1	=B3*2	=2^n		=2^D3
2	n	y	y = 2^n		2^62.4
3	0	1	1	62.4	6.1E+18
4	1	2	2		
5	2	4	4		
6	3	8	8		
7	4	16	16		
8	5	32	32		
9	6	64	64		
66	63	9.22E+18	9.22E+18		
67	64	1.84E+19	1.84E+19		
1025	1022	4.5E+307	4.5E+307		
1026	1023	9.0E+307	9.0E+307		
1027	1024	#NUM!	#NUM!		

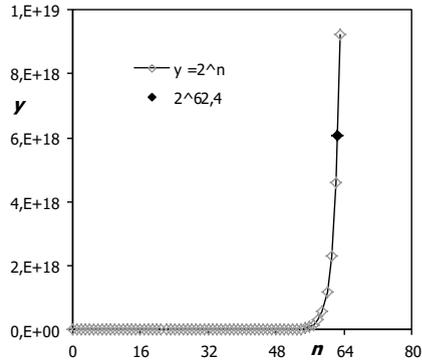


Fig. 3.12 a (left, S) Powers of 2, $y = 2^n$, obtained by repetitive multiplication by 2 (column B) and by potentiation (C: E). The formula in E1 refers to E3 **b** (right) Graphical representation of the powers of 2 of **a**

The values are graphically displayed in Fig. 3.12b. You can see that the explosion by a factor 10^{19} takes place on the last few squares. On squares 0–62, there are $2^{63}-1$ grains, on all squares together, $2^{64}-1$ grains.

Profit

Question

What would be the share of the internationally traded rice in 2016 if the loser of the game (a wealthy medieval Sultan) had been able to deliver?¹⁵

What is the value of 10E3 in EXCEL and 10e3 in Python? Be careful!¹⁶

- ▶ **Mag** Can the winner satisfy his hunger with his win?
- ▶ **Alac** Perhaps once.
- ▶ **Tim** I've heard that exponential growth means explosion. So, maybe the winner can live well on his heap of rice for a week.
- ▶ **Mag** World rice production in 2015/16 was 470 million metric tons, but only about 5% were traded on the world market. Unlike wheat, rice is consumed by more than 95% of the population in the cultivating countries.

¹⁵ For 64 fields, the winner would have received $2^{64}-1 = 18 \times 10^{18}$ grains, corresponding to about $10^{18} \text{ g} = 10^{12}$ tons of rice. This is the 2000-fold amount of the rice harvest 2015/16 of 470 million tons.

¹⁶ $10E3 = 10 \times 10^3 = 10^4 = 10,000$, the same with 10e3 in Python.

► **Tim** I've counted. One kilogram of rice contains about $40,000 = 4 \times 10^4$ grains. According to the rule of the game, more than 10,000 times the volume of one year's world trade of rice should pile up on the chessboard. Incredible!

► **Alac** Crazy! A disaster! This cannot be true! What's the catch?

► **Mag** There is no catch. The catastrophe results from the rule Ψ plus 1 yields times 2 governing the exponential 2^n .

The power function $y = 2^n$ is calculated in column C of Fig. 3.12a (S). It is entered as the worksheet formula [= 2^N] and yields the same results as the repetitive multiplication by 2 in column B. The argument of the power function does not have to be an integer. In E3, the value for the power in D3 (= 62.4) is calculated and inserted as a filled diamond in Fig. 3.12b.

Maximum float number

We continue doubling the preceding number in the worksheet beyond $n = 63$ until the application can no longer store the resulting number. As of row 66 in Fig. 3.12a (S), the numbers are represented in exponential form, $9.22\text{E}+18^{+18} = 9.22 \times 10^{18}$. The number 2^{1024} can no longer be calculated in EXCEL2019; see the error message #NUM! in row 1027.

In Python 3, the int format can store arbitrarily large numbers. However, if the number is to be calculated as float, the same limit holds as in Excel: $2^{**}1023$ gives $8.988\text{e}+307$ and $2^{**}1024$ returns the `OverflowError: int too large to convert to float`.

3.4.2 General Exponential Function

The power function can be generalized to $y = a^x$, where a and x are real numbers. If a is Euler's number $e = 2.718$, then the power function becomes the known exponential function with the formula [= EXP(...)] (EXCEL) or `np.exp()` (Python).

Exponential function with characteristic length

The normalized exponential function is usually written in mathematical textbooks as follows:

$$f(x) = |a| \cdot \exp(ax) = |a| \cdot e^{ax} \quad (3.15)$$

The letter e denotes Euler's number $e = \sum_0^{\infty} \frac{1}{n!} = 2.718$. Normalized means that the integral from 0 to $+\infty$ (for $a < 0$) or from 0 to $-\infty$ (for $a > 0$) is 1. The integral is dimensionless because the product of the units of dx and a is 1.

It is, however, often physically more sensible to write the exponential with a characteristic x value x_0 :

$$f(x) = \left| \frac{1}{x_0} \right| \cdot \exp\left(\frac{x}{x_0}\right) = \left| \frac{1}{x_0} \right| \cdot e^{\frac{x}{x_0}} \tag{3.16}$$

Thus, the unit of x_0 is equal to the unit of x , e.g., a length or a time, and has an intuitive meaning: the tangent at $x = 0$ intersects the y -axis at the amplitude $|1/x_0|$, and the x -axis at the characteristic length x_0 .

When the function is specified with an amplitude $\frac{A}{|x_0|}$, its integral from 0 to, respectively, $+\infty$ (for $a < 0$) or $-\infty$ (for $a > 0$), is A .

3.4.3 Representation in a Diagram

First, the tangent at $t = 0$!

Two exponential functions, together with their tangents at $t = 0$, are shown in Fig. 3.13a. Here, the independent variable is time t .

► **Mag** Do you now know how to draw an exponential function $A_e \cdot \exp(t/t_0)$ by hand on a piece of paper?

► **Alac** Sure! First, mark the amplitude A_e on the vertical axis and the characteristic time t_0 on the horizontal and pass a straight line through the two points. The exponential curve approaches the tangent at $t = 0$ and the horizontal axis for $t \rightarrow \infty$ or $t \rightarrow -\infty$, depending on the characteristic parameter's sign.

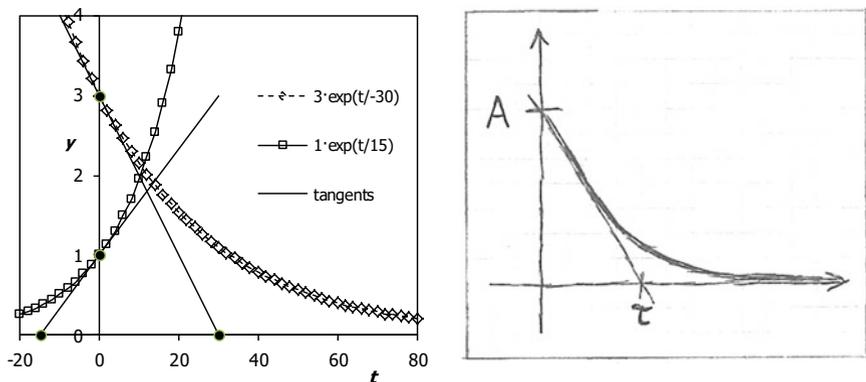


Fig. 3.13 a (left) Two exponential functions $A_e \cdot \exp(t/t_0)$ with their tangents at the intersections with the y -axis. b (right) How to draw an exponential function by hand: First, the tangent as a straight line with its intersections with the x - and y -axes!

- ▶ **Mag** Correct, just as in Fig. 3.13b! Keep in mind:
 - ▶ Ψ Plus one in x , times e in y
 - Ψ Expo with kink and straight line
- ▶ **Tim** “Plus one” and “Straight line” are clear, but why “kink”?
- ▶ **Mag** You have seen this in Fig. 3.12b; it will be explained in Sect. 3.4.4. An essential feature of an exponential is an explosion on a suitably scaled y -axis.

3.4.4 Diode Characteristics $I(U)$

The current I through a semiconductor diode depends exponentially on the applied voltage U . The $I(U)$ characteristics of a semiconductor diode are described by an exponential function passing through zero:

$$I = I_s \cdot \left(\exp\left(\frac{U}{U_T}\right) - 1 \right) \tag{3.17}$$

This function has two parameters: the strength of the reverse current I_s also called the saturation current, and the thermal voltage U_T , which is given by $k_B T/e$, with k_B being the Boltzmann constant, e the elementary charge, and T the absolute temperature. At room temperature, $U_T = 25$ mV. The current through a diode is zero when the applied voltage is zero.

We will represent such a function for $I_s = 1 \times 10^{-14}$ A and $U_T = 0.025$ V in various plots (see Fig. 3.14).

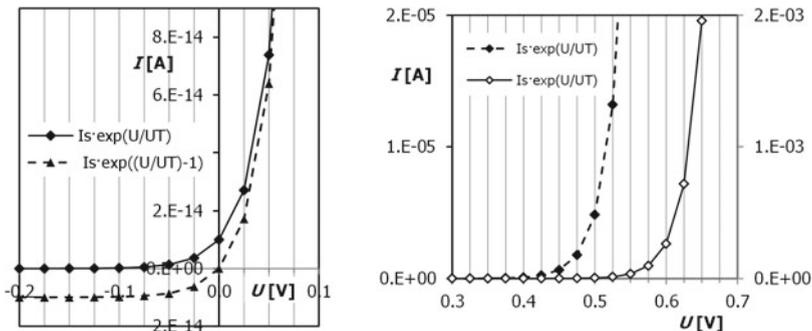


Fig. 3.14 a (left) Diode characteristics and associated exponential function, representation for small currents. b (right) Twice the same exponential as in a, but with different scaling of the I axes; left y -axis for the left curve, right y -axis for the right curve; the vertical grid lines have a distance $U_T = 25$ mV

Questions

Which of the curves in Fig. 3.14a, b are exponential functions? Which are shifted on the I -axis? Which are shifted on the U -axis?¹⁷

At which U -values in Fig. 3.14b are you most likely to find the “kink points”, in electronic engineers’ jargon?¹⁸

Also shown in Fig. 3.14a is an exponential function without the term -1 in the parentheses of Eq. 3.17. This function intersects the I -axis at I_s . It grows by a factor of $\exp(1) \approx 2.7$ whenever U progresses by dU . The curves in Fig. 3.14a appear to be exponential, the way they are usually represented.

For exponential functions to the base e , the broom rule is *Ψ Plus 1 yields times e*. In the concrete case, this means that, if U advances by U_T (the distance between the vertical gridlines in Fig. 3.14), \exp increases by a factor of about 2.7.

In Fig. 3.14b, the same data as in Fig. 3.14a are shown twice, only with different scaling of the two I -axes: I_{Max} is now at 2.5×10^{-5} A (left vertical axis) or 2.5×10^{-3} A (right vertical axis); the curves seem to exhibit a kink at about 0.5 and 0.6 V. In electrical engineering, this voltage is called the “kink voltage”, “knee voltage” or “cut voltage”.

The increase of the U -value by a factor of 2.7 when progressing from $U = 0.475$ to $U = 0.500$ V looks like a steep rise. It increases, however, by the same factor when it progresses from $U = 0$ to $U = 0.025$ V but then appears like the familiar soft curvature of the exponential. The position of the seeming kink on the U -axis is a function of the scaling of the I -axis.

In Fig. 3.15a, the I -axis is logarithmic. In this representation, the diode characteristics appear as a straight line, except for the points below $U = 0.05$, because, there, the term -1 is quantitatively significant. A kink is nowhere to be seen.

3.4.5 Data Structure and Nomenclature

Exponential function

A, B	amplitudes of functions \exp_A and \exp_B
t_A, t_B	time constants of \exp_A and \exp_B
dt	length of time interval
t	array of instants of time, separated by dt
\exp_A, \exp_B	values at times t .

¹⁷ All curves represent exponential functions. The diode curve in Fig. 3.14a is shifted downwards on the I -axis by the saturation current I_s , so that it passes through zero. The curves in Fig. 3.14b have not been shifted on the U -axis.

¹⁸ At about 0.5 and 0.6 V.

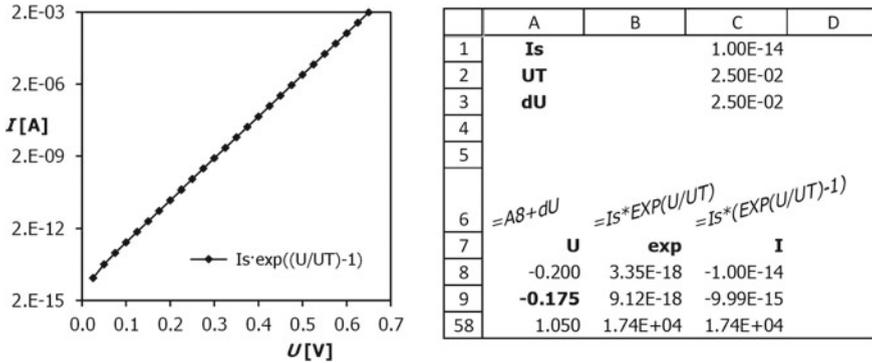


Fig. 3.15 a (left) Diode characteristics, logarithmic scale of the I axis (semi-log plot). The curve can only be represented for $I > 0$, because the logarithm is defined only for positive values. b (right) Spreadsheet layout for calculating diode characteristics, I_s = saturation current, U_T = temperature voltage, $dU = U_T$

Diode characteristics

- I_s saturation current
- U_T thermal voltage, 25 mV at room temperature, characteristic parameter of the exponential
- dU interval width, here, $dU = U_T$
- U sequence of voltages, dU apart
- exp exponential function with I_s and U_T as amplitude and characteristic voltage
- I diode current, $I(U)$.

3.4.6 Spreadsheet Calculation

Initial slope of the exponential function

In Fig. 3.16 (S), two exponential functions are calculated for 51 points equidistant in t ,

$$exp_A(t) = A \cdot exp\left(\frac{t}{t_A}\right) = A \cdot e^{\frac{t}{t_A}} \tag{3.18}$$

and exp_B correspondingly. The characteristic time, t_A or t_B , may be positive or negative. The distance dt between the points is set in B3.

- For the exponential function $y = e^x$, applies; Ψ Plus 1 (in the argument) yields times e (in the value).
For the power of 2, $y = 2^n$ holds: Ψ Plus 1 yields times 2.

	A	B	C	D	E	F	G	H	I
1	A	3.0	tA	-30.0		$3 \cdot \exp(t/-30)$	$=A \cdot \exp(t/tA)$		
2	B	1.0	tB	15.0		$1 \cdot \exp(t/15)$			
3	dt	2.0							
4	$=A6+dt$	$=A \cdot \text{EXP}(t/tA)$		$=B \cdot \text{EXP}(t/tB)$					
5	t	expA	expB			tangents			
6	-20.0	5.8	0.3			30	$=-tA$	0.0	
7	-18.0	5.5	0.30			0.0		3	$=A$
8	-16.0	5.1	0.34			-30	$=tA$	6	$=2 \cdot A$
56	80.0	0.2	207.13						

Fig. 3.16 (S) Two exponential functions whose independent variable t is specified as column vector A6:A56 and their parameters amplitude A , B and time constant t_A , t_B are given in A1:D2. The legends for the functions are compiled in F1 and F2 with the formula in G1 of type Ψ “Text” & Variables

The table in Fig. 3.16 (S) has a typical Γ layout. The 51 values of the independent variable time t are located to the left of Γ in a column vector named “t”. The parameters A , B , and t_A , t_B of the curves are specified above Γ , as well as the time interval dt , the distance on the horizontal axis between the calculation points. The initial value of t , here, -20 , has to be entered into cell A6. The values for the remaining 50 t values are determined successively from the respective predecessor. The t -predecessor for cell A7 is cell A6, $A7 = [A6 + dt]$. In the formula, A6 is not provided with a dollar sign. It is a relative reference, so that the address of the addressed cell adapts during copying. Therefore, $A56 = [A55 + dt]$.

Questions

Suppose that A6 contains an initial time $t = 5$ and the length of a time segment is stored in a cell with the name dt and has the value 2. What values for time t are in cells A7 and A8?¹⁹

How can the coordinates of the tangent to the exponential $A_e \cdot \exp(t/t_0)$ at $t = 0$ be derived from the parameters of the exponential function?²⁰

What is the distance on the horizontal axis between the functions’ calculation points in Fig. 3.13a created from Fig. 3.16 (S)?²¹

What is the value of $\int_0^\infty 3 \cdot \exp(-\frac{t}{30}) dt$?²²

► **Task** Change the parameters A , t_A , and dt and check whether the diagram reacts accordingly! This is the case if each cell contains the correct formula. Remember:

¹⁹ $A6 = [5]$, $A7 = [A6 + dt] = 7$; $A8 = [A7 + dt] = 9$.

²⁰ Straight line through the points $(0, A)$ and $(-t_e, 0)$.

²¹ The distance of adjacent points is $dt = 2$ (see Fig. 3.16 (S), B3).

²² The definite integral is 90, based on Eq. 3.16, because $3 = \frac{1}{30} \cdot 90$.

you cannot merely transfer the numbers from the figures to your worksheet. Most cells contain a formula; only sometimes are numerical values entered directly.

Diode characteristics

A possible calculation model is presented in Fig. 3.15b (S).

The parameters I_s and U_T of the diode characteristics are set in the named cells C1:C2. In C3, the horizontal distance between neighboring sampling points is defined. Here, we have chosen $dU = U_T = 0.025$ V that is valid for room temperature. The I - U characteristics are illustrated in Fig. 3.14. They intersect the I -axis at 0, showing that no current flows without applied voltage.

Formatting the axis of a diagram

The axis of a diagram is formatted by activating it with the left mouse button and then clicking FORMAT. In the CURRENT SELECTION group to the left of the ribbon, VERTICAL (VALUE) AXIS shows up in the bar. Upon clicking FORMAT SELECTION, a window opens that allows you to set the minimum, maximum and other parameters of the axis. If the axis is to be scaled logarithmically, the appropriate box, LOGARITHMIC SCALE, must be activated with a checkmark.

3.4.7 Python Program

Initial slope

A Python program for drawing the exponentials of Fig. 3.13a and their slopes at $t = 0$ is given in Table 3.12.

The list t of time instants is created by `np.arange(-20, 80 + dt, dt)`, mimicking the construction of the time vector in the spreadsheet of Fig. 3.16 (S) with, e.g., A8 = [=A7 + dt]. The lower limit -20 corresponds to the value in A6. To come to 80 (in A56), we have to specify $80 + dt$ as the upper limit, because `np.arange` creates an interval that is open at its right end with the endpoint excluded. exp_A and exp_B are constructed from t with the corresponding amplitudes and time constants

Table 3.12 **a** (top) Specifications of two exponential functions, equivalent to rows 1 through 3 in Fig. 3.16 (S); **b** (bottom) labels for the two functions created in lines 7 and 8

1	<code>dt=2.0</code>	
2	<code>A,tA= 3.0,-30.0</code>	<code>#Amplitude and time constant</code>
3	<code>B,tB=1.0,15.0</code>	
4	<code>t=np.arange(-20,80+dt,dt)</code>	
5	<code>expA=A*np.exp(t/tA)</code>	
6	<code>expB=B*np.exp(t/tB)</code>	
7	<code>lblA=(str(A)+'*'+str(t/tA)+'')</code>	
8	<code>lblB=(str(B)+'*'+str(t/tB)+'')</code>	
	<code>lblA 3.0*exp(t/-30.0)</code>	
	<code>lblB 1.0*exp(t/15.0)</code>	

Table 3.13 Specifications for diode characteristics, the same as in Fig. 3.15b

```

1  Is=1e-14
2  UT=2.5e-2
3  dU=2.5e-2
4  U=np.arange(-0.2,1.05+dU,dU)
5  exp=Is*np.exp(U/UT)
6  I=Is*(np.exp(U/UT)-1)      #Diode characteristics

```

(A, t_A) and (B, t_B). The curves can be plotted with our standard function `StdFig`. In Fig. 3.13a, they are represented with open symbols. In Python, this is specified by `plt.plot(..., fillstyle='none', ...)`.

Questions

What are the instructions for plotting the tangents in Fig. 3.13a?²³

What are the size and last element of:

- L1 = `np.arange(-20, 80+dt, dt)` with $dt = 1$,²⁴
- L2 = `np.linspace(-20, 80, 100)`,²⁵
- L3 = `np.linspace(-20, 80, 101)`²⁶
- L2 = `np.linspace(-20, 80, 100, endpoint=False)`.²⁷

Diode characteristics

A Python program corresponding to the spreadsheet layout in Fig. 3.15b is shown in Table 3.13.

Subplots

A plot like Fig. 3.14b with two vertical axes cannot be achieved with our standard figure, the function `FigStd` defined in Sect. 2.4.5. We have to refer to the function `subplots` of the `matplotlib` library (see Table 3.14). $I(U)$ is plotted twice, with `ax1.axis([0.3, 0.7, 0, 2e-5])` for the primary (left) y-axis from 0 to $2e-5$ and with `ax2.axis([0.3, 0.7, 0, 2e-3])` for a second y-axis from 0 to $2e-5$, declared with `ax2 = ax1.twinx()` as the secondary (right) y-axis.

Logarithmic scaling

Logarithmic scaling of an axis can be achieved with `plt.yscale(value="log")` and `plt.xscale(value="log")`.

²³ `plt.plot([0, tA], [A, 0], 'k-')`, compare F5:H8 in Fig. 3.16 (S)!

²⁴ L1, size 101, last element `L1[-1]` is 80.

²⁵ L2, size 100, last element 80.

²⁶ L3, size 101, last element 80.

²⁷ L4, size 100, last element $80 - 100/100 = 79$.

Table 3.14 Setting up a diagram similar to Fig. 3.14b with primary and secondary axes, variables specified in Table 3.13

```

1  fig, ax1 = plt.subplots()
2  ax1.plot(U, I, 'k-')
3  ax1.axis([0.3, 0.7, 0, 2e-5])
4  for x in np.arange(0.3,0.7,dU):
5      ax1.plot([x,x],[0,2e-5],'k-',lw=0.6)
6
7  ax2 = ax1.twinx()      #Secondary axis
8  ax2.axis([0.3,0.7,0,2e-3])
9  ax2.plot(U, I, 'r--')
10 plt.show()

```

3.5 Questions

General advice

1. To practice programming, translate the spreadsheet solutions of this chapter into Python and compare with the suggested programs!

Python-specific

2. Let $x = \text{np.linspace}(-100, 100, 101)$. What is the distance between neighboring elements? What does $x[1:] - x[:-1]$ look like? What are the first and last elements of $(x[1:] + x[:-1])/2$?
3. Let $A = 3.001$ and $t_A = -30.0$. How can we produce a label “ $3.0 * \exp(t - 30)$ ”?

Concerning Table 3.15:

4. What is the size of the list *col*?
5. What are the first two elements of *col*?
6. What are the last two elements of *col*?
7. How do you compile the four lines into one line with list comprehension?

Check your answers by programming!

Table 3.15 Producing a list $x*y$

```

1  col=[]
2  for x in range(1,11):
3      for y in range(1,6):
4          col.append(x*y)

```

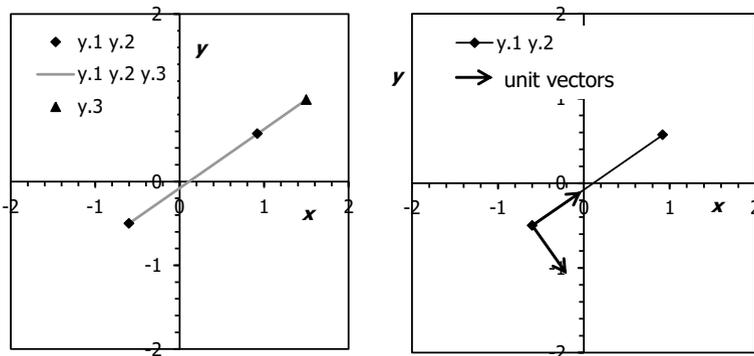


Fig. 3.17 **a** (left) The segment defined by two points (x_1, y_1) and (x_2, y_2) is extended to a third point whose x -coordinate x_3 can be selected arbitrarily. **b** (right) The unit vectors in the direction of the line and perpendicular to the line are attached to the line, defined in Fig. **a**, at the left point

Broom rules

- 8. Explain the broom rules:
 - Ψ Lens equation with plus and minus.
 - Ψ Blank lines separate curves.
 - Ψ Doppler effect with plus and minus.

Straight-line equation

A straight line is defined by two points (x_1, y_1) and (x_2, y_2) (see Fig. 3.17a).

- 9. What is the equation for determining the distance between the two points?
- 10. Which equation must be used to find the value y_3 for a given horizontal position x_3 so that (x_3, y_3) lies on the straight line?
- 11. Deduce from the coordinates of the two points the direction vector (D_x, D_y) of the straight line normalized to length 1!
- 12. How do the coordinates of the two points result in the vector (P_x, P_y) , the perpendicular to the straight line and normalized to length 1?
 A spreadsheet layout for three points on a straight line is shown in Fig. 3.18 (S).

- 13. What is the linked cell and MIN and MAX of the slider in F4:H4?
- 14. What formulas are in B3 (input from E3) and D4 (input from E4)?
- 15. Write a Python program that performs the calculations of Fig. 3.18 (S)! Replace the function of the sliders with simple assignments with random functions: E1= ... ; E4= ... !

	A	B	C	D	E	F	G	H	I	J	K	M	N	P
1	Three points on a straight line					Unit vectors								
2	x.1	-0.6	y.1	-0.5							along the straight line			
3	x.2	0.92	y.2	0.57	92	◀ [Slider] ▶				x.1	-0.6	y.1	-0.5	
4	x.3	1.5	y.3	0.98	57	◀ [Slider] ▶				x.p	-0.03	y.p	-0.1	
5	perpend. to the straight line													
6	Length of segment 1-2			Slope of segment 1-->2			x.1				-0.6	y.1	-0.5	
7	l.12	2.66			m.12	0.70	x.v				-0.2	y.v	-1.07	

Fig. 3.18 (S) Spreadsheet layout used to create the coordinates for Fig. 3.17a and b, B3 and D4 depend on the outputs on the sliders. The slider in F4:H4 goes from 0 to 80

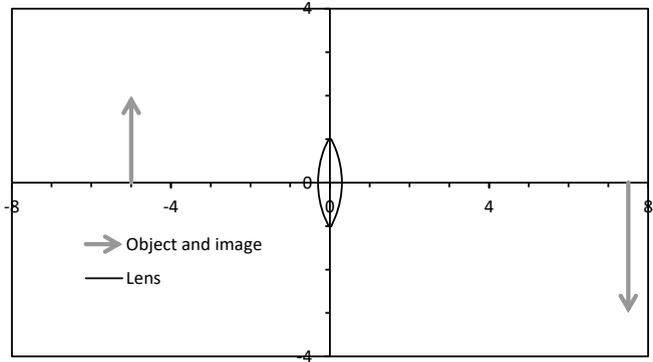


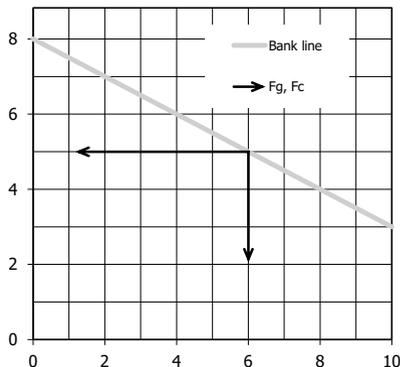
Fig. 3.19 Incomplete image construction for imaging with a converging lens

Image construction with converging and diverging lenses

16. In school, one usually learns the equation $1/f = 1/o + 1/i$ for imaging with converging lenses (o is object distance, i is image distance). How is this imaging equation modified according to DIN 1335²⁸ and made suitable for numerical calculation in spreadsheets and Python both for converging and diverging lenses?
17. How is the magnification factor defined in DIN 1335?
18. What characterizes a converging lens in the imaging equation?
19. What characterizes a diverging lens in the imaging equation?
20. Draw the rays for the image construction in Fig. 3.19!
21. What is the image-side focal length?
22. Draw the bundle of rays that contributes to the image formation!

²⁸ Equations 3.5 and 3.6.

Fig. 3.20 Gravitational F_g and centrifugal F_c force when driving through a curve



Forces when driving through a curve

- 23. What are the formulas for the gravitational force F_g and the centrifugal force F_c on a car of mass m , traveling with speed v through a curve with radius r ?
- 24. What do you have to do to get a true-angle display when the x -axis is scaled from -2 to 8 km and the y -axis from 10 to 15 km, in EXCEL and in Python?
- 25. How is the static friction force defined? What does a static friction coefficient $\mu = 0.5$ mean?
- 26. Draw in Fig. 3.20, with a triangle ruler, the resulting force, the force in the plane, and the force perpendicular to the plane! Which force determines the static friction?
- 27. Draw a vector in the bank line and another one perpendicular to it!
- 28. What are the vector equations for determining the quantities of Question 26?

Doppler effect

$$\frac{f_E}{f_Q} = \frac{c \pm v_E}{c \mp v_Q} \tag{3.19}$$

- 29. What do the letters in Eq. 3.19 stand for? Which signs are to be used when? Adjust the formula for the three cases in Fig. 3.21!

A car (sender S) drives along the x -axis. Its position at time t is specified in an array named x_S . A pedestrian (receiver R) moves along the y -axis towards the x -axis. Its position at time t is indicated in an array named y_R .

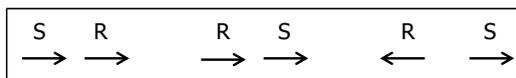


Fig. 3.21 A sound source S and a sound receiver R move on a straight line

30. What is the distance d_{SR} between car and pedestrian as a function of t ?
31. What is the formula for calculating the relative velocity v_{SR} in the direction of the link line? To answer this question, you have to define vectors x_R , y_R and d_{SR} .

Exponential function

32. To draw an exponential function freehand on paper, it is useful to start with a straight line as a guide. How is this straight line determined by the exponential function parameters, amplitude A and time constant τ ?
33. An exponential function increases from 1 to 2 when the argument is increased from $t = 0.0$ to 0.1 s. How much does it increase if the argument is increased from 0.0 to 0.2 s? Think binary! How much does it increase if the argument is increased from $t = 0.8$ to 1.0 s? Sketch the same exponential function twice, each time with the t -axis from 0 to 1.2, but with a y -axis scaling from 0 to 4 for the first sketch and from 0 to 16×4 for the second sketch! What do you see in the second sketch that is not obvious in the first sketch?

Macros with Visual Basic and Their Correspondences in Python

4

We practice the *basic programming structures*: loops, branches, sub-routines (FOR, IF, SUB/def), with particular emphasis on data exchange between spreadsheets and procedures. We will learn to obtain the sequences for EXCEL-typical spreadsheet operations by recording the associated commands with a macro recorder. With this knowledge, we will:

- draw dense-packed crystal planes,
- decode protocols of measuring instruments and compile clear summaries of the results,
- systematically modify the parameters of calculation models with log and control procedures and continuously enter the results of the calculations into another range of the spreadsheet, and
- outsource complicated formulas into user-defined spreadsheet functions in order to arrange the tables more clearly.

We will present parallel solutions in Python creating the drawings with the library `turtle`.

4.1 Introduction: For, If, Sub/Def

Solutions of Exercises 4.3 (Excel), 4.5 (Python), 4.7 (Excel), and 4.8 (Python) can be found at the internet address: go.sn.pub/gTtbiH.

Tim worries, Alac brags

► **Mag** This chapter will teach us to program, to let macros interact with spreadsheet calculations, and to realize parallel solutions in Python.

► **Tim** That sounds pretty demanding. Is it at all manageable for beginners like me?

► **Mag** Quite clearly: Yes. Many people have already achieved that, even students who had never before written a line of code. Visual Basic is a good-natured programming language that does not require much knowledge, at least not for the tasks that we want to tackle. This chapter will not only take away your fear of programming, but you will also find it fun to write programs that do amusing things.

► **Tim** Well, people who had just finished the course told me that they had to deal with routines, macros, programs, and procedures.

► **Mag** Don't worry, we don't make any distinction among those terms, we use them all synonymously. Our programs include both EXCEL-typical command sequences and classical algorithmic structures.

► **Alac** I'm not afraid of VISUAL BASIC or Python; after all, I already took a course about another programming language and used it to write funny programs.

► **Mag** That is certainly a good prerequisite for faster success. You will have an easier time than Tim. Nevertheless, don't take the tasks for granted. In our course, programming tasks are combined with physics exercises (so much for our hopes for just "funny":-)). I've often experienced instances in which mere programmers have been discouraged by their limited progress in this kind of programming and have given up.

► **Alac** So, I'm essentially learning more physics?

► **Mag** Not only that. You should certainly understand more about physics after the course than before. Nevertheless, the exercises will also familiarize you with good programming skills: to develop systematically, to document carefully, to detect and correct mistakes.

► **Tim** One more question. Many workplaces require programming skills in special programming languages. Wouldn't it be better for me to learn such languages from the onset?

► **Mag** Don't worry.' In our tasks with VISUAL BASIC and Python, you will have room to make a sufficient number of mistakes to learn from so that you can be confident in becoming a computer expert. The algorithmic constructions are the same in all programming languages. More important than acquiring special knowledge at an early stage is that you gain the ability to cope with "hard" programming tasks and master the rules for good programming.

How do we proceed?

This course cannot give a general introduction to programming, because that would require an excess of repetition of things that are already well described in specific textbooks. As in Chap. 2, we will do a basic exercise that you should follow step by step and convert into your own program. It contains all of the commands and constructions that we will need later, but not much more. However, this basic knowledge will enable you to find your way around in EXCEL help or Python internet aid and choose suitable textbooks for programming with VISUAL BASIC and Python.

Only a few algorithmic constructions

In our programs, we use constructions that are the same in all algorithmic languages: *Loops, logical branches, sub-routines* (FOR, IF, SUB/DEF). Special EXCEL instructions, e.g., for handling files or creating drawings, will be provided to us by the macro recorder, which records commands that the user makes when setting up spreadsheet calculations.

Three types of program will be used repeatedly in the following chapters: log procedures, formula procedures, and user-defined functions.

Rep-log and scan-log procedures

Two tools can be used sensibly for every task in EXCEL: Sliders and log procedures. With a slider, variables in a spreadsheet can easily be changed manually, and the user can see how the results change, as we have learned in the basic Exercise 2.2. A *rep-log procedure* systematically changes independent variables in the spreadsheet and continuously writes the calculation results into another spreadsheet range. The spreadsheet calculation works like a function. The equivalent in Python is functions that are repetitively called in loops. *Scan-log procedures* scan a range of a spreadsheet and write the data into another range in a structured manner. The program in Fig. 4.5 (P) is a simple example. In Chaps. 8 to 10, log procedures repeat stochastic experiments to illustrate statistical rules and laws.

User-defined functions in visual basic

We generally recommend performing calculations step by step in several rows or columns. If the calculation runs without errors, spreadsheet space and calculation time can be saved if you execute this calculation in a user-defined spreadsheet function, with values transferred from the spreadsheet into the program, and with values written into the spreadsheet by the function, exactly as we learned with built-in spreadsheet functions, e.g., $\cos(x)$ or $\sin(x)$. As examples, we shall implement functions for the scalar and the vector products of three-dimensional vectors, as usual, in both Visual Basic and Python.

Pandas

Pandas (“Python Data Analysis”) is a library for Python, based on NumPy. It is designed for data management and analysis and works with structured data (DataFrame (2-dimensional)) and time series (Series (1-dimensional)), thus

mimicking spreadsheet calculations. We shall use it in this book only in Exercise 4.8 “Processing the protocol of a measuring device”.

4.2 Basic Exercise: FOR-Loops

We get to know the VISUAL-BASIC-Editor and practice reading cell contents and filling in cells. We are using For-loops to execute tasks one after the other with systematically changed parameters. This often involves incrementing a running index in the loop that indicates a cell’s position in the spreadsheet. Such loops are called *loop2i*, obeying the broom rule Ψ *Continue counting in the loop!*

4.2.1 Visual-Basic-Editor 1: Editing

In the menu ribbon (see Sect. 1.7), click on the main group DEVELOPER and then on the tab VISUAL BASIC; the “SHEET1 (CODE)” window appears (Fig. 4.1). If the right lower rectangle (below “(General)”) is gray, double-click the SHEET1 line in the MICROSOFT EXCEL OBJECTS so that it turns white. Instructions that you write into this sheet in the VISUAL BASIC editor are executed in Sheet1.

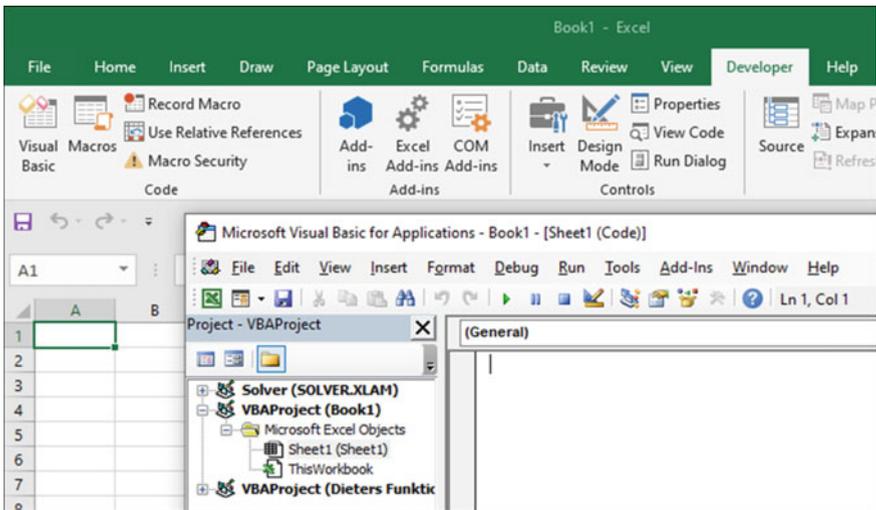


Fig. 4.1 Spreadsheet and associated VISUAL BASIC sheet after executing DEVELOPER/VISUAL BASIC (EXCEL 2019)

4.2.2 Programming

Enter values and formulas into cells

Write SUB *Name* into the first line of the white area! For “Name”, enter your own name. The editor adds one line END SUB. You can now write statements between SUB Name and END SUB. In Fig. 4.2b, SUB *Annegret* has been created with four instructions.

Cell B1 in a spreadsheet can be addressed in two ways, with CELLS(1,2) or with RANGE(“B1”). CELLS(*r*; *c*) addresses the cell in the *r*th row and *c*th column. Cell A1 (= CELLS(1,1)) is filled with a text in quotation marks, here, with “Annegret”. Of course, you should write your own name.

Individual cells can also be addressed with instructions of the type [RANGE(“A1”) = ...], as in lines 3 and 5 of the macro in Fig. 4.2b. In loops, the addressing with CELLS(*r*; *c*) can be used more effectively, if, e.g., the row index *r* or the column index *c* is to be scanned systematically.

► **Task** First, write only this one line [CELLS(1,1) = “Annegret”] into the vba sheet and start the program by pressing the start button ► (high-lighted in Fig. 4.2b). For the procedure to be executed, the pointer | must be somewhere in the procedure. Then, insert the other lines of SUB ANNEGRET one after the other, execute the macro after each line and observe what happens in the spreadsheet:

- a number is written into Cell B1, here, 12.25.
- CELLS(2,1) = CELLS(1,2) means that cell A2 (CELLS(2,1)) is being filled in with the contents of cell B1 (CELLS(1,2)). The content of B1 is transferred once to A2 by the program, and A2 remains unchanged, even if the content of B1 is changed later.
- The text “ = A1” is written into cell B2, interpreted in the spreadsheet as a formula. The corresponding cell is filled in with the contents of cell A1, B2 =

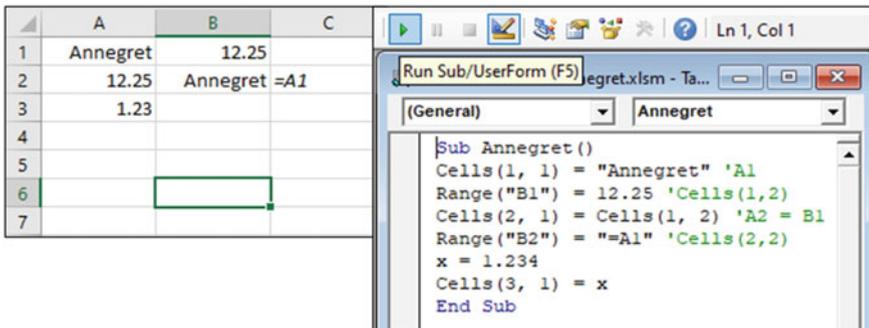


Fig. 4.2 SUB *Annegret* in **b** (right) writes into range A1:B3 in **a** (left). The text after the apostrophe' is interpreted by the VBA interpreter as a comment and not as program code

[=A1]. If the contents of A1 in the spreadsheet are now changed, the new value also appears in cell B2.

► **Task** Delete all entries in the spreadsheet again and then run through the program step by step by placing the cursor | in the program and repeatedly pressing the function key F8 (also obtained and explained in the vba developer, Fig. 4.9, by DEBUG/ STEP INTO). Step by step, the previously deleted entries will appear again in the spreadsheet.

► **FOR loops** The macros in Fig. 4.4 (P), Fig. 4.5 (P), and Fig. 4.6 (P), wherein FOR loops are used, fill in the spreadsheet in Fig. 4.3 (S).

SUB *Protoc1* in Fig. 4.4 (P) fills in column A. Line 2: the text “x” is written into A1. In the FOR loop, the variable *x* is incremented from 3 to 9.5 in steps of 0.25 and written into cells in A. The variable *x* takes on 3 as the first value and is then incremented by 0.25 each time the loop is traversed until the value 9.5 is reached. The variable of the FOR loop, here, *x*, is called the *loop index*. In line 9, a formula is entered into C6, normal text enclosed in quotation marks but starting with an equal sign.

	A	B	C	D	E	F	G	H	I	J	K
1	x		9.50		x	Cos(x)	Sin(x)	Tan(x)			
2	3.00		-1.00 =COS(C1)		3.00	-0.99	0.14	-0.14		3	9.50
3	3.25		-0.08 =SIN(C1)		3.25	-0.99	-0.11	0.11		4	
4	3.50		0.08 =TAN(C1)		3.50	-0.94	-0.35	0.37		5	x
5	3.75				3.75	-0.82	-0.57	0.70		6	Cos(x)
6	4.00		0.99 =COS(C1)^2		4.00	-0.65	-0.76	1.16		7	Sin(x)
7	4.25				4.25	-0.45	-0.89	2.01		8	Tan(x)
8	4.50				4.50	-0.21	-0.98	4.64			
23	8.25				8.25	-0.39	0.92	-2.39		3	0.08
24	8.50				8.50	-0.60	0.80	-1.33		4	=TAN(C1)
25	8.75				8.75	-0.78	0.62	-0.80		5	3.50
26	9.00				9.00	-0.91	0.41	-0.45		6	-0.94
27	9.25				9.25	-0.98	0.17	-0.18		7	-0.35
28	9.50				9.50	-1.00	-0.08	0.08		8	0.37

Fig. 4.3 (S) Column A is filled in by SUB *Protoc1* in Fig. 4.4 (P). The formula in C6 has also been entered by SUB *Protoc1*. The list in columns E:H is obtained from the spreadsheet calculation in C1:C4 by executing SUB *Protoc2* in Fig. 4.5 (P). The range C1:H4 is transferred by SUB *ScanCopy* in Fig. 4.6 (P) into the two columns J and K

1	Sub Protoc1()	r2 = r2 + 1	6
2	Cells(1, 1) = "x"	Next x	7
3	r2 = 2		8
4	For x = 3 To 9.5 Step 0.25	Range("C6") = "=Cos(C1)^2"	9
5	Cells(r2, 1) = x	End Sub	10

Fig. 4.4 (P) SUB *Protoc1* fills in A in Fig. 4.3 (S) and a formula into C6. Syntax for calling a cell: CELLS (ROW, COLUMN)

1	Sub Protoc2()	Cells(r2, 6) = Cells(2, 3)	'6 = column F	7	
2	r2 = 2	'Running index for writing to cells	Cells(r2, 7) = Cells(3, 3)	'7 = column G	8
3	Cells(1, 6) = "Cos(x)"		Cells(r2, 8) = Cells(4, 3)	'8 = column H	9
4	For x = 3 To 9.5 Step 0.25		r2 = r2 + 1		10
5	Cells(1, 3) = x	'Column C	Next x		11
6	Cells(r2, 5) = x	'Column E	End Sub		12

Fig. 4.5 (P) SUB *Protoc2* changes the value in cell C1 (line 5) and writes the function values from C2:C4 consecutively into the columns F (6th) to H (8th) of the spreadsheet in Fig. 4.3 (S)

Loop2i structure

In Fig. 4.4 (P) , we have introduced a *running index* r_2 , which specifies the row of the cell to be filled in. It is set in line 2 to 2 before the start of the loop and is incremented by 1 in line 6 at the end of each loop cycle, so that the values of x are sequentially written into lines 2 to 28. We will often use such structures, call them *Loop2i*, because they comprise two indices, and we memorize them with a broom rule Ψ .

- ▶ Ψ *Loop2i*: Continue counting (the running index) in the loop!

Spreadsheet calculation used as a function

In cells C2:C4 of Fig. 4.3 (S), formulas with the trigonometric functions COS, SIN, and TAN are written by hand with the argument in C1, e.g., C1 = [9.50] and C2 = [=COS(C1)]. These spreadsheet calculations are used by SUB *Protoc2* in Fig. 4.5 (P) as a function. It is a typical *rep-log procedure*, changing a parameter in a spreadsheet calculation and writing the calculation results to another range of the spreadsheet.

SUB *Protoc2* changes the value in cell C1 in the ($x =$)-loop, which is used as an argument in the functions in C2:C4, and transfers the results of the spreadsheet calculation from C2:C4 (one below the other) to F to H (side by side).

The statement in line 3 is: CELLS(1,6) = "Cos(x)". The quotation marks indicate a text in between, which is to be written as text into the cell. An instruction CELLS(1,6) = COS(x) would cause the program first to calculate the cosine of the variable x , to which one would have to have assigned a value somewhere earlier in the procedure, and then write the result, a number, into the cell. An instruction CELLS(1,6) = ' = COS(x)' would write a formula into the cell, similar to line 9 in Fig. 4.4 (P).

All cell references with fixed row and column indices can also be expressed with RANGE, e.g., RANGE("C2") instead of CELLS(2,3).

Questions

concerning SUB *Protoc2*, Fig. 4.5 (P)

1	Sub ScanCopy()	Cells(r2, c2 + 1) = Cells(r, c)	7	
2	r2 = 2	'Row 2	r2 = r2 + 1	8
3	c2 = 10	'Column J	Next c	9
4	For r = 1 To 4	'Rows 1 to 4	r2 = r2 + 1	10
5	For c = 3 To 8	'columns C to H	Next r	11
6	Cells(r2, c2) = c		End Sub	12

Fig. 4.6 (P) SUB *ScanCopy* writes the contents of the range A1:F4 of Fig. 4.3 (S) consecutively into columns J and K of the same spreadsheet. CELLS (2,3) corresponds to C2 in the spreadsheet

What do the instructions in lines 3 and 5 have to be if you want to address with RANGE?¹

Which instructions must be added to the code to write the headings in G1 and H1 of Fig. 4.3 (S)?²

How would you have to change the for-loop in lines 4 to 11 if you want to read the x values from column A?³

Nested loops

SUB *ScanCopy* in Fig. 4.6 (P) transfers the range ($r = 1$ to 4; $c = 3$ to 8), i.e., C1:H4 of the table, to columns J and K of Fig. 4.3 (S).

Range C1:H4 is read horizontally, row by row, and written consecutively vertically into column K ($c_2 + 1 = 11$) with the ($c =$) loop; line 8: The running index r_2 is incremented, indicating the next free row in the spreadsheet.

To be read, the range with the two coordinates row number r (from 1 to 4) and column number c (from 3 to 8) must be scanned with the cells being addressed with CELLS(R,C). This is done with two nested loops, an outer loop (FOR $r =$), and an inner loop (FOR $c =$) that is called within the outer loop and ends in line 9 with NEXT C. SUB *ScanCopy* also writes the index c into column J (line 6, $c_2 = 10$ from line 3).

The line index r_2 is incremented by one at the end of each of the two loops FOR $c =$ and FOR $r =$. The increment in the inner loop (FOR $r =$) causes the adjacent entries in a row of the table, e.g., C1:H1, to be written consecutively into rows 2 to 7 of J, J2:J7. The increment in the outer loop (FOR $c =$) causes a row, e.g., row 8 in the table in Fig. 4.3 (S), to be skipped.

¹ RANGE(„F1“) = “Cos(x)”: RANGE(„C1“) = x .

² RANGE(„G1“) = “Sin(x)”: RANGE(„H1“) = „Tan(x)“; do not forget the quotation marks!

³ FOR $r=2$ TO 28: $x = \text{CELLS}(r, 1)$: ... To put multiple statements on one line in VBA, separate the statements by a colon “:”!

Questions

Using the variables x and r_2 in the *loop2i* in SUB *Protoc2* in Fig. 4.5 (P), explain the broom rule: Ψ *Loop2i: Continue counting (the running index) in the loops!*⁴

It would have been easier to specify the columns in SUB *ScanCopy* in Fig. 4.6 (P) as numbers, i.e. `CELLS(r_2 , 11)` instead of `CELLS(r_2 , $c_2 + 1$)`. Does the variant `CELLS(r_2 , $c_2 + 1$)` offer any advantage?⁵

Why is J8:K8 in Fig. 4.3 (S) not filled in?⁶

What is the value of r_2 at the end of SUB *ScanCopy* in Fig. 4.6 (P)?⁷

How do we proceed further?

In this basic exercise, we have learned how to read content from cells into procedures and fill in cells. We have also become familiar with FOR loops and the special construction of *loop2i* with a loop index and a running index. In this chapter's next exercises, we will get to know sub-routine calls (SUB) and logical queries (IF). FOR, SUB, IF are already the essential basic structures of programming, which we will repeat in Python (`def` instead of SUB) and apply in the following chapters over and over again.

4.3 Macro-Controlled Drawings with FOR, SUB, IF

We construct a macro for drawing filled circles of variable diameter at different coordinates in the spreadsheet. The required instructions are obtained by recording macros generated when a circle is inserted and formatted as a shape by hand. They are combined in a sub-routine to be called from the main program, specifying the circles' position. Similarly, we get the instructions for drawing rectangles and triangles. We are practicing the basic structures of programming: FOR, IF, SUB.

⁴ The rows from $r = 1$ to 4 and the columns from $c = 3$ to 8 are scanned in the nested for-loops. The 24 scanned values are stored in successive rows. The index of these rows, r_2 , must be incremented in the inner FOR loop after every entry.

⁵ If the data is to be output to another range of the spreadsheet, only one parameter for the columns, namely, c_2 , must be adjusted in addition to r_2 .

⁶ Because, in *ScanCopy* in Fig. 4.6 (P), at the end of the loop FOR $r = 1$ to 4, the index r_2 is incremented without data having previously been written into cells in that row.

⁷ At the end of *ScanCopy*, the following applies: $r_2 = 2$ (initial value) + $4 \times 6 = 24$ (c -loop) + 4 (r -loop) = 28 (Row 28 in Fig. 4.3 (S)).

4.3.1 Macro Recorder

We are going to record the commands that are executed when we insert an ellipse into a spreadsheet and format it. The recorded macro is converted into a subroutine that is called several times by the main program with modified coordinates.

In Fig. 4.7a, you see a decorative spiral drawn with the tools acquired in this exercise. The starting point is a macro (Fig. 4.7b), recorded when an ellipse was inserted and formatted.

What do We Learn in This Exercise?

- ▶ **Mag** Once you have completed this task, you can create images like the one in Fig. 4.7a.
- ▶ **Alac** Great, that will amaze my friends!
- ▶ **Mag** More importantly, you will master Visual Basic statements such as those in Fig. 4.7b (P).
- ▶ **Tim** Terribly complicated! I will never be able to keep all of that in my head at the same time.
- ▶ **Mag** You're not supposed to. Figure 4.7b (P) contains a series of instructions that the Macro Recorder has recorded when an ellipse has been inserted by hand.

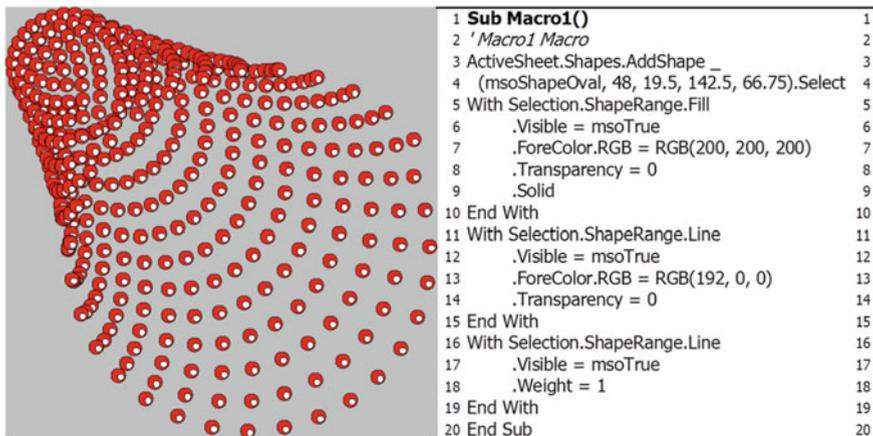


Fig. 4.7 a (left) Decorative spiral, drawn by a macro. b (right, P) Macro recorded by the macro recorder while an ellipse is inserted into the spreadsheet. Superfluous instructions have been deleted. If possible, do not write such code by hand! Get it using DEVELOPER/ RECORD MACRO and modify it as needed!

► **Alac** So, everything is done by the macro recorder?

► **Mag** No, you still have to modify the recorded code, introduce variables and learn the basic program constructions: loops (FOR i = ... TO ...), logical branches (IF THEN ... ELSE ...), and sub-routines. (CALL SUB(a, b, c, ...)).

The tab DEVELOPER/RECORD MACRO

We want to apply a VBA macro to draw a series of filled circles. First, we have to get the elementary commands for drawing a circle. These can be found, in principle, in manuals for VISUAL BASIC FOR APPLICATIONS. Nevertheless, we make life easier for us and use the macro recording function. You can find it in the main register tab DEVELOPER, (see Fig. 4.8). Further explanations can be found in EXCEL help under the keyword CREATE A MACRO.

If the main tab DEVELOPER does not appear in your ribbon, you must activate it in the EXCEL options, with FILE/OPTIONS/CUSTOMIZE RIBBON/MAIN TABS/☑ DEVELOPER.

Circles, Squares, Triangles, by Hand and by Macro

After turning on the macro recording function, we draw a circle by hand (INSERT/ILLUSTRATIONS/SHAPES) and format it. For example, we select the color of the filling and the thickness and color of the border. When we have finished the drawing, we end the macro recording by clicking the STOP RECORDING button, which appears in the toolbar in place of RECORD MACRO. The recorded macro is in a project MODULE (see Fig. 4.9 under “Modules”), not in a VBA sheet connected with a spreadsheet.

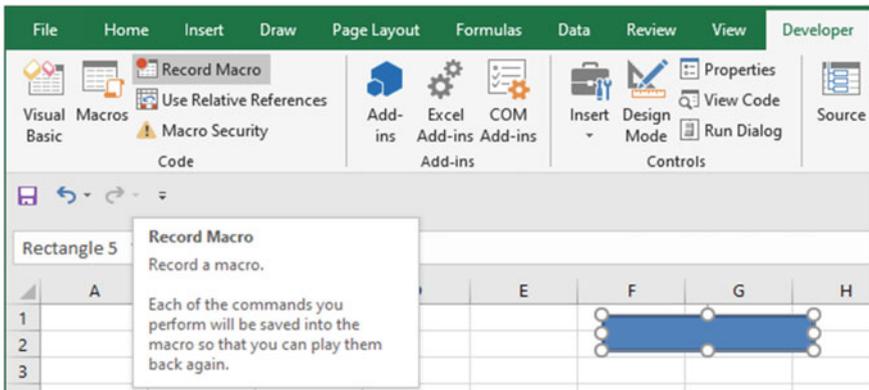


Fig. 4.8 The DEVELOPER/RECORD MACRO tab records all program code associated with the spreadsheet operations performed by the user, e.g., introducing a rectangle as in F1:H2 (INSERT/ILLUSTRATIONS/SHAPES). The VISUAL BASIC button (far left) activates the VISUAL BASIC EDITOR (see Fig. 4.1)

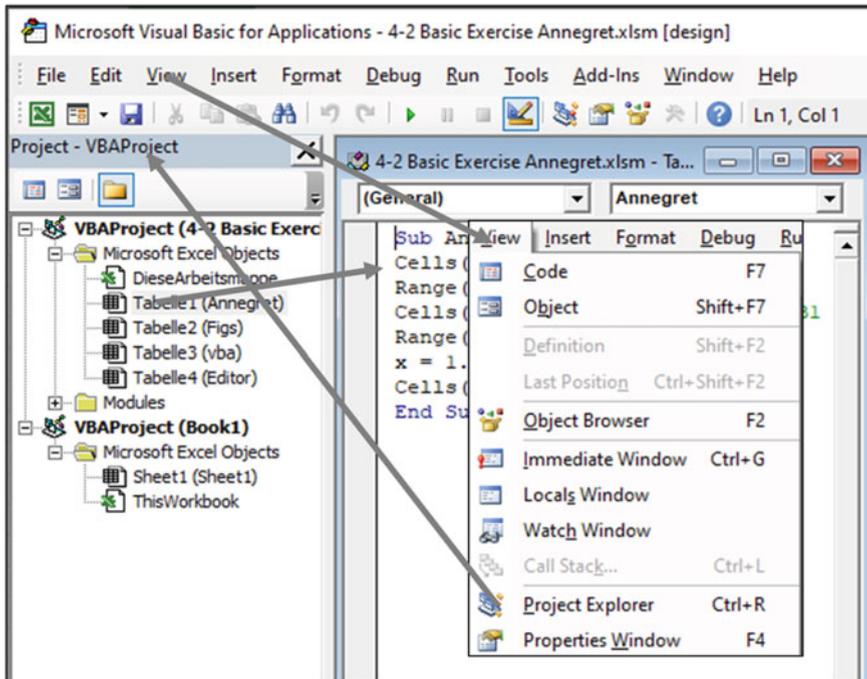


Fig. 4.9 Visual basic editor. You have to activate PROJECT EXPLORER (with VIEW/PROJECT EXPLORER) to see all open files. The recorded macro is located in Module 1 (hidden in the group MODULES) of the VBA project (4–2 Basic Exercise *Annegret*)

VBA is the abbreviation for “Visual Basic for Applications”. The addition “for applications” indicates that the application’s instructions, here, EXCEL, are available as internal instructions, e.g., `ACTIVESHEET.ADDSHAPE`, with which a geometric form is inserted into the spreadsheet.

Select Objects and Edit them Together

You can select objects with the arrow cursor. To do this, click HOME/FIND AND SELECT/\SELECT OBJECTS at the ribbon’s far-right. With the new mouse pointer held down, you can now span a rectangle within which all drawing objects are selected and edit this set of objects as a whole, for example, color them, group them, or delete them.

Within a VBA macro, use the command `ACTIVESHEET.DRAWINGOBJECTS.SELECT` to select all drawing objects in the active worksheet. With `SELECTION.DELETE`, you can delete all objects.

4.3.2 Visual-Basic Editor 2: Macro Recording, Debugging

We can review the result of our macro recording in the Visual Basic editor. This editor is activated when you click on DEVELOPER/VISUAL BASIC (far left in Fig. 4.1) or press ALT F11. A window like that in Fig. 4.9 appears when the program page of a sheet or a module is additionally double-clicked. SHEET1 has been clicked here, which already contains SUB *Annegret* from Sect. 4.3.2.

Upon clicking on the “View” tab, a menu opens up that has been placed over SUB *Annegret* in Fig. 4.9.

We click on the PROJECT EXPLORER button, and the PROJECT—VBAPROJECT sub-window appears. In this window, each worksheet (SHEET1, SHEET2, SHEET3, SHEET4) is assigned a VBA sheet in which Visual Basic code can be generated and edited. In Fig. 4.9, SHEET1 (Tabelle1(Annegret)) has been clicked, and in the editor, the macro SUB ANNEGRET from Sect. 4.2.2 has popped up.

Since we have already recorded a macro, another object MODULE1 appears under Modules . It contains the program code SUB MACRO1, which we have transferred to Fig. 4.7b (P), with four instructions:

- Lines 3 and 4: An ellipse (MSOSHAPEOVAL) has been created. The first two numbers in the argument list are the x and y coordinates; both are measured from the upper left corner of the spreadsheet. The next two numbers in the list are the two diameters of the ellipse.
- Line 7: The area within the ellipse is colored.
- Line 13: The border of the ellipse is colored.
- Line 18: The thickness of the border of the ellipse is specified.

You can edit the macro commands in the editor like normal text. The syntax must, of course, comply with the rules of the VBA interpreter.

Debug/Step Into

Let the macro run again; best if you do it step by step! If you place the cursor in a program in the Visual Basic Editor and press the function key F8, each step of the program is executed individually (DEBUG/STEP INTO). You can then see exactly what is happening and check whether the drawing is changing as you expect. You can also change the instructions before they are executed. Going through a macro step by step is a good way to detect programming errors.

If you place the cursor on a variable name, the value of that variable will pop up.

► **Task** Change the coordinates and the size of the diameters by modifying the instructions!

► **Mag** Now, the real programming starts, with loops and sub-routines!

4.3.3 Programming Elements

Variables Instead of Numbers

In SUB MACRO2 in Fig. 4.10 (P), we have replaced the current numbers in ADDSHAPE with variables x , y , dx and dy , to which we have assigned values in lines 2 to 5. If we run this macro, one of the shapes in Fig. 4.11a (S) is created, or a similar one if other values have been chosen.

The colors in lines 9, 12, 32, and 35 are composed of red, green, and blue components (intensity between 0 and 255) via the specification RGB (red, green, blue).

<pre> 1 Sub Macro2() 2 x = 400 3 y = 20 4 dx = 100 5 dy = 50 6 ActiveSheet.Shapes.AddShape(msoShapeOval, _ 7 x, y, dx, dy).Select 8 With Selection.ShapeRange.Fill 9 .ForeColor.RGB = RGB(220, 220, 220) 10 End With 11 With Selection.ShapeRange.Line 12 .ForeColor.RGB = RGB(180, 0, 0) 13 .Weight = 1 14 End With 15 End Sub 16 17 Private Sub CommandButton1_Click() 18 Call Circles 19 End Sub </pre>	<pre> Sub Circles() For i = 1 To 3 Call Disc(i * 50 + 60, i * 25) Next i End Sub Sub Disc(x, y) 'x and y are the coordinates of the center d = 50 'diameter of the circle ActiveSheet.Shapes.AddShape(msoShapeOval, _ x - d / 2, y - d / 2, d, d).Select With Selection.ShapeRange.Fill .ForeColor.RGB = RGB(220, 220, 220) End With With Selection.ShapeRange.Line .ForeColor.RGB = RGB(180, 0, 0) .Weight = 1 End With End Sub </pre>	<pre> 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 </pre>
--	--	---

Fig. 4.10 a (left, P) Variable names are introduced, MACRO1() from Fig. 4.7b (P) becomes Macro2(). The macro SUB CommandButton1 is triggered by the command button in Fig. 4.11b. **b** (right, P) Macro2() is converted into a sub-routine Disc(x,y), which is called repeatedly by the main program Circles with various values for x and y , with the result in Fig. 4.11b

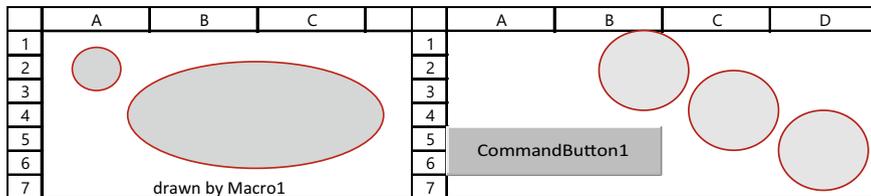


Fig. 4.11 a (left, S) Circle and ellipse after executing SUB Macro2 in Fig. 4.10a (P). **b** (right, S) Result of the procedure Circles in Fig. 4.10b (P)

Questions

Which color is created with `RGB(180, 0, 0)`?⁸

Which color is created with `RGB(220,220,220)`?⁹

How can you tell that a circular disc, and not an elongated ellipse, is produced with `SUB Disc`?¹⁰

Sub-routines

We want to summarize the relevant instructions in a sub-routine “Disc”, which contains, in the procedure header, the coordinates (x, y) of the center of the circle in the parameter list, `Disc(x, y)`. This sub-routine is called from a main program with different values for (x, y) . Figure 4.10b (P) suggests a solution for this task, with the main program `SUB Circles` and the sub-routine `SUB Disc`. Superfluous specifications in the recorded macro have been deleted.

The diameter of the circular disc is set to $d = 50$ in `SUB Disc` (line 28). Grey is now selected as the fill color, lines 31, 32. The line width remains as before (... `LINE.WEIGHT = 1`). These parameters cannot be changed by the main program, because they are not in the procedure header.

When placing the circular disc in the spreadsheet, note that the center of the circle is passed via the procedure header (`SUB Disc(x,y)`), but that it is the upper left corner of the shape that must be specified in the drawing command.

The ratio of the scaling in Visual Basic to the grid scale in the spreadsheet can be seen from the following data:

- A circle with diameter 100 points has a diameter of 3.53 cm.
- 28.4 point correspond to 1 cm.
- 28.5; 28.6; 28.7 point all correspond to 1.01 cm.
- 28,8 point correspond to 1.02 cm.

Transferring parameters to sub-routines

- ▶ When parameters are passed to sub-routines, the order in the argument list in the procedure header is decisive; the names in the main program are not significant.

A procedure header in Fig. 4.10b (P) reads `SUB Disc(x,y)`. This sub-routine is called in `SUB Circles()` with `CALL Disc(I*50 + 60, I*25)`. The first entry in the header in `SUB Disc` is taken over as x , and the second entry as y . We often name the variables

⁸ `RGB(180, 0, 0)` is a strong red, intensity 180 of 255.

⁹ `RGB(220,220,220)` is a light grey; red, green and blue are equally present.

¹⁰ In lines 29, 30, `ACTIVESHEET.SHAPES.ADDSHAPE(..., d, d)`, the same variable d is used for both diagonals of the ellipse.

in the main program the same as in the sub-routine. So, we could write $x = i*50 + 60$ and $y = i*25$ within the loop in `SUB Circles()` and then call `CALL Disc(x, y)`, with the same result as above.

If we executed `CALL Disc(y, x)`, the first entry, here, y from the main program, would be interpreted as x in the subprogram and the second entry as y . The row of the three circular discs would start at A8 and go down more steeply. In the main program, we could also choose completely different variable names, e.g., a and b , and then proceed with `CALL Disc(a, b)` or `CALL Disc(b, a)`.

- ▶ Name the variables such that you are best able to keep an overview!

Questions

concerning Fig. 4.10b (P)

Which three circle centers are passed to `SUB Disc(x,y)` in `SUB Circles`?¹¹

Which argument in `CELLS(a,b)` stands for the row index in the spreadsheet?¹²

Main Program

A main program is characterized by the fact that it contains no parameters in the procedure header. Only main programs are executable programs. Sub-routines generally contain parameters in the header that must be assigned values by a higher-level program. Examples:

- `SUB CIRCLES()` in Fig. 4.10b (P) is a main program that the user can start.
- `SUB DISC(X,Y)` in Fig. 4.10b (P) is a sub-routine with x and y in the procedure header. It cannot run on its own, but can only be called by another procedure with specified values for the parameters x and y .

▶ **Task** Change the procedure so that, in addition to the coordinates of the center point, the diameter d of the circle and the thickness w of the boundary (SHAPE OUTLINE) are selected in the main program and are transferred to the sub-routine as parameters in the procedure header!¹³

FOR loop

The main program `Circles` calls the sub-routine `DISC` in the loop (`FOR i =`) three times. The centres of the circular disks are set to (110, 25), (160, 50) and (210,75) for $i = 1, 2, 3$. The drawing resulting from these specifications is shown in Fig. 4.11b.

¹¹ $(x, y) = (110, 25), (160, 50)$ and $(210, 75)$.

¹² The first argument, a , stands for the row: `CELLS(row, columns)`.

¹³ `SUB DISC(X,Y,D,W), LINE 36, WEIGHT = W.`

A FOR loop is used in the macro *Circles*. The general syntax for a FOR loop is:

```
FOR x = xmin TO xmax STEP delta_x
  {LIST OF COMMANDS}
NEXT x
```

An example with integers:

```
r2 = 10
FOR N = -211 TO 453 STEP 12
  CELLS(r2, 2)=N
  R2 = R2 + 1
NEXT N
```

When this loop is executed, the loop index n assumes the values $-211, -199, \dots, 437, 449$. CELLS(10,2) to CELLS(65,2) are filled in. In the argument of CELLS, the row number comes first and the column number second. Cells B10 to B65 are therefore filled in with $-211, -199, \dots, 449$.

A further example is the loop in SUB *Circles()*, in which the sub-routine *Disc* is called three times:

```
FOR I = 1 TO 3
  CALL Disc(I*50+60, I*25)
NEXT I
```

► **Task** Develop a macro for drawing a row of rectangles! “Develop” means that you get the instructions with RECORD MACRO and redesign the recorded macro using variables, sub-routines, and loops.

► **Task** Write a macro that draws a (4×4) array of filled circles, the colors thereof being composed of fractions of red and green, with the green fraction systematically increasing in each row and the red fraction systematically increasing in each column!

Command Button, Design Mode On/Off

In Fig. 4.11b, a command button has been inserted in “Design mode” in A5:B6 with DEVELOPER/INSERT/ACTIVEX CONTROLS/COMMAND BUTTON (visible when the mouse is over the  icon).

In the PROPERTIES card, revealed by right-hand clicking on COMMAND BUTTON/PROPERTIES, COMMANDBUTTON1 (as text) is assigned both as a NAME and a CAPTION for the command button. As is usually the case with controls, with the DESIGN MODE turned on (click the DESIGN MODE button on the DEVELOPER tab, see Fig. 4.1 and Fig. 1.1 of Sect. 1.7), the PROPERTIES can be changed, e.g., name and caption. When the design mode is switched off (by clicking the DESIGN MODE button again), the control can be operated.

The procedure `SUB COMMANDBUTTON1_CLICK` in Fig. 4.10a (P) is assigned to the command button. In detail, proceed as follows: In the VISUAL BASIC Editor (Fig. 4.9), click on the arrow ∇ at (GENERAL). A list opens up in which `SUB COMMANDBUTTON1` appears. This entry is activated by clicking on it. Next, click on the arrow \blacktriangledown next to the cell with the inscription “Annegret” (as shown in Fig. 4.9, or the name you have chosen). A list opens up in which `CLICK` appears, together with other commands. Click on this entry, and `SUB COMMANDBUTTON1` is immediately completed to `SUB COMMANDBUTTON1_CLICK`. The upper line in the VBA editor will now read `COMMANDBUTTON1 \blacktriangledown : CLICK()`.

In our case, in Fig. 4.10a (P), only one procedure, `SUB Circles`, is called. We could omit `SUB COMMANDBUTTON1_CLICK` by naming the command button `Circles` and completing `SUB Circles()` to `SUB Circles_CLICK()`. Please note that we would have to change the *name* of the button, which is independent of its *caption*.

4.4 A Checkerboard Pattern (Excel)

We obtain the VBA commands for drawing elementary geometric shapes by recording macros and incorporate them into a sub-routine that executes the drawing in the desired layout. The shapes' position in the spreadsheet is passed to the procedure via its header or via global variables. The respective shapes, as well as the color of their borders and interiors, are selected randomly.

4.4.1 Checkerboard, Same-Colored and Multi-colored

Checkerboard with same-colored shapes

In this exercise, a checkerboard of rectangles, triangles, and circles, as shown in Fig. 4.12a, is to be drawn. The procedure for this is shown in Fig. 4.13 (P).

`SUB DRAW11` in Fig. 4.13 (P) is the main program that randomly calls one of the sub-routines `Rect`, `Ova`, or `Tria`, ten times in each of eight rows, drawing a rectangle, ellipse, or triangle at the current position of x and y . Its core is a nested loop with two loop indices, k for the row and i for the column address within a row.

The variable `ROT` in `SUB draw11` determines whether a rectangle, an ellipse (oval) or a triangle shall be drawn. In line 5, the variable `ROT` is randomly assigned a value 0, 1, or 2. Chance is brought in by the function `RND()` generating a random number between 0 and 1, which is then multiplied by 3. This real number is turned into an integer by `INT` (into the variable `ROT`). To give some examples: `INT(0.75*3) = INT(2.25) = 2`; `INT(0.22*3) = INT(0.66) = 0`; `INT(0.54*3) = INT(1.53) = 1`.

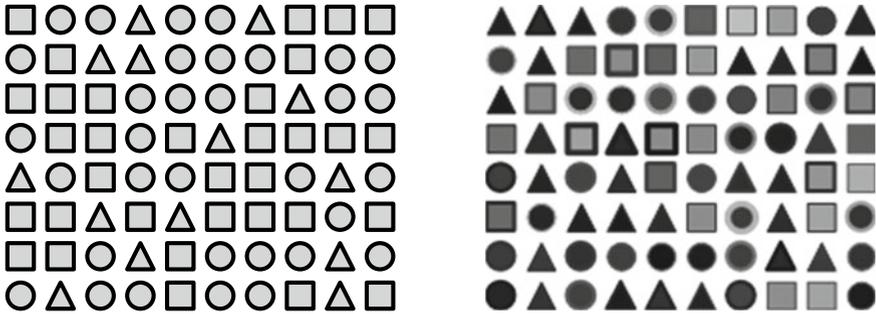


Fig. 4.12 a (left) A checkerboard pattern of rectangles, circles and triangles, all equally formatted, drawn with *SUB drawi1* in Fig. 4.13 (P). b (right) Like a, but with forms differently formatted, filled with different colors, and surrounded with borders of different thickness and different color, drawn with *SUB drawi* in Fig. 4.14 (P)

1	Sub drawi1()	If ROT = 2 Then Call Tria(x, y)	8
2	x = 100	x = x + 15	9
3	For k = 1 To 8 'next row	Next i	10
4	For i = 1 To 10 'within row	x = 100 'reset left position	11
5	ROT = Int(Rnd() * 3) '0, 1 or 2	y = y + 15 'advance top position	12
6	If ROT = 0 Then Call Rect(x, y)	Next k	13
7	If ROT = 1 Then Call Ova(x, y)	End Sub	14

Fig. 4.13 (P) Procedure *SUB drawi1* with which Fig. 4.12a is drawn

In lines 6 to 8, logical IF queries determine which shape is drawn. After the shape has been drawn, the x value is increased by 15 (line 9).

► **Task** First, draw only one row by omitting the loop (FOR k = ...)! The sub-routines *Rect(x,y)*, *Ova(x,y)* and *Tri(x,y)* should be written according to the model of *SUB Disc(x,y)* in Fig. 4.10b (P). Apart from MSOSHAPEOVAL, MSOSHAPERECTANGLE and MSOSHAPETRIANGLE have to be used.

► **Task** Draw the complete checkerboard pattern!

A randomly more colored checkerboard pattern

We draw 8 rows with 10 shapes each, such as shown in Fig. 4.12b. The format of the shapes, namely, the color to be filled in and the color and thickness of the border, is now determined using a random number. In addition, the positions (left, top) of the shapes are not passed to the sub-routines through the procedure header, but via global variables. The main procedure can be found in Fig. 4.14 (P), a typical sub-routine in Fig. 4.15 (P).

1	Private x, y As Single	If ROT = 0 Then Call Rect	10
2	'Position of the shape to be currently drawn	If ROT = 1 Then Call Ova	11
3		If ROT = 2 Then Call Tria	12
4	Sub drawi()	Next i	13
5	x = 100 'left position	x = 100 'reset left position	14
6	y = 100 'top position	y = y + 15 'advance top position	15
7	For k = 1 To 8 'next row	Next k	16
8	For i = 1 To 10 'within row	End Sub	17
9	ROT = Int(Rnd() * 3) '0, 1 or 2		18

Fig. 4.14 (P) SUB *drawi* is the main program calling, 10 times in each of 8 rows, one of the sub-routines *Rect*, *Ova*, or *Tria*, which draw a rectangle, ellipse, or triangle at the current position of x and y . Similar to Fig. 4.13 (P), but with parameters stored in global variables x , y defined in line 1

1	Sub Rect() 'Draws a rectangle of width 10 and height 10	1
2	ActiveSheet.Shapes.AddShape(msoShapeRectangle, x, y, 10, 10).Select	2
3	Call Lin(255 * Rnd(), 63 * Rnd(), 63 * Rnd(), 10) 'Rim of shape	3
4	'Formats rim: Red fully varied, green and blue, half intensity, strength	4
5	Call Interi(63 * Rnd(), 128 + 127 * Rnd(), 63 * Rnd()) 'Interior of shape	5
6	'Formats interior: Green always more intense than 50%	6
7	x = x + 15 'Advances position in the row	7
8	End Sub	8

Fig. 4.15 (P) SUB *Rect* draws a rectangle with a fixed size, but with randomly selected colors for the border (line 3) and interior (line 5). The instructions for coloring are executed in the *Lin* and *Interi* sub-routines in Fig. 4.16 (P)

4.4.2 Global Variables

Global variables are also valid in sub-routines. They must be declared as `PRIVATE` or `PUBLIC` before the routines (see line 1 in Fig. 4.14 (P)). Variables of type `PRIVATE` are only available in the module in which they are declared, those of type `PUBLIC` in the whole workbook. The positions (Left, Top) of the shapes are now stored in global variables x and y that can be read and modified by each sub-routine.

The data type `SINGLE` in line 1 of Fig. 4.14 (P) denotes a single-precision floating-point number stored in 4 bytes. Decimal numbers of the data type `DOUBLE` are stored in 8 bytes. You can find out more about other data types with EXCEL help in the VBA-Editor.

- ▶ Don't just copy the macros if you already have some programming practice! Rehearse the sequence of instructions in your mind and get the commands for drawing shapes through macro recording!

In SUB *Rect* in Fig. 4.15 (P), a square is drawn, and its interior and border are formatted with the sub-routines *Lin* and *Interi* in Fig. 4.16 (P). The position is taken from the global variables (x , y) and passed to `MSOSHAPERECTANGLE` through the procedure header of *AddShape*. Both side lengths are fixed to 10. The arguments

1 Sub Lin(r, g, b, w)	Sub Interi(r, g, b)	8
2 <i>red, green, blue and weight w</i>	<i>'red, green, blue</i>	9
3 With Selection.ShapeRange.Line	With Selection.ShapeRange.Fill	10
4 .ForeColor.RGB = RGB(r, g, b)	.ForeColor.RGB = RGB(r, g, b)	11
5 .Weight = w	End With	12
6 End With	End Sub	13
7 End Sub		14

Fig. 4.16 (P) SUB *Lin* and SUB *Interi* color the border and the interior of the shape (line 2), respectively, according to the variables *r* (red), *g* (green), and *b* (blue). The thickness (weight) of the border is specified in *w*, set to 10 in Fig. 4.15 (P)

in the headers of *Lin* and *Interi* are generated in SUB *Rect(...)* with RND() which returns a random number between 0 and 1.

Questions

- concerning Fig. 4.15 (P):
 - The procedure header of SUB *Rect* is empty. How does the sub-routine know the position in the spreadsheet where the rectangle is to be drawn?¹⁴
 - Which color dominates the border of the shapes?¹⁵

4.5 A Checkerboard Pattern (Python)

We draw a multi-colored checkerboard pattern using the Python library *turtle*, setting up a similar program structure as for EXCEL in Sect. 4.4, but considering the differences in code from Visual Basic.

4.5.1 Turtle

In order to draw a set of shapes with Python, we use the library *turtle*. This simple plot program’s illustrative idea is that of turtles running across the screen, thereby creating colorful traces.

In Table 4.1, the libraries *turtle* and *numpy.random* are imported and a turtle named *t* is created. This will be the first cell in every program; it must be run before the functions are compiled, because they resort to these libraries. Generally, several turtles can be active at the same time. We use, however, only

¹⁴ SUB *Rect* accesses the global variables *x*, *y*.
¹⁵ Line 5 in Fig. 4.15 (P): Green = 128 + 127*RND() is represented at least with strength 128.

Table 4.1 Importing relevant libraries; creating a screen with a title; the internet address in line 1 points to an introduction to Turtle graphics

```

1 #https://docs.python.org/3.3/library/turtle.html
2 import turtle
3 import numpy.random as npr
4 t=turtle.Turtle() #Creates a turtle with name t
5 turtle.title("Checkerboard")

```

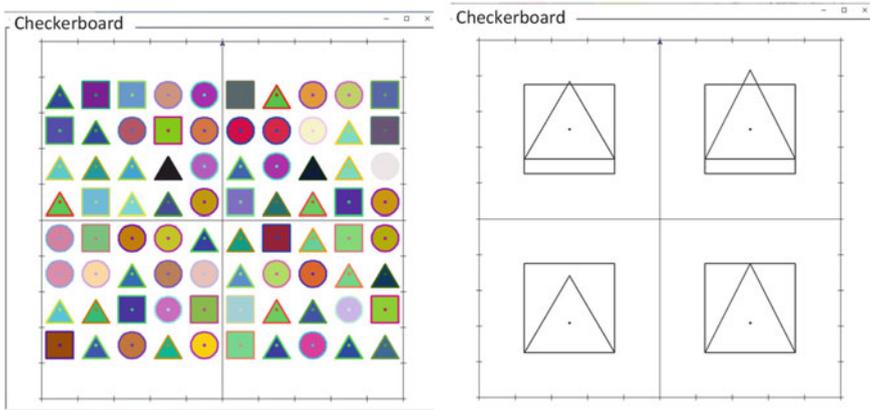


Fig. 4.17 a (left) Turtle screen created by the Python program in this section, in a frame spanned by the points $(-400, -400)$ and $(400, 400)$. b (right) Triangles of different size and at different positions in a square

one turtle instance called t . In the second cell in Table 4.1, a screen is created with a program-specific name, here, *Checkerboard*.

The standard size of the screen is $1000 \text{ pt} \times 800 \text{ pt}$ spanned between $(-500, -400)$ to $(500, 400)$. Such a screen with a checkerboard pattern is shown in Fig. 4.17a. Turtle and EXCEL apply coordinate systems with different origins. In EXCEL, it is at the upper left corner of the spreadsheet, and all coordinates are positive. In Turtle, the origin is at the center of the screen, and the coordinates are positive or negative.

Attention: The turtle window may be below the Python window.

Different types of triangle

The centroid (x_C, y_C) of a triangle, corresponding to its center of gravity, is calculated as

$$x_C = \frac{x_1 + x_2 + x_3}{3}, y_C = \frac{y_1 + y_2 + y_3}{3}$$

It is marked in Fig. 4.17b with dots.

The function *drawTria*, used to draw Fig. 4.17a, draws an equilateral triangle (bottom left in Fig. 4.17b) whose centroid does not coincide with the square's center,

in contrast to the equilateral triangle at top left. The bottom right triangle has its top point at the midpoint of the upper side of the square. It is shifted in the upper right figure so that its centroid coincides with the square's center.

Basic functions and measurements

The basic functions (always completed with parentheses `()`), attributed to the instance of a turtle, e.g., `t.penup()` or `t.pos()`, are.

Settings:

<code>.pen(...)</code>	specifying speed, pensize, pencolor, fillcolor
<code>.pu()</code>	pen up
<code>.pd()</code>	pen down
<code>.setpos(x, y)</code>	moving to the specified position
<code>.setheading(ϕ)</code>	setting the orientation as angle ϕ in $^\circ$ with respect to the x -axis
<code>.rt(ϕ)</code>	right turn by angle ϕ in $^\circ$
<code>.lt(ϕ)</code>	left turn by angle ϕ in $^\circ$
<code>.fd(r)</code>	step forward by r pixels
<code>.bk(r)</code>	step backward by r pixels
<code>.dot(s, c)</code>	plots a dot with diameter s and color c at current position
<code>.begin_fill()</code>	beginning to fill in the contour
<code>.end_fill()</code>	ending to fill in the contour

Measurements:

<code>.pos()</code>	returns Cartesian coordinates of turtle
<code>.heading()</code>	returns angle of direction
<code>.distance(x, y)</code>	returns distance to point (x, y)

Question

How do you get the position of a turtle named *doro* in polar coordinates? There are two possibilities.¹⁶

4.5.2 Differences to Visual Basic

We will demonstrate some syntactic differences between Python and VISUAL BASIC by means of the function *drawSquare*, which creates a square shape.

¹⁶ `r = doro.distance(0, 0)`, `phi = doro.heading()/180*np.pi()` or `x = doro.pos(0)`, `y = doro.pos(1)`, `r = np.sqrt(x**2 + y**2)`, `phi = np.arctan2(y, x)`

Table 4.2 Function *drawSquare* in Python, drawing a square with center (x, y) and size *sz* that is filled if the keyword variable *fill* is set to True

```

1 def drawSquare(x,y,sz,fill=False):
2     r=sz/2
3     t.pu()                               #Pen up!
4     t.setpos(x-r,y-r)                   #Run to position!
5     t.pd()                               #Pen down!
6     if fill==True: t.begin_fill()
7     for i in range(4):
8         t.fd(sz)                         #Forward!
9         t.lt(90)                         #Turn left!
10    if fill==True: t.end_fill() #Fill square with color!
11    t.pu()
12    t.setpos(x,y)

```

In Table 4.2, it is implemented in Python with four segments of equal length and a 90° turn after each segment; the features of the pen have been specified in a superordinate program. In Fig. 4.18 (P), this is realized in a Visual Basic sub-routine that is called from the main program SUB DSQ().

Positional and keyword arguments

The procedure headers in both cases, Python and VBA, contain positional arguments x , y , sz , and a keyword argument *fill* that decides whether or not the contour is filled (with the default set to False).

The first three positions in the header must contain appropriate values when the function is called. It is the position in the header that determines the variable in the procedure to which the value is assigned. In Fig. 4.18 (P), a variable y with value 40 is passed through the third position to the procedure, where it is assigned to a variable named sz .

The keyword argument *fill* is defined in the procedure header with a default value taken in the procedure, if not specified otherwise.

1 Dim col(2) As Integer	Sub drawSquare(x, y, sz, _	13
2	Optional fill As Boolean = False)	14
3 Sub dSq()	ActiveSheet.Shapes.AddShape(_	15
4 col(0) = 100	msoShapeRectangle, x, y, sz, sz).Select	16
5 col(1) = 200	Selection.ShapeRange.fill.Visible = msoFalse	17
6 col(2) = 0	If fill = True Then	18
7 y = 40	With Selection.ShapeRange.fill	19
8 For a = 100 To 200 Step 50	.Visible = msoTrue	20
9 Call drawSquare(a, 100, y, fill:=True)	.ForeColor.rgb = rgb(col(0), col(1), col(2))	21
10 Next a	End With	22
11 Call drawSquare(100, 100, y, fill:=True)	End If	23
12 End Sub	End Sub	24

Fig. 4.18 (P) *DrawSquare* realized in visual basic for EXCEL

Global parameters

The first line in the VISUAL BASIC program in Fig. 4.18, before the procedures, defines an array *col* of integers with three elements accessible in the whole module. It is written in the main program on lines 4 to 6 and read in *drawSquare* on line 21.

In Python, arrays can be declared anywhere in the program, e.g., by `col = [100, 200, 0]`. All variables, as well as arrays, are valid in subordinate functions unless the variable name is again declared in a function with an equal sign, e.g., `col = [10, 100, 10]`, creating a new object with own memory space.

Grouping blocks of code

In Python, indentation has a syntactic function. A block of code is necessarily grouped by the same amount of indentation. In Table 4.2, there are two examples: the statements of the function are all indented by 4 spaces; the two statements to be executed in the for-loop are further indented by another 4 spaces.

EXCEL uses code words to state the end of a block. In Fig. 4.18 (P), the procedure (sub-routine) ends with `END SUB`, the IF block with `END IF`, and the WITH block with `END WITH`. The (FOR A =) block in the main program ends with `NEXT A`. Although indentation does not have a syntactic function in Visual Basic, we use it to maintain a better overview of the program structure.

Case-sensitivity

Python is case-sensitive: `True` and `False` have to be written with capital T and F; `x` and `X` are two different variables. Visual Basic is case-insensitive: an input “true” is automatically changed to “TRUE”; `x` and `X` are regarded as the same variable. When we change the case of the first letter anywhere in the program, names in other places will automatically adapt.

4.5.3 Checkerboard with Squares, Triangles, and Circles

User-defined functions for square, triangle, circle, dash

To draw a checkerboard pattern as in Fig. 4.17a, we need functions that draw, besides the rectangle already realized in Table 4.2, a triangle and a circle. They should have the same procedure header as *drawSquare*, with the center point (x,y) and the size *sz* as positional arguments and *fill* as a keyword argument.

Drawing a triangle in Table 4.3 is similar to how it is done in *drawSquare*. Drawing a circle with radius *r* is achieved with the built-in function `circle(r)` (see Table 4.4). There, the turtle runs along a circle of radius *r*, starting at its current position and with its current direction.

In the three functions mentioned above, the turtle runs along a shape, starting at its current position and with its current direction. It was, however, intended by the programmer that the turtle start running straight to the right. This is indeed

Table 4.3 Drawing a triangle with center (x, y)

```

13 def drawTria(x, y, sz, fill=False):
14     r=sz/2
15     t.pu()
16     t.setpos(x-r, y-r)           #Left lower edge
17     t.pd()
18     if fill==True: t.begin_fill()
19     for i in range(3):
20         t.fd(sz)                 #Forward!
21         t.lt(120)                #Turn left!
22     if fill==True: t.end_fill()
23     t.pu()
24     t.setpos(x, y)

```

Table 4.4 Drawing a circle with center (x, y)

```

25 def drawCircle(x, y, sz, fill=False):
26     r=sz/2
27     t.penup()
28     t.setpos(x, y-r)
29     t.pendown()
30     if fill==True: t.begin_fill()
31     t.circle(r)                 #Is a function within turtle
32     if fill==True: t.end_fill()
33     t.penup()
34     t.setpos(x, y)

```

assured in our current main program, but nevertheless, not making this intention explicit is considered a big mistake in Software Engineering.

Questions

The turtle named t in Tables 4.2, 4.3, and 4.4 is a global instance accessed within the functions. This is possible because you have only one turtle running. What do you do if several turtles are on the field?¹⁷

In Table 4.4, it is implicitly assumed by the programmer that the turtle is heading straight to the right at start, a big programming mistake. How do you avoid this bug?¹⁸

¹⁷ The turtle name has to be an argument, e.g. `t, def drawSquare(t, x, y, sz, fill = false)`, so that it is no longer regarded as a global instance in the functions.

¹⁸ Introduce `t.setheading(0)` before the turtle starts running. If necessary, store the original direction at the beginning, e.g., `phi0 = t.heading()` and reset it at the end with `t.setheading(phi0)`.

Table 4.5 Drawing a dash

```

35 def dash(ds):
36     t.rt(90)      #Right turn
37     t.fd(ds)
38     t.rt(180)   #180° to the right
39     t.fd(2*ds)
40     t.rt(180)
41     t.fd(ds)    #Back to zero
42     t.lt(90)    #90° to the left, original direction

```

Table 4.6 Drawing the checkerboard pattern with a nested loop; when lines 3 and 20 are activated, the turtle runs faster

```

1  turtle.clearscreen()
2  t=turtle.Turtle()
3  #turtle.tracer(0, 0)
4  #Draw checkerboard pattern
5  for rn in range(-280,285,80):
6      for c in range(-360,365,80):
7          r=npr.rand()
8          g=npr.rand()
9          b=npr.rand()
10         tup=(r,g,b)    #Red, green, blue
11         tup2=(g,b,r)  #g=Red, b=green, r=blue
12         t.pen(pencolor=tup2, fillcolor=tup,
13              pensize=4, speed=0)
14         if r<0.33:
15             drawTria(c,rn,60,fill=True)
16         elif r<0.67:
17             drawSquare(c,rn,60,fill=True)
18         else: drawCircle(c,rn,60,fill=True)
19         t.dot()
20 #turtle.update()

```

The function *dash* in Table 4.5 draws a dash at the current position perpendicular to the current turtle heading, of extension *ds* to both sides.

Main program

The main program has three parts. The first part is shown in Table 4.1, importing the necessary libraries and creating a screen with the title *Checkerboard*. Then, in Table 4.6, the checkerboard pattern is drawn with a nested loop over 10*x* positions and 8 *y* positions, with randomly choosing one of our three shapes.

Before we call the *draw** functions, the pen specifications have to be set in the main program, *pencolor* and *fillcolor* in our program, by *rgb* (red, green, blue) in standard mode with values between 0 and 1, randomly chosen with *npr.rand()*.

Table 4.7 Drawing a frame with dashes around the checkerboard

```

21 #Draw frame with dashes
22 t.pen(pencolor="black", pensize=1, speed=10)
23 t.penup()
24 t.setpos(-400,-400)      #Left bottom corner
25 t.pendown()
26 for k in range(4):      #4 straight lines
27     for i in range(10): #10 segments with dashes
28         dash(6)
29         t.fd(80)        #Forward!
30     dash(6)
31     t.lt(90)           #Left turn 90°!
32 #turtle.update()

```

Table 4.8 Drawing the axes of the coordinate system

```

33 #Axes of the coordinate system
34 t.pu(), t.setpos(-400,0), t.pd()
35 t.fd(800)                #Forward!
36 t.pu(), t.setpos(0,-400), t.pd()
37 t.setheading(90)        #Direction 90° to x axis
38 t.fd(800)
39 #turtle.update()

```

The keyword variable `pencolor` expects a tuple with 3 elements to specify the color. The individual variables *r* (red), *g* (green), *b* (blue) are set in lines 7 to 9 and assembled into two different tuples, *tup* for `fillcolor` and *tup2* for `pencolor`.

Frame around the figure in Turtle

The main program is continued in Tables 4.7 and 4.8, through drawing of a frame around the checkerboard and the axes of the coordinate system, respectively.

Questions

What type of triangle is specified in *drawTria*, Table 4.3: equilateral, or acute-angled?¹⁹

How do you draw a triangle touching a square at two neighboring corners and the center of the opposite side (see Fig. 4.17b, bottom right)?²⁰

¹⁹ The function *drawTriangle* draws a triangle with all angles equal to 60° so that it becomes equilateral.

²⁰ Let the turtle run with `setpos(..)` along $(x-sz/2, y-sz/2)$, $(x + xz/2, y-sz/2)$, $(0, y + sz/2)$, back to $(x-sz/2, y-sz/2)$!

What is the direction of the turtle after having been guided by the code snippet in Table 4.7?²¹

How do we speed up Python's turtle function?

The answer is found with an internet search (2020):

<https://stackoverflow.com/questions/16119991/how-to-speed-up-pythons-turtle-function-and-stop-it-freezing-at-the-end>

with the answers:

- (1) Set `turtle.speed()` to `fastest`.
- (2) Use the `turtle.mainloop()` functionality to do work without screen refreshes.
- (3) Disable screen refreshing with `turtle.tracer(0, 0)`, then, at the end, do `turtle.update()`

We trigger variant (3) when we activate lines 8 and 25 in Table 4.6. The instruction `turtle.tracer(0, 0)` eliminates the millisecond delays that occur when the screen is updated after every turtle change. The screen is refreshed with the complete picture by `turtle.update()` (Table 4.8).

4.6 Drawing Densely-Packed Atomic Layers; Crystal Physics

We draw two different stackings of three planes with densely packed spheres that correspond to the cubic face-centered (*fcc*) or hexagonal dense-packed (*hdp*) crystal structure.

4.6.1 Program Structure and Geometry

In this task, a top view of a close-packed plane of atoms is to be drawn with a program based on a nested For-loop. The subordinate loop draws a horizontal row of discs, representing the atoms, touching each other. This is achieved by shifting a new disk to the right by the diameter d relative to the previous disc. In a higher-level loop, the rows are to be shifted one after the other in the plane so that the circular disks touch each other in a hexagonal arrangement, as shown in Fig. 4.19.

In a second development step, the main program resulting from the first step is to be converted into a procedure that draws a plane and to which the coordinates

²¹ The turtle turns $90 + 180 + 180 - 90 = 360^\circ = 0^\circ$ toward its original direction.

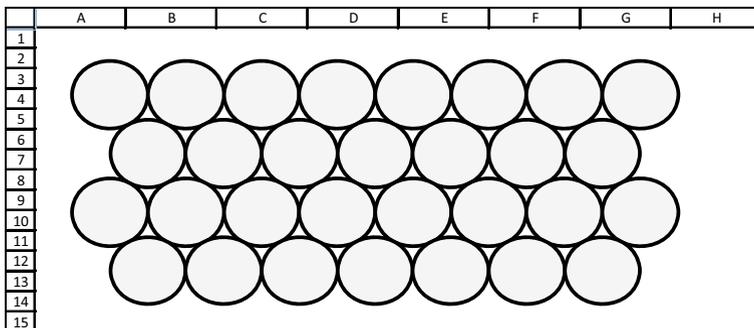


Fig. 4.19 (S) Hexagonally packed plane, drawn with SUB DIEB from Fig. 4.22 (P), here, however, in half-size

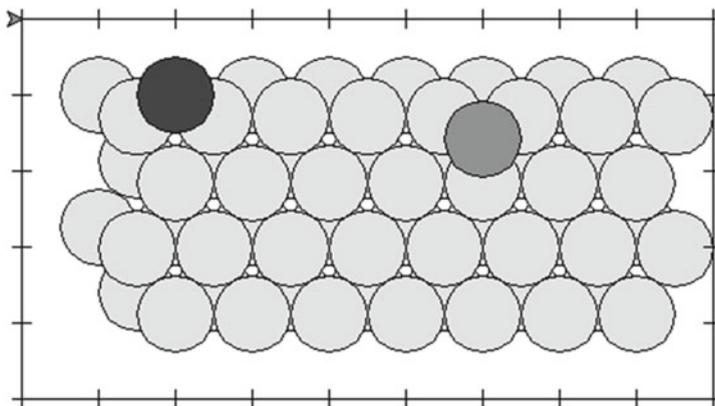


Fig. 4.20 Two densely packed planes and only two “atoms” in the third plane; dark grey “atom” with dotted border top left: position as in the hexagonal close packing (hcp); middle grey “atom” with black border: position as in the face-centered cubic (fcc) shape; the drawing is obtained with Tables 4.11 and 4.12

of the first disc are passed. An extended main program puts a second plane onto the gaps in the first plane, and ultimately places two discs, one at a position typical of the face-centered cubic (*fcc*) structure and the other one characteristic of hexagonal densest packing (*hdp*), resulting in Fig. 4.20.

Questions

How many neighbors does a sphere in a close-packed plane have?²²

²² Six nearest neighbours.

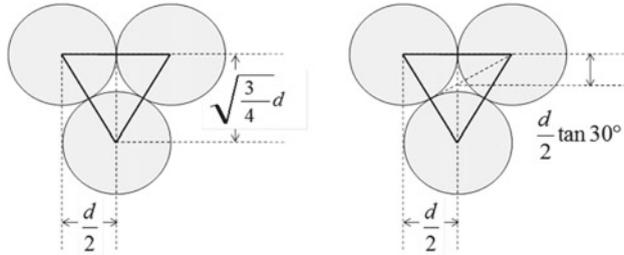


Fig. 4.21 **a** (left) Geometry of a hexagonal packing in a plane, displacement of the second row of atoms with respect to the first row of atoms. **b** (right) Position of an atom in the second plane

How many neighbors does a sphere in a stack of close-packed planes have?²³

The drawings in Fig. 4.21 indicate the coordinates of the centers of the circular disc in the planes. Figure 4.21a gives the position of a disc in a row relative to the previous row, and Fig. 4.21b that of a disc in a plane relative to the previous plane.

Three stacked planes

The first row of circular disks starts at (x_0, y_0) . The second row is offset from the first row in the x - and y -directions by distances indicated in the geometric construction in Fig. 4.21a. A second plane is to be placed over the first one, with the circular disks lying over the gaps in the first plane (see Fig. 4.21b), resulting in the assembly of light grey discs in Fig. 4.20.

For the third plane, there are two possibilities:

- It lies exactly above the first plane, as in the crystal structure of hexagonal close packing (*hcp*). For the drawing in Fig. 4.20, only one circular disk is placed in the correct position (dark grey, top left in the picture).
- It lies above the still visible gaps in the first plane, as in the cubic face-centered (*fcc*) crystal structure. In Fig. 4.20, this is done only for one circular disk, drawn in middle grey with a black border. The displacement of the third plane in the x - and y -directions is, for *fcc*, twice as large as the displacement of the second plane, both with respect to the first plane.

²³ 12 nearest neighbours, 6 of them in its own plane, 3 below and 3 above.

4.6.2 Data Structure and Nomenclature

The positions of the discs in the following list are deduced from Fig. 4.21:

d	disc diameter
(x_0, y_0)	position of the first disc, top left
$(delX, delY)$	shift of a plane with respect to (x_0, y_0)
$delX = 0, delY = 0$	for the 1st plane
$delX = d/2$	for the 2nd plane (see Fig. 4.21b)
$delY = -d/2 \cdot \tan(30/180 \cdot \pi)$	for the 2nd plane (see Fig. 4.21b)

$$x = x_0 + delX + dx + i \cdot d \text{ position of disc } i.$$

$dx = delX + 0$	1st and 3rd rows (see Fig. 4.21a)
$= delX + d/2$	2nd and 4th rows (see Fig. 4.21a)
$y = y_0 + dy$	y position of disc i (see Fig. 4.21a)
$dy = delY + d_{34} \cdot n$	nth row (see Fig. 4.21a and Eq. 4.1)

with:

$$d_{34} = d \cdot \sqrt{3/4} \quad (4.1)$$

read from Fig. 4.21a.

4.6.3 Excel

A row of discs

We draw four rows of circular disks so that the disks touch each other, using four consecutive FOR-loops (see SUB *DiEb* Fig. 4.22 (P)).

The sub-routine SUB *Disc(x,y)*, already reported in Fig. 4.10b, is called with individual positions x, y passed via the procedure header. Contrary to the figure, the disc's diameter is set in the main procedure to $d = 100$. In each row, the x -position of the following circle is shifted to the right by a circle's diameter. The y -position is always the same for a row.

Stacking planes

To draw several planes, one on top of the others, we convert the main program *DiEb* described above into the sub-routine SUB *Plane(delX, delY)* (repeated in Fig. 4.23 (P)) to which the coordinates of the top left disc are transferred. The displacements $delX$ and $delY$ can be determined with the help of Fig. 4.21b.

1	Sub DiEb()		15
2	x0 = 100	'3rd row of discs	16
3	y0 = 100	dx = 0 'shift with respect to 1st row	17
4	d = 100 'diameter of the disc	dy = d * Sqr(3 / 4) * 2	18
5	'1st row of discs	For i = 0 To 7	19
6	For i = 0 To 7	Call Disc(x0 + dx + i * d, y0 + dy)	20
7	Call Disc(x0 + i * d, y0)	Next i	21
8	Next i	'4th row of discs	22
9	'2nd row of discs	dx = d / 2 'shift with respect to 1st row	23
10	dx = d / 2 'shift with respect to 1st row	dy = d * Sqr(3 / 4) * 3	24
11	dy = d * Sqr(3 / 4)	For i = 0 To 6	25
12	For i = 0 To 6	Call Disc(x0 + dx + i * d, y0 + dy)	26
13	Call Disc(x0 + dx + i * d, y0 + dy)	Next i	27
14	Next i	End Sub	28

Fig. 4.22 (P) SUB *DiEb* for drawing a plane; the four rows of atoms are drawn using four loops, result shown in Fig. 4.19 (S)

1	Sub Plane(delX, delY)		15
2	x0 = 100 'Offset to left upper corner	dx = 0 + delx	16
3	y0 = 100 'of the worksheet	dy = d * Sqr(3 / 4) * 2 + dely	17
4	d = 100 'Diameter of the disc	For i = 0 To 7	18
5	dx = delx	Call Disc(x0 + dx + i * d, y0 + dy)	19
6	dy = dely	Next i	20
7	For i = 0 To 7	dx = d / 2 + delx	21
8	Call Disc(x0 + dx + i * d, y0 + dy)	dy = d * Sqr(3 / 4) * 3 + dely	22
9	Next i	For i = 0 To 6	23
10	dx = d / 2 + delx	Call Disc(x0 + dx + i * d, y0 + dy)	24
11	dy = d * Sqr(3 / 4) + dely	Next i	25
12	For i = 0 To 6	End Sub	26
13	Call Disc(x0 + dx + i * d, y0 + dy)		27
14	Next i		28

Fig. 4.23 (P) SUB *DiEb* is converted into a sub-routine *Plane* to which the initial coordinates are transferred by a higher-level program

In the main program SUB *hcp_fcc* in Fig. 4.24 (P), SUB *Plane* is called twice, for the initial layer with $\text{delX} = 0$ and $\text{delY} = 0$ and for the second layer with its

1	Sub hcp_fcc()	dy = 0	10
2	x0 = 100	dx = 100 * 1	11
3	y0 = 100	Call Disc(x0 + dx, y0 + dy)	12
4	Call Plane(0, 0)	'fcc	13
5	delx = 100 / 2	dy = 100 * Sqr(3 / 4) - 100 / 2 * Tan(30 _	14
6	dely = 100 / 2 * Tan(30 _	/ 180 * 3.14159265)	15
7	/ 180 * 3.14159265)	dx = 100 * 5	16
8	Call Plane(delx, dely)	Call Disc(x0 + dx, y0 + dy)	17
9	hdp	End Sub	18

Fig. 4.24 (P) Main program, which calls SUB *Plane* twice, places two atoms on top (with SUB *Disc* from Fig. 4.10b (P)), and thus draws a picture similar to Fig. 4.20

discs on the gaps of the first layer. There are two possibilities for the third plane, *hcp* or *fcc*, represented with one disc each in lines 12 and 17.

Questions

In Fig. 4.23 (P), the variables in the header are called *delX* and *dely*. In the body of the procedure, the formulas refer to different names, *delx* and *dely*. Will this discrepancy lead to error messages?²⁴

How do you have to change SUB *Disc* in Fig. 4.10b (P) so that *d* becomes a global variable?²⁵

Grouping and copying shapes in various picture formats

To group the shapes into an integrated picture, activate the white arrow SELECT OBJECTS in the FIND&SELECT tab (far right in the HOME tab of the EXCEL ribbon, Fig. 1.1 in Sect. 1.7), drag the selection rectangle around the shapes, then click FORMAT/GROUP. You can now copy the group as one graphic into other applications, e.g., into a Word file or a PowerPoint file.

► **Task** Group your drawing into an image of type *png*, *tif*, or some other image format, and copy this image to another area of the spreadsheet or to another application, e.g., to a PowerPoint file! To do so, select the object, click COPY, move the cursor to another location in the table, click PASTE/PASTE SPECIAL, and select the desired format.

4.6.4 Python

In the first cell of Table 4.9, the relevant libraries are imported. In the next cell, a screen with the name “Crystal planes” is created, and global parameters are specified, with the disc diameter *d* being set to 50 and the position of the first disc in the upper left corner at $(x_0, y_0) = (-175, 175)$. In the third cell, the first row of discs is drawn, comprising eight discs drawn from left to right, using the function *drawCircle* in Table 4.4. When developing the program, you should check this snippet of code and see whether a row of gray discs is really plotted from left to right.

Comments concerning Table 4.9:

- Line 1, link for an introduction into the basic features of turtle,
- Line 5, the screen is cleared. This is important when developing a program, and you have to improve the code and repeat it again and again until it runs error-free.

²⁴ No, Visual Basic is case-insensitive, *contrary to Python*.

²⁵ Skip line 28 “*d = 50*”, insert DIM *d* AS INTEGER as the first program line, and specify *d = ...* somewhere in the program before SUB *Disc* is called for the first time.

Table 4.9 Importing relevant libraries; creating a screen and setting global parameters; drawing 1st row of discs, function *disc* from Table 4.4

```

1  #https://docs.python.org/3.3/library/turtle.html
2  import turtle
3  import numpy as np
4  import numpy.random as npr
5  turtle.clearscreen()
6  t=turtle.Turtle()           #Create turtle with name t!
7  turtle.title("Crystal planes")
8  tup=(0.9,0.9,0.9)         #Light grey
9  t.pen(pencolor="black", fillcolor=tup, pensize=1, speed=10)
10
11 x0=-175
12 y0=175
13 d=50                       #Disc diameter
14 #1st row of discs
15 dx=0
16 dy=0
17 for i in range(8):
18     drawCircle(x0+dx+i*d,y0+dy,d,fill=True)
19 print(t.pos())             #Current position

```

Table 4.10 Function for drawing a close-packed plane; *delX*, *delY* position of top left disc with respect to (x_0, y_0) , *drawCircle* from Table 4.4

```

1  def Plane(delX,delY):
2      #1st row of discs
3      dx=0+delX
4      dy=0+delY
5      for i in range(8):
6          drawCircle(x0+dx+i*d,y0+dy,d,fill=True)
7      #2nd row of discs
8      dx=d/2+delX
9      dy=-d*np.sqrt(3/4)+delY
10     for i in range(7):
11         drawCircle(x0+dx+i*d,y0+dy,d,fill=True)
12     #3rd row of discs
13     dx=0+delX
14     dy=-d*np.sqrt(3/4)*2+delY
15     for i in range(8):
16         drawCircle(x0+dx+i*d,y0+dy,d,fill=True)
17     #4th row of discs
18     dx=d/2+delX
19     dy=-d*np.sqrt(3/4)*3+delY
20     for i in range(7):
21         drawCircle(x0+dx+i*d,y0+dy,d,fill=True)

```

Table 4.11 Plotting the first and second planes

22	Plane(0,0)
23	delX=50/2
24	delY=-50/2*np.tan(30/180*np.pi)
25	Plane(delX,delY)

Questions

concerning Table 4.9

What is the position of the turtle after having drawn the first row?²⁶

What are the pen color and fill color of the turtle?²⁷

What is the global parameter accessed in *drawCircle*?²⁸

To produce a figure like that in Fig. 4.20, two planes and, additionally, two discs have to be placed at appropriate positions. The function for drawing a close-packed plane is shown in Table 4.10. Its arguments are the shifts *delX* and *delY* of the first disc's position with respect to (x_0, y_0) . The four rows of discs are drawn with loops calling *drawCircle* (from Table 4.4), always with the same variables, but with *dx* and *dy* set for each row individually, according to Eq. (4.1).

The first two planes are drawn with the program in Table 4.11. Two atoms in the third plane representative of the hexagonal-dense packed and face-centered cubic structures, respectively, are drawn in Table 4.12. For *hdp*, the position is just on top of a disc in the first plane, whereas for *fcc*, *x* and *y* get a double shift from a disc in the first plane, whereby the *x* position becomes identical to the neighboring disc in the first plane and the *y* position is in the center of a gap in the second plane.

Question

In line 29 of Table 4.12, $dx = 50*1$ is specified; the number 50 is precisely the disc diameter. Nevertheless, what is bad about this instruction, and likewise about the instructions in lines 36 and 37, and lines 23 and 24 in Table 4.11?²⁹

²⁶ The position of the turtle after the first row is $x = -175 + 7*50 = +175$; $y = 175$, as can be deduced from lines 18 and 19 in Tab. 4.9.

²⁷ `pencolor = "black", fillcolor = tup, tup = (0.9, 0.9, 0.9)`, a light gray.

²⁸ The disc diameter *d* is a global parameter accessed in *drawCircle*. It is also used in line 18 to specify the shift of the position of the current disc with respect to the preceding one.

²⁹ When the value of the global parameter *d* is changed, these instructions do not follow. Set $dx = d*1$ in lines 47, 48, 53, 60, 61 instead!

Table 4.12 Drawing an atom of the third plane, either *hdp* or *fcc*

```

26 #hdp
27 tup=(0.3,0.3,0.3)      #Dark grey
28 t.pen(pencolor="black", fillcolor=tup,pensize=1, speed=0)
29 dx=50*1
30 dy=0
31 drawCircle(x0+dx,y0+dy,d,fill=True)
32
33 #fcc
34 tup=(0.6,0.6,0.6)      #Middle grey
35 t.pen(pencolor="black", fillcolor=tup,pensize=1, speed=0)
36 dx=50*5
37 dy=-50/2*np.tan(30/180*np.pi)*2
38 drawCircle(x0+dx,y0+dy,d,fill=True)

```

4.7 Text Processing

Operations on strings (texts) are practiced by the example of swapping letters within words. We will use the functions *Len*, *Split*, *Join* in both Visual Basic and Python. The functions LEFT, RIGHT, and MID are used in VISUAL BASIC to cut out pieces of strings, achieved in Python through list slicing.

4.7.1 Cutting and Joining Strings

Swirling characters

► **Text** It is said that a text can be understood even if you have only the beginning and the end letter in each word in place but exchange middle letters. Do you believe that or is it nonsense?

► **Task** Write a program that first reads a text from a spreadsheet. This text is then to be broken down into words. The individual words are transformed so that the first and last letters remain in place, but the inner letters are randomly swapped.

We are going to write a program that exchanges letters in an originally correctly written text. Such a program has already swirled the first two paragraphs of this description. The number of letter swappings in the above text: 1 (Text), 2 (Task).

Plain text of the first two paragraphs

► **Text** It is said that a text can be understood even if you leave only the beginning and the end letter in each word in place but exchange middle letters. Do you believe that, or is it nonsense?

► **Task** Write a program that first reads a text from a spreadsheet. This text is then to be broken down into words. The individual words are transformed so that the first and last letters remain in place, but the inner letters are randomly swapped.

VBA instructions for text processing

We need the following VBA instructions that affect character strings:

LEN(...), SPLIT(...), JOIN(...), LEFT(...), RIGHT(.) and MID(...).

An overview is given in the box. For more information, refer to VBA help! There, you can find out, for example, about the MID function:

Returns a Variant (String) containing a specified number of characters from a string.

► **Syntax** MID (STRING, START [, LENGTH]).

STRING Required. String expression from which characters are returned.

START Required; Long. Character position in STRING at which the part to be taken begins.

LENGTH Optional; Variant (Long). Number of characters to return.

Text processing in VBA and Python

VISUAL BASIC	Python
SPLIT(STRING, [SEPARATOR...])	String.split("separator")

Splits a string expression into words and stores them in an array. Unless otherwise specified, a space is interpreted as a separator between words.

LEN(STRING)	len(String)
-------------	-------------

Determines the length of a given string.

LEFT(STRING, LENGTH)	String[:Length]
----------------------	-----------------

Cuts out a piece of length *Length* from a character string starting from the left.

RIGHT(STRING, LENGTH)	String[-Length:]
-----------------------	------------------

Cuts out a piece of length *Length* from a character string starting from the right.

MID(String, Start, Length) String[Start: Stop]

Cuts out a piece of length LENGTH or (Stop - Start) from a character string starting from position START and to the right of it. Attention: VISUAL BASIC starts indexing from 1, Python from 0.

JOIN(Array[, DELIMITER) 'Delimiter'.join(Array)

Returns a string created by joining several substrings contained in an array. If 'Delimiter' is omitted, the space character (" ") is used.

Questions

Let the variable *Tx* contain the text "Cutting out". With which instructions do you get the first, last, and second characters of *Tx*? How do you copy the string "ing" from *Tx* to a new variable *Wd*? In Visual Basic,³⁰ in Python?³¹

Consider the string *Sente* = "This is. Our goal." How do you separate the string into the two sentence fragments terminated by full stops? Use the Split command! How do you get a new string *JS* = "This is." including a full stop? EXCEL?³² Python?³³ Compare!

4.7.2 Data Structure and Program Flow

Text text to be processed, words separated by spaces

Words list of the words in *Text*

For every *Word* in *Words*:

 Exchange two letters with function *ExchLett(Word)*

 Split *Word* → *Letters* list of letters in *Word*

 Join *Letters* → *NewWord*

Join sequence of *NewWord* → *NewSentence*

³⁰ LEFT(*Tx*, 1), RIGHT(*Tx*, 1), MID(*Tx*, 2, 1), *Wd* = MID(*Tx*, 5, 3); the first character has index 1.

³¹ *Tx*[0], *Tx*[-1], *Tx*[1], *Wd* = *Tx*[4:7]; the first character has index 0.

³² SINGSENT = SPLIT(SENTE, "."); DIM NEWSSENT(1) AS STRING: NEWSSENT(0) = SINGSENT(0): NEWSSENT(1) = ".": JS = (NEWSSENT, ""), 5 statements; the colon is the separator between statements in a line.

³³ SingSent = Sente.split("."); JS = ''.join([SingSent[0], "."]), 2 statements; the semicolon is the separator between statements in a line.

In the VISUAL BASIC program in Sect. 4.7.3, we use one array of words containing the original words at the beginning and the new words at the end. In Python in Sect. 4.7.6, we are using two arrays: `Words`, with original data remaining unchanged, and `WordsNew` obtained consecutively by appending one scrambled word after the other.

4.7.3 Excel

In the following, first, a complete program (main program *Scramble* and sub-routine *XWord*) that solves the task is introduced. Do not copy it, but rather continue reading! Then, the program is developed step by step in test macros to follow the effect of the individual instructions.

Please be aware that, in the following continuous text, according to our spelling convention, the words in SMALL CAPS are VBA internal terms, while the words in *italics* are invented by the programmer.

Split (Sentence)

The main program *Scramble()* (Fig. 4.25 (P)) reads a text from cell A1 of the spreadsheet (*Text* = `CELLS(1,1)`), splits it into words (*Words* = `SPLIT(Text)`, line 3) and passes the words one by one to the sub-routine *XWord(Word)* (line 7) which exchanges two letters. If two letters are to be exchanged for a second time, line 8, now commented out, has to be activated. The instruction `SPLIT` specifies the variable *Words* automatically as an array. For the data type *array*, see Sect. 4.7.5.

The sub-routine *XWord (Word)* (Fig. 4.26 (P)) splits the word transferred via the header into individual letters (lines 18–22), randomly exchanges two inner letters (lines 23–28), puts together the new word in lines 30 to 34, and returns the modified word to the higher-level procedure from which the sub-routine was called.

1	Sub Scramble()	Call XWord(Words(IS)) ' <i>Letters are exchanged</i> '	7
2	Text = Cells(1, 1)	'Call XWord(Words(IS)) ' <i>... a second time</i> '	8
3	Words = Split(Text)	Next IS	9
4	For IS = 0 To UBound(Words)	newText = Join(Words, " ")	10
5	'Cells(IS + 1, 2) = Words(IS)	Cells(7, 1) = newText	11
6	' <i>activate for Scramble_test</i>	End Sub	12

Fig. 4.25 (P) SUB *Scramble* reads a text from cell A1, splits it into words stored in the array *Words*, and passes the words one by one to the sub-routine *XWord*. The words returned by *XWord* (in the variable *Word* in the header) are assembled into a new sentence in line 10 and output to cell A7 (`CELLS(7,1)`) of the spreadsheet. Line 8, now a comment, has to be activated when two letters are to be exchanged for a second time

13	Sub XWord(Word)	<i>'Random positions 2 to lWord-1</i>	25
14	Dim Letter(20) As String	L0 = Letter(n1)	26
15	Debug.Print (Word)	Letter(n1) = Letter(n2)	27
16	lWord = Len(Word) '#Letters in the word	Letter(n2) = L0	28
17	If lWord >= 4 Then		29
18	For n = 1 To lWord	For n = 1 To lWord	30
19	<i>'Letters are singled out.</i>	Word = Join(Letter, "")	31
20	Letter(n) = Mid(Word, n, 1)	<i>'Cells(n, 4) = Letter(n)</i>	32
21	<i>'Cells(n, 3) = Letter(n)</i>	<i>'Output to spreadsheet in Xword_test</i>	33
22	Next n	Next n	34
23	n1 = Int(Rnd() * (lWord - 2)) + 2	End If	35
24	n2 = Int(Rnd() * (lWord - 2)) + 2	End Sub	36

Fig. 4.26 (P) SUB *XWord* detects the length of the transferred word (line 16), splits it into letters (lines 18–22), swaps two inner letters (lines 23–28), and reassembles the letters into the modified word (line 31). Commented lines 21 and 32 have to be activated to obtain SUB *XWord_Test*, mentioned in Sect. 4.7.4

Join (Sentence)

The main program SUB *Scramble* in Fig. 4.25 (P) reassembles the modified words into a text ($newText = JOIN(Words, " ")$, line 10) and writes it into cell A7 ($Cells(7,1) = newText$). The second entry " " in JOIN causes a space to be inserted after each element of the array *Words*.

VBA terms and user-defined variable names

In the procedures *Scramble* and *XWord*, there are terms that VBA assigns a *precisely defined meaning* to:

- CELLS(*r*, *c*): cell in the row *r* and column *c* of the current spreadsheet,
- FOR ... TO ...; DO WHILE ... LOOP; ON ERROR GOTO;
- The functions SPLIT(...); JOIN(...); INT(...); RND().

Such terms are printed in the text in SMALL CAPS.

There are also eleven *variable names*, which the programmer has invented himself/herself:

- *Text*, *Words*, *lSent*, *newText*, *lWord*, *Letter*, *n1*, *n2*, *L0*, *x*, *n*.

He might as well have taken eleven letters:

- *a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*, *i*, *j*, *k*,

or eleven combinations of letters and numbers:

- *a1*, *a2*, *a3*, *a4*, *b5*, *b6*, *b7*, *b8*, $\times 1$, $\times 2$, $\times 3$.

All combinations of letters and numbers are allowed as variable names, but the first character must be a letter. Such names are *italicized* in the text. To keep the program clear, you should choose variable names that easily convey their meaning in the program.

Attention: *l* (small el) and *I* (capital i) can easily be confused! Variable names in different places of the program then look the same but designate two different variables. So, it is better to use a capital *L*: *LWord* instead of *lWord*.

Questions

What should lines 2 and 11 of Fig. 4.25 (P) be if the statement is formulated with RANGE instead of CELLS?³⁴

Which variable names in Fig. 4.25 (P) did the programmer come up with himself/herself?³⁵

Which variable names in Fig. 4.26 (P) did the programmer come up with himself/herself?³⁶

Do lines 23 and 24 of Fig. 4.26 (P) guarantee that two letters are always exchanged?³⁷

4.7.4 Programming Step by Step

Step by step, we develop a program that performs the text swirling described in Sect. 4.7.1. It interacts with the spreadsheet, i.e., reads from cells and fills in the spreadsheet cells as shown in Fig. 4.27 (S).

	A	B	C	D
1	A sentence is to be decomposed.	A	s	s
2		sentence	e	e
3		is	n	n
4		to	t	t
5		be	e	n
6		decomposed.	n	e
7	A sentnece is to be decomposed.		c	c
8			e	e

Fig. 4.27 (S) A1 contains the sentence to be processed. B, C, D, and cell A7 are filled in by the program. The individual words are in B, the individual letters of the second word are in C, those of the swirled word in D

³⁴ *Text* = RANGE("A1"); RANGE("A7") = *newText*.

³⁵ *Text*, *Words*, *IS*, *newText*.

³⁶ *Word*, *Letter*, *lWord*, *n*, *n*₁, *n*₂, *L*₀.

³⁷ No, *n*₁ and *n*₂ can be identical, so that there is no effective exchange.

We modify the procedures SUB *Scramble* in Fig. 4.25 (P) by activating the still out-commented line 5 (filling in column B of Fig. 4.27 (S)) and SUB *XWord* by activating lines 21 (filling in column C) and 32 (filling in column D).

The sub-routine *XWord_Test(Word)* is the same as *XWord(Word)* in Fig. 4.26 (P), however, with the lines 21, 22, and 33 to 35 activated that now output intermediate results into the spreadsheet.

Question

concerning Fig. 4.26 (P):

Which program lines guarantee that the first and last letter of a word are not displaced? From which range of the spreadsheet is the text to be processed read? Into which ranges of the spreadsheet are the individual words of the text, the individual letters of a word, the swirled letters, and the modified text written?³⁸

After checking these macros to see if they do what we want them to do, we transform them into a procedure that reads a sentence from cell A1 and outputs the changed sentence in A2, like SUB *Scramble* in Fig. 4.25 (P). We now only swap letters from the word's interior, i.e., leave the first and last letters as they are.

► **Task** Do this exercise with other texts as well, and surprise your friends with playful letters!

4.7.5 VBA Constructs

The data type Array in VBA

Arrays are declared in VBA as follows:

```
DIM Variable name(shape) AS data type
```

For example, [DIM *Fel*(2) AS DOUBLE] defines an array with three cells (to be addressed with 0, 1, 2), where each cell can contain a real number of type DOUBLE. DIM AR(2,3) AS INTEGER defines a two-dimensional array of integers of shape 3 rows × 4 columns.

4.7.6 Python

The basic functions and methods for text processing, namely, splitting a text into words, a word into letters, and, the other way around, joining letters to form a new

³⁸ Text read from A1 (CELLS(1,1)), words written into B, letters of the selected word into C, swirled letters into D (CELLS(R,4)), modified text into A7. Compare with Fig. 4.27 (S)!

Table 4.13 Basic functions for text processing

1	Text="""A sentence is decomposed."""	
2	Words=Text.split()	
3	Letters=list(Words[1])	
4	L=Letters[4]	
5	NewWord=':'.join(Letters[1:-1]) #Concatenate with ":"	
6	print(Words)	['A', 'sentence', 'is', 'decomposed.']
7	print(Letters)	['s', 'e', 'n', 't', 'e', 'n', 'c', 'e']
8	print(L)	e
9	print(NewWord)	e:n:t:e:n:c

word, are presented in Table 4.13. The `join()` method creates a new string from “precursor strings”, e.g., a list of letters. The letters in the new word are separated by a colon as specified in a prefix to `join` (see lines 5).

These basic functions are again applied in Table 4.14 to split a longer text into words, pass each individual word to the function *ExchLett* (reported in Table 4.15), and join the scrambled words into a new text, with blanks as separators. The content of the variable *Text* starts with three quotation marks """ , indicating that the following text covering several lines up to the next three quotation marks """ is a string. To enter multi-line strings, use triple codes to start and end them!³⁹

The function *ExchLett* in the third cell of Table 4.15 uses the function `random.sample` from the `random` library to choose two different internal letters and then exchange them. This function was found with a search in *stackoverflow.com*. Its syntax and mode of action can be deduced from the second cell, which presents the result of line 23.

In lines 30 to 32, the temporary variable L_0 is introduced in order to swap two variables. The code in line 34 does the same, but without the use of any temporary variable. In Python, a backslash (\) indicates that the instruction line is continued. Statements can also be split up after a comma.

³⁹ <https://stackoverflow.com/questions/10660435/pythonic-way-to-create-a-long-multi-line-string>.

Table 4.14 Rewriting scrambled words in a text **a** (left) program; **b** (right) result

```

10 Text="""It is said that you can read a text also if you
    leave only the beginning and the end letter in each word as
    they are and swirl two middle letters. Do you believe that
    or is it nonsense?"""
11
12 Words=Text.split()
13 WordsNew=[]
14 for i in range(len(Words)):
15     WN= ExchLett(Words[i])
16     WordsNew.append(WN)
17 sentNew=' '.join(WordsNew)           #Concatenate with blank
18 print(sentNew, "\n")

```

```

It is siad taht you can raed a txet aslo if you lvae olny the
beginning and the end lteter in ecah wrod as tehy are and
swril two mildde ltteers. Do you belveie taht or is it
noesnse?

```

Table 4.15 Swapping internal letters

```

19 #https://stackoverflow.com/questions/9755538/
20 #how-do-i-create-a-list-of-random-numbers-without-
    duplicates
21
22 import random
23 random.sample(range(0,10), 10)
    [7, 2, 5, 9, 0, 6, 3, 4, 8, 1]
24 def ExchLett(Word):
25     #Split a word into letters
26     Letters=list(Word)
27     #Exchange two internal letters of a word
28     if len(Letters)>=4:
29         n=random.sample(range(1,len(Letters)-1), 2)
30         L0=Letters[n[0]]
31         Letters[n[0]]=Letters[n[1]]
32         Letters[n[1]]=L0     #Join letters into a word!
33     return ''.join(Letters)
34 Letters[n[0]],Letters[n[1]]=\
    Letters[n[1]],Letters[n[0]]

```

Table 4.16 Scrambling a text passage

```

1 Task="""Write a program that first reads a text from a
  spreadsheet. This text is then to be broken down into
  words. The individual words are transformed so that the
  first and last letters remain in their place, but the inner
  letters are randomly swapped."""
2 Words=Task.split()
3 WordsNew=[]
4 for i in range(len(Words)):
5     WN= ExchLett(Words[i])
6     WordsNew.append(WN)
7 sentNew=' '.join(WordsNew)
8 print(sentNew)

```

Wrtie a porgram taht frist rdaes a txet form a sprdaesheet.
 Tihst txet is tehn to be broekn dwon itno worsd. The indaviduul
 wodrs are transofrmed so taht the fisrt and lsat lettres
 remian in thier palce, but the inner lettres are rdnaomly
 seappwd.

Questions

Write lines 30 to 32 of Table 4.16, swapping two variables using the temporary variable L0, as one statement!⁴⁰

How can lines 5 and 6 of Table 4.15 be merged into one statement?⁴¹

How can you introduce an additional line of code into Table 4.16 to achieve two letter swappings in a word?⁴²

Table 4.16 processes the second text passage, now performing two letter exchanges.

4.8 Processing the Protocol of a Measuring Device

Continuous text is decomposed and rearranged in tables. Knowledge acquired by processing texts in the previous exercise is applied to separate text and numbers in reports created by measuring instruments, with the aim to recognize code words and rearrange the essential results in tables.—The Python program is based on the library Pandas.—A piece of advice: With

⁴⁰ Letters[n[0]], Letters[n[1]] = Letters[n[1]], Letters[n[0]].

⁴¹ WN = EXCHLETT(EXCHLETT(WORDS[I])).

⁴² Introduce WN = ExchLett(WN) after line 5!

the knowledge gained in this exercise, you may earn a small bit of extra income from a side job in scientific projects!

4.8.1 Protocol of a Measuring Device

Many measuring devices output a plain-text file as a protocol containing both text and numbers. As an example, we will use the output of chemical analysis with RBS (Rutherford Back Scattering) concerning the composition of four Nb-doped TiO₂ layers on a silicon substrate (spreadsheet in Fig. 4.28 (S)). Every layer contains different fractions of Ti (titanium), O (oxygen), Nb (niobium), and Ar (argon). We will convert this information into a table, as in Fig. 4.29 (S).

	A	B	C	D
1	T1.lay	T2.lay	T3.lay	T4.lay
2	!-----	!-----	!-----	!-----
3	d=0.20E18	d=0.25E18	d=0.30E18	d=0.35E18
4				
5	Ti#,1	Ti#,1	Ti#,1	Ti#,1
6	O#,2.5	O#,2.4	O#,2.3	O#,2.2
7	Nb#,0.03	Nb#,0.05	Nb#,0.07	!-----
8	Ar#,0.01	!-----	Ar#,0.008	s=
9	!-----	s=	!-----	Si#,1
10	s=	Si#,1	s=	
11	Si#,1		Si#,1	

Fig. 4.28 (S) Protocol of RBS measurements, transferred into an EXCEL spreadsheet; row 1 = names of four different samples; parameters: d = number of atoms per cm²; Ti, O, Nb, Ar = elements found in the layer with their indices in the chemical formula

	A	B	C	D	E	F
1						
2	SampNam	NAtoms	Ti	O	Nb	Ar
3	T1	2,00E+17	1	2,5	0,03	0,010
4	T2	2,50E+17	1	2,4	0,05	
5	T3	3,00E+17	1	2,3	0,07	0,008
6	T4	3,50E+17	1	2,2		
7						

Fig. 4.29 (S) The data from Fig. 4.28 (S) have been written into this table in a spreadsheet with the name “TabLay”. Each sample has its own row. The first row is left blank in order to insert an index for the next free row later

36	For r = 3 To r2	cll = Left(cl, 2) <i>'take first two letters</i>	39
37	cl = Cells(r, sample) <i>'e.g.: d=0.20E18</i>	If cll = "d=" Then NAtoms = Right(cl, Le - 2)	40
38	Le = Len(cl) <i>'length of the string</i>	Next r	47

Fig. 4.30 (P) Cuts off the first two letters of a data string (line 39) and checks whether this is the code word “d =”

1	Sub DecodeRBS()	For sample = 1 To 4	15
2	<i>'Original data in sheet "RBS-data"</i>	Sheets("RBS-data").Select <i>'original data</i>	16
3	<i>'Table with sample characteristics in "TabLay"</i>	<i>'Get sample name!</i>	17
4	Dim cl, cll As String	SampNam = Cells(1, sample) <i>'sample name</i>	18
5	<i>'Write headers!</i>	Le = Len(SampNam) <i>'length of the name</i>	19
6	r3 = 2	SampNam = Left(SampNam, Le - 4)	20
7	Sheets("TabLay").Select	<i>' ".lay" is removed</i>	21
8	Cells(r3, 1) = "SampNam"	<i>'Identify range with information on sample!</i>	22
9	Cells(r3, 2) = "NAtoms"	For r1 = 3 To 30 <i>'scans rows 3 to 30</i>	23
10	Cells(r3, 3) = "Ti"	cl = Cells(r1, sample) <i>'Content of cell</i>	24
11	Cells(r3, 4) = "O"	cll = Left(cl, 5)	25
12	Cells(r3, 5) = "Nb"	If cll = "l----" Then r2 = r1 - 1	26
13	Cells(r3, 6) = "Ar"	<i>'r2 = last row of information on the sample</i>	27
14	r3 = r3 + 1 <i>'next free row for output</i>	Next r1	28

Fig. 4.31 (P) Complete program for rearranging the raw data from Fig. 4.28 (S) into a table as in Fig. 4.29 (S); continued in Fig. 4.32 (P)

4.8.2 Detection of Code Words

The information in a column of Fig. 4.28 (S) is to be decoded and stored in a row of Fig. 4.29 (S). The main task is to identify certain code words that indicate the physical or technical quantity to which the following numbers refer. An extract from the complete Visual Basic decoding program (Fig. 4.32 (P)) can be found in Fig. 4.30 (P).

The data set for a layer is read line by line. The data strings “d = 0.20E18” and “Ti#,1” contain information about the number of atoms per cm² and the titanium content in the sample, respectively. When the program processes the file, it is not clear from the outset what type of data string is currently involved. For decoding, therefore, the first parts of the data line are separated (line 39 in Fig. 4.30 (P)), and it is queried as to whether this part is “d =” (line 40). If this is the case, the following string is interpreted as a number and written into the corresponding variable; for “d =”, this is *NAtoms* (line 40 in Fig. 4.30 (P)).

In Python, the task is tackled with the library *Pandas*, which mimics spreadsheet calculation.

4.8.3 Data Structure and Nomenclature

wb	name in <i>Pandas</i> of the workbook containing the data
sh	name in <i>Pandas</i> of the worksheet within the workbook
T1, T2, T3, T4	identifiers of samples

„!----“	code word to indicate the end of useful information
“d=”	two-character code word for atomic coverage, atoms per cm ²
“O#,”	three-character code word for oxygen
“Ti#,” “Nb#,” “Ar#,”	four-character code words for titanium, niobium, and argon
r ₂	last row of useful information
r ₃	current row in the spreadsheet for output
NAtoms	number of atoms per cm ²
O, Ti, Nb, Ar	fraction of the respective element

4.8.4 Excel

The complete decoding procedure can be found in Fig. 4.31 (P) and Fig. 4.32 (P). SUB *DecodeRBS* writes headings into the spreadsheet “TabLay” (lines 7 to 13), successively reads rows 3 through 30 from the spreadsheet “RBSData” (line 23, index r₁), determines the last row of useful data (line 26), decodes the useful data (FOR loop in rows 36 through 47), and finally writes the decoded data row by row (index r₃) into the spreadsheet “TabLay” (rows 49 through 55).

The loops are of type *loop2i*, with the loop index *r* and the running index *r₃*, indicating the next free row in the table and set to 2 at the beginning (line 6) and incremented in line 56 after the extracted values for the parameters have been entered into the output table within the For-loop with the index *r* scanning the input table.

Since the code words have different lengths (“d =” has two letters, “Ti#,” has four letters), in lines 39, 41, and 43, two, three, and four letters are, one after the other, cut off from the beginning of the data string and it is checked as to whether they correspond to one of the code words. The part of the data string following

```

29 'Remove old information!                                If cll = "Ti#," Then Ti = Right(cl, Le - 4) 44
30 NAtoms = Empty                                         If cll = "Nb#," Then Nb = Right(cl, Le - 4) 45
31 O = Empty                                               If cll = "Ar#," Then Ar = Right(cl, Le - 4) 46
32 Ti = Empty                                              Next r                                       47
33 Nb = Empty                                              'Write decoded data into a different sheet! 48
34 Ar = Empty                                              Sheets("TabLay").Select                    49
35 'Decode information in rows 3 to r2!                   Cells(r3, 1) = SampNam                     50
36 For r = 3 To r2                                         Cells(r3, 2) = NAtoms                     51
37   cl = Cells(r, sample) 'e.g.: d=0.20E18              Cells(r3, 3) = Ti                         52
38   Le = Len(cl) 'length of the string                  Cells(r3, 4) = O                          53
39   cll = Left(cl, 2) 'take first two letters           Cells(r3, 5) = Nb                         54
40   If cll = "d=" Then NAtoms = Right(cl, Le - 2)        Cells(r3, 6) = Ar                         55
41   cll = Left(cl, 3) 'take first three letters         r3 = r3 + 1 'next free row              56
42   If cll = "O#," Then O = Right(cl, Le - 3)           Next sample                             57
43   cll = Left(cl, 4) 'take first four letters         End Sub                                   58

```

Fig. 4.32 (P) Continuation of Fig. 4.31 (P)

the cut-off contains a number separated with RIGHT(..) and then assigned to the corresponding variable (e.g., $NAtoms$, Ti , ...). This separation is easy because all numbers have the same format.

Questions

Which code word is queried in Fig. 4.30 (P)?⁴³

Which code words in Fig. 4.32 (P) have length 4?⁴⁴

Module in VBA

Since the program SUB *DecodeRBS* refers, with SHEETS("RBS-data").SELECT and SHEETS("TabLay").SELECT, to two different spreadsheets, it must be operated in a module. If it stands in the VBA sheet associated with a spreadsheet, it operates only in that spreadsheet with instructions like CELLS(r,c) = .

VBA keyword empty

In lines 36 to 47 of Fig. 4.32 (P), the variables O, Ti, Nb and Ar are only filled in if the assigned code words occur in the original protocol. If a code word does not occur in the current data line, the content of the associated variable is not overwritten, and the old value persists. To prevent this from happening, we have entered the assignments in lines 30 to 34, e.g., Nb = EMPTY. EMPTY is a VBA keyword, to ensure that the variables do not contain any value when a new sample is processed.

Sample T4 does not contain any Nb; cell E6 in Fig. 4.29 (S) thus remains empty. If we had not set Nb = EMPTY before decoding, then Nb would still contain the value 0.07 of the previous sample and would have been incorrectly entered into the table of results. This would be a grave error in content!

The table is continued the next time the macro is called

► **Tim** The presented program sorts the data of exactly four samples into another table. What if a new set of samples comes in and the table is to be continued? Can the program remember the value of the index r_3 for the next free row and use it for the next call?

► **Mag** No, the program forgets the values of the variables when it finishes its execution.

► **Alac** Then, I will simply adapt the code before each new call. In line 14, the next free row in the table of Fig. 4.29 (S) is entered, $r_3 = 7$, and for the last FOR loop index in line 15, I will enter the current number of samples.

⁴³ „d = “ is queried.

⁴⁴ The code words „Ti#“, „Nb#“, and „Ar#“, have the length 4.

► **Mag** That’s a practical idea, and it works. But it can also be done more elegantly with the following two pieces of program.

You can use the first row in Fig. 4.29 (S), still empty, e.g., cell A1, to store the number of the first free row after the previous entries, and cell D1 to specify the number of new samples. This information is then read with $r_3 = \text{RANGE}(\text{“A1”})$ in line 14 and ... $\text{TO RANGE}(\text{“D1”})$ in line 15. Cell A1 is now overwritten with the value of r_3 by a new instruction at the end of `SUB DecodeRBS`. The number of new samples must be entered manually in D1 when a new data series is to be decoded, or the programmer can devise a query to automatically determine the number of samples in the raw data of Fig. 4.28 (S).

DO ... LOOP UNTIL

The next free row can also be determined by querying the current cell content in a loop to see if the current cell is empty:

```
DO
   $r_3 = r_3 + 1$ 
LOOP UNTIL CELLS( $r_3$ , 1) = EMPTY
```

Similarly, the number of NOT EMPTYs in a new raw data file can be obtained.

For more information on the instructions DO ... LOOP, DO WHILE ... LOOP, and DO ... LOOP UNTIL, see EXCEL help!

4.8.5 Python

Pandas (“Python Data Analysis”) is a library for Python, based on NumPy. It is designed for data management and analysis and works with structured data (DataFrame (2-dimensional)) and time series (Series (1-dimensional)), thus mimicking spreadsheet calculations. We use it only in this exercise to read data from an EXCEL book, decode the measurement protocol, and write the results into an EXCEL sheet, in the same form as in Fig. 4.29 (S).

R1C0 in Pandas

In Table 4.17, the EXCEL workbook ‘RBS_data.xlsx’ (Fig. 4.28 (S)) is opened, so that its data are available in Pandas. Without further specification, the workbook must be in the same directory as the Python program. The data in *Sheet1* are entered into a two-dimensional matrix *sh*. This matrix can be addressed in A1 or R1C0 reference style (see second cell). In A1 style, a column is addressed by a letter and a row by a number starting at 1; in R1C0 style, both are addressed by numbers, *with columns being numbered starting at 0*.

With the statement in line 5, the contents in the cells will be copied directly. If a cell contains a formula, this formula will be transferred. If it is desired that all

Table 4.17 **a** (top cell) opening an EXCEL workbook; **b** (bottom cell) addressing cells and ranges and reading their content

<pre> 1 import numpy as np 2 import pandas as pd 3 import openpyxl #A Python library to read/write Excel 2010 4 5 wb=openpyxl.load_workbook('RBS_data.xlsx') 6 sh=wb['Sheet1'] </pre>
<pre> Sh: <Worksheet "Sheet1"> sh['A1'].value: T1.lay sh[1][0].value: T1.lay </pre>
<pre> 7 for r in range(1,4): 8 print(sh[r][0].value) </pre>
<pre> range(1, 6): T1.lay !----- d=0.20E18 None Ti#,1 </pre>

formulas be evaluated and only the results transferred, the opening has to include a keyword argument `data_only = True`:

```
wb = openpyxl.load_workbook('RBS_data.xlsx', data_only = True).
```

In Table 4.18, the sample name is extracted from the first entry in $c = 0$ (column A), reported in the second cell. The content of the bottom cell is produced by the print statements shown explicitly in lines 5 to 9. In contrast to this, print statements

Table 4.18 Extracting the sample name from the first entry in column c ; print statements resulting in the output cell (bottom cell) are explicitly reported in lines 5 to 9

<pre> 1 c=0 2 SN=list(sh[1][c].value)[:4] #Delete ".lay" 3 SampNam=''.join(list(SN)) 4 5 print("sh[1][c] ",sh[1][c]) 6 print("sh[1][c].value ",sh[1][c].value) 7 print("list(sh[1][c].value)",list(sh[1][c].value)) 8 print("SN ",SN) 9 print("SampNam ",SampNam) </pre>
<pre> sh[1][c] <Cell 'Sheet1'.A1> sh[1][c].value T1.lay list(sh[1][c].value) ['T', '1', '.', 'l', 'a', 'y'] SN ['T', '1'] SampNam T1 </pre>

Table 4.19 Getting the last row of useful information; line 5 contains an error

```

1 def LastRow(c):
2     for r1 in range(2,15):
3         cl=sh[r1][c]           #Cell address
4         if cl.value!=None:
5             cll=''.join(list(cl.value)[:5])
6             #Indentation 4 spaces!
7             if cll=='!----': r2=r1-1
8     return r2

```

are usually omitted in our tables reporting Python programs; only the results are usually reported in an output cell.

Questions

What are the values of `sh[1][0].value` and `list(sh[1][0].value)[0]`?⁴⁵

Line 5 of Table 4.19 contains a bug. Which one?⁴⁶

In Table 4.19, column *c* is scanned for the occurrence of “!—”, the code word signaling the end of the information on the first layer.

Table 4.20 reports the function for decoding the string in the cell in column *c*, row *r*₂. The Python code mimics the VBA procedure in Fig. 4.32 (P).

In Table 4.21, a data frame *out* is created reproducing the EXCEL sheet in Fig. 4.29 (S). A FOR loop runs over *sample* with sample data, determining the last row *r*₂ with useful information and decodes the range rows 3 to *r*₂ of the column. The print statement in line 11 produces the output in the lower cell.

Output from Pandas to an Excel file

Writing our results into an EXCEL file requires some care. The simple statement in Table 4.22a creates a new file *RBS_data3.xlsx* and writes our frame *out* into a sheet with the name *Sheet3*. If a file with said name already exists, it is overwritten. The keyword arguments `header = False`, `index = False` cause row 1 and column 1 of Table 4.21 (bottom cell) not to be output.

To write the data into an already existing file, we have to open that file and specify a writer (lines 8 and 9 in Table 4.22).

⁴⁵ `sh[1][0].value -> T1.lay, list(sh[1][0].value)[0] -> T.`

⁴⁶ Indentation with respect to the `if` line is only 3 spaces; it must be 4 spaces.

Table 4.20 Function for decoding a string

```

1  def decode(c,r2):                #Column c, row r2
2      nAtoms=None
3      O=None
4      Ti=None
5      Nb=None
6      Ar=None
7      #Decode information in rows 3 to r2!
8      #First index in array starts with 0
9      for r in range(2,r2+1):
10         cl=sh[r][c]              #Cell address
11         if cl.value!=None:       # != means "not equal"
12             cll=''.join(list(cl.value)[:2])
13             if cll=='d':
14                 nAtoms=''.join(list(cl.value)[2:])
15             cll=''.join(list(cl.value)[3:])
16             if cll=='O#,': O=''.join(list(cl.value)[3:])
17             cll=''.join(list(cl.value)[4:])
18             if cll=='Ti#,': Ti=''.join(list(cl.value)[4:])
19             if cll=='Nb#,': Nb=''.join(list(cl.value)[4:])
20             if cll=='Ar#,': Ar=''.join(list(cl.value)[4:])
21         nAtoms=float(nAtoms)
22     return (nAtoms,Ti,O,Nb,Ar)

```

4.9 User-Defined Functions

We code functions in VISUAL BASIC that can be applied in spreadsheets just like built-in functions. As examples, we realize the vector operations scalar product and cross-product with three-component vectors as arguments and with, respectively, a scalar and a vector as the return variable.

4.9.1 User-Defined Functions as Add-In

Functions in modules

We often call built-in functions in cells, e.g., trigonometric functions with formulas like $B5 = [=B\$1*\cos(B\$2*\$A5)]$. We can also create functions ourselves and apply them in the same way. As an example, we implement a function $CosSq(x)$, calculating the square of a cosine. This has to be done in a module that we create with INSERT/MODULE in the project explorer (see Fig. 4.33a).

The function is implemented in the corresponding Visual Basic sheet MODULE1. The qualifier is FUNCTION, not SUB, as for procedures. For a function, a value must be

Table 4.21 Data frame reproducing the structure of the EXCEL sheet in Fig. 4.29 (S); *NaN* stands for “Not a number”

```

1 out = pd.DataFrame(index=range(1,8), s=list('ABCDEF'))
2 title=['SampNam','NAtoms','Ti','O','Nb','Ar']
3 out.iloc[1]=title
4     #iloc is integer position from 0 to length-1 of the axis
5 for sample in range(4):
6     SampNam=sh[1][sample].value
7     SampNam=SampNam[:-4]      #Remove .lay
8     r2=LastRow(sample)
9     nAtoms,Ti,O,Nb,Ar=decode(sample,r2)
10    result=[SampNam,nAtoms,Ti,O,Nb,Ar]
11    out.iloc[sample+2]=result
12 print(out)
13 out.to_excel("output.xlsx",sheet_name="Sheet2",
14     header=False,index=False)  #A..F and 1..7 not transferred

```

#NaN means "Not a number"						
	A	B	C	D	E	F
1	NaN	NaN	NaN	NaN	NaN	NaN
2	SampNam	NAtoms	Ti	O	Nb	Ar
3	T1	2e+17	1	2.5	0.03	0.01
4	T2	2.5e+17	1	2.4	0.05	None
5	T3	3e+17	1	2.3	0.07	0.008
6	T4	3.5e+17	1	2.2	None	None
7	NaN	NaN	NaN	NaN	NaN	NaN

Table 4.22 **a** (top cell) Creates a new file and writes the data into the specified sheet; **b** (bottom cell) Opens an existing file and adds a new sheet

```

14 out.to_excel('RBS_data3.xlsx',sheet_name="Sheet3",
15     header=False,index=False)
16 #Creates a new workbook with one sheet "Sheet3".
17 #Overwrites 'RBS_data3.xlsx' if it exists already.
18 """"https://stackoverflow.com/questions/20219254/
19     how-to-write-to-an-existing-excel-file-
20     without-overwriting-data-using-pandas""""
21 wb2 = openpyxl.load_workbook('RBS_data2.xlsx')
22 writer = pd.ExcelWriter('RBS_data2.xlsx',
23     engine='openpyxl')
24 writer.book = wb2 #Necessary for not deleting other sheets
25 out.to_excel(writer, "data3",header=False,index=False)
26 writer.save()

```

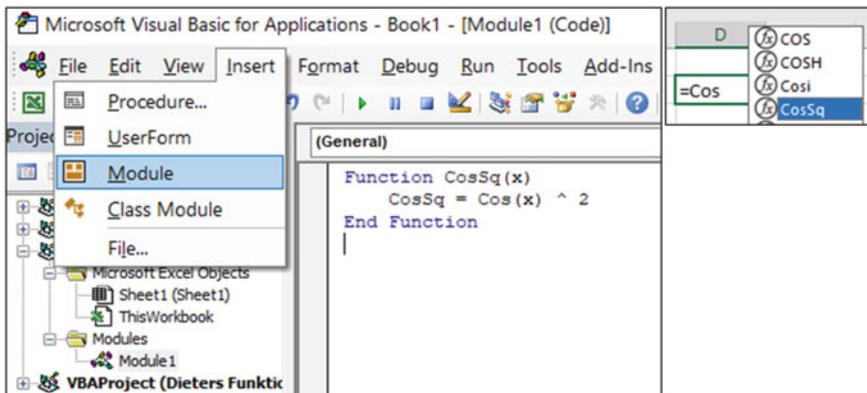


Fig. 4.33 a (left) PROJECT-EXPLORER window; the user-defined spreadsheet function *CosSq* is in MODULE1. b (right) *CosSq* pops up after typing “=Cos” among the other functions starting with *cos*

assigned to the function identifier within the function’s body, in our case $cosSq = \cos(x)^2$. When now “=Cos” is written into a cell, a list pops up with all functions starting with *Cos*, including our *CosSq* (see Fig. 4.33b).

If you want to use your functions in every EXCEL file, they must be saved as an ADD-IN. To do so, create an EXCEL file, enter your function codes into VBA modules, and finish with: SAVE AS/EXCEL ADD-IN. This add-in must be activated in the EXCEL options. Upon selecting FILE/OPTIONS/ADD-INS/ a list appears with an entry “ Dieter’s Functions” that must be included by ticking the box . In the VBA editor, VBA PROJECT *Dieters Funktionen.xlam* now also appears in the project explorer under PROJECT-VBA PROJECT (see Fig. 4.33a, bottom line).

4.9.2 Scalar Product and Vector Product

We are developing functions for the scalar and vector products of two three-dimensional vectors stored in cell ranges that are entered as arguments in the functions. Let’s consider two three-dimensional vectors:

$$\underline{r}_1 = (x_1, y_1, z_1) \text{ and } \underline{r}_2 = (x_2, y_2, z_2)$$

Their scalar product is defined as.

$$\underline{r}_1 \cdot \underline{r}_2 = x_1 \cdot x_2 + y_1 \cdot y_2 + z_1 \cdot z_2 \tag{4.2}$$

Their vector product (or cross product) is defined as.

$$\underline{r}_1 \times \underline{r}_2 = (y_1 \cdot z_2 - y_2 \cdot z_1, z_1 \cdot x_2 - z_2 \cdot x_1, x_1 \cdot y_2 - x_2 \cdot y_1) \tag{4.3}$$

1	Function Scl(r1 As Range, r2 As Range)	Function Crsm(r1 As Range, r2 As Range)	12
2	Scl = r1(1) * r2(1) + r1(2) * r2(2) + r1(3) * r2(3)	Dim cs(2, 2)	13
3	End Function	cs(0, 0) = r1(2) * r2(3) - r1(3) * r2(2)	14
4		cs(1, 0) = r1(3) * r2(1) - r1(1) * r2(3)	15
5	Function Crs(r1 As Range, r2 As Range)	cs(2, 0) = r1(1) * r2(2) - r1(2) * r2(1)	16
6	Dim cs(2)	cs(0, 1) = cs(1, 0)	17
7	cs(0) = r1(2) * r2(3) - r1(3) * r2(2)	cs(0, 2) = cs(2, 0)	18
8	cs(1) = r1(3) * r2(1) - r1(1) * r2(3)	Crs = cs	19
9	cs(2) = r1(1) * r2(2) - r1(2) * r2(1)	End Function	20
10	Crs = cs		21
11	End Function		22

Fig. 4.34 (P) User-defined functions for the scalar product *Scl* and the vector product *Crs* of two three-dimensional vectors r_1 and r_2 ; the function *Crsm* can process and output both column and row vectors

The output of the scalar product is one number returned into the cell with the corresponding formula; that of the vector product is a set of three components to be entered into a row range or a column range.

These two products are calculated with the two user-defined spreadsheet functions *Scl* and *Crs* in Fig. 4.34 (P).

Questions

How many components does the array *cs*(2) in Fig. 4.34 (P) have?⁴⁷

What are the differences between the arrays named *cs* in the functions *Crs* and *Crsm* in Fig. 4.34 (P)?⁴⁸

Scalar product

The scalar product is easy to program. It can be calculated in one code line (line 2 in Fig. 4.34 (P)). Two three-dimensional cell ranges must be entered as arguments. These can both be column ranges or both row ranges or one column range and one row range (see Fig. 4.35 (S)). Consequently, the variables in the function header are declared as RANGE.

Vector product

The result of a vector product is again a vector. In Fig. 4.36 (S), two column vectors \underline{a} and \underline{b} are defined in range A2:B4. The row vectors \underline{c} and \underline{d} in range B6:D7 contain the same coefficients as \underline{a} and \underline{b} . In column D, the cross-product $\underline{a} \times \underline{b}$ is calculated with spreadsheet formulas.

⁴⁷ The array DIM *cs*(2) has the three components *cs*(0), *cs*(1), *cs*(2).

⁴⁸ In *Crs*, a one-dimensional array (type *cs*(2)) is written, in *Crsm*, a two-dimensional array (type *cs*(2,2)).

	A	B	C	D	E	F	G	H	I	J	K	L
1	a	b										=ScI(A2:A4;B2:B4)
2	1	2		c_	1	2	3					=ScI(A2:A4;E3:G3)
3	2	2		d	2	2	2					=ScI(a;b)
4	3	2										=ScI(c_;d)
5												=ScI(a;d)

Fig. 4.35 (S) Contains the results of the user-defined spreadsheet function *ScI*, which calculates the scalar product of two three-dimensional vectors

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	a	b	a x b				=crs(a;b)	=crs(c_;d)	=crs(c_;b)			=crsm(a;b)	=crsm(c_;d)	=crsm(c_;b)	
2	3,0	-2,0	-10,0 =A3*B4-B3*A4				-10,0	-10,0	-10,0			-10,0	-10,0	-10,0	
3	2,0	8,0	-20,0 =A4*B2-B4*A2				-10,0	-10,0	-10,0			-20,0	-20,0	-20,0	
4	2,5	5,0	28,0 =A2*B3-B2*A3				-10,0	-10,0	-10,0			28,0	28,0	28,0	
5															
6	c_	3,0	2,0	2,5			-10,0	-20,0	28,0 =crs(a;b)			-10,0	-20,0	28,0 =crsm(a;b)	
7	d	-2,0	8,0	5,0			-10,0	-20,0	28,0 =crs(c_;d)			-10,0	-20,0	28,0 =crsm(c_;d)	
8							-10,0	-20,0	28,0 =crs(a;d)			-10,0	-20,0	28,0 =crsm(a;d)	

Fig. 4.36 (S) The vector product $a \times b$ is calculated in column D using spreadsheet formulas; the range G2:I8 contains results of the user-defined function *crs*, which can only output row vectors (wrong results in G2:I4); the range L2:N8 contains results of the user-defined spreadsheet function *crsm*, which can accept and output row and column vectors

In columns G to I, the function *Crs* is used. As you can see from the results, this function accepts row and column vectors as input, but only returns correct values if they are output as row vectors (see G6:I8); the results in G2:I4 are wrong.

In columns L to N, the function *Crsm* from Fig. 4.34 (P) is applied, which can output the result either as a row vector (e.g., L6:N6) or as a column vector (e.g., L2:L4). This is because, in this function, a 3×3 matrix is written into the range declared with `DIM cs(2,2)`, of which only one row or one column is output if only one row range or one column range is activated.

The functions *Crs* and *Crsm* must be called as matrix functions. In Fig. 4.36 (S), for example, the area G6:I6 was activated, the formula entered according to J6 and closed with the magic chord Ψ (*Ctl* + *Shift*) + *Enter*. In L2:L4, a column area was activated, and in L6:N6, a row area, so that in each case, vectors with three components are returned by *crsm*.

Questions

Why is `cs(2,2)` in FUNCTION *Crsm* sufficient as an array for a 3×3 matrix?⁴⁹

⁴⁹ `DIM cs(2,2)` is a (0, 1, 2) x (0, 1, 2)-Matrix. The indices begin at 0.

Table 4.23 Two column vectors *a*, *b* and two row vectors *c*, *d* are specified

<pre> 1 a=np.array([[1,2,3])).transpose(1,0) 2 b=np.array([[2,2,2])).transpose(1,0) 3 c=np.array([1,2,3]) 4 d=np.array([2,2,2]) </pre>		
<p>a</p> <pre> [[1] [2] [3]] </pre>	<p>b</p> <pre> [[2] [2] [2]] </pre>	<p>c</p> <pre> [1 2 3] </pre> <p>d</p> <pre> [2 2 2] </pre>

Table 4.24 Scalar product, line 8 deactivated

<pre> 5 def Scl(r1,r2): 6 o=r1[0]*r2[0]+r1[1]*r2[1]+r1[2]*r2[2] 7 out=o 8 #if type(o) == np.ndarray: out = o[0] 9 return out </pre>	
<pre> Scl(a,b) Scl(c,d) Scl(a,d) </pre>	<pre> [12] 12 [12] </pre>

How can this function be used to output row and column vectors to a spreadsheet?⁵⁰

4.9.3 Python

Scalar product

The specifications in Table 4.23 for column vectors *a*, *b*, and row vectors *c*, *d* are the same as in Fig. 4.35 (S).

The function *Scl* (for “scalar product”) as reported in Table 4.24 corresponds literally to the Visual Basic function of the same name (Fig. 4.34 (P)). Its output is, however, only a scalar if two row vectors are multiplied.

Column vectors are two-dimensional arrays; a scalar product with one of them is broadcast into a one-dimensional array (see lines 10 and 12 of Table 4.24). If a scalar is always desired, line 8 has to be activated by removing the # character.

Vector product or cross product

A function *Crsm* for calculating the cross-product of two three-dimensional vectors is reported in Table 4.25.

⁵⁰ Because internally a 3 × 3-matrix is created, see explanations for Fig. 4.36 (S)!

Table 4.25 Cross product of two three-dimensional vectors, output optionally transposed in line 16 to become a column vector

```

1 def Crsm(r1, r2, C):
2     cs=np.empty(3)
3     cs[0]=r1[1]*r2[2]-r1[2]*r2[1]
4     cs[1]=r1[2]*r2[0]-r1[0]*r2[2]
5     cs[2]=r1[0]*r2[1]-r1[1]*r2[0]
6     if C==True:cs=np.array([cs]).transpose(1,0)
7     return cs

```

Table 4.26 Cross-product of row and column vectors

<pre> 1 a=np.array([[3,2,2.5]]).transpose(1,0) 2 b=np.array([[-2,8,5]]).transpose(1,0) 3 c=np.array([3,2,2.5]) 4 d=np.array([-2,8,5]) </pre>			
Crsm(a,b)	[-2.00 4.00 -2.00]	Crsm(c,b,True)	[[-2.00] [4.00] [-2.00]]
Crsm(c,d)	[-2.00 4.00 -2.00]		

In Table 4.26, again, two column vectors a , b and two row vectors c , d are specified. Their pairwise cross-product, obtained with *Crsm*, is reported in the lower cells of the table. If only the vectors are transferred to the function, row vectors are returned (bottom left cell), whereas a column vector is returned if the optional parameter C is assigned “True” (bottom right cell).

4.10 Questions and Tasks

Densely packed planes

1. How many neighbors does a sphere have in a closely-packed plane?
2. How many neighbors does a sphere have in a stack of closely-packed planes?

Program-controlled drawings

3. Write a macro that writes the numbers 1 to 20 in a diagonal of a table, e.g., in cells A1, B2, etc.
4. What does the broom rule Ψ *Empty lines separate curves* mean?

Record macro

The diagram in Fig. 4.37 (S) has been created with the macro recorder switched on. The program code can be found in Fig. 4.38 (P). The diagram has been formatted

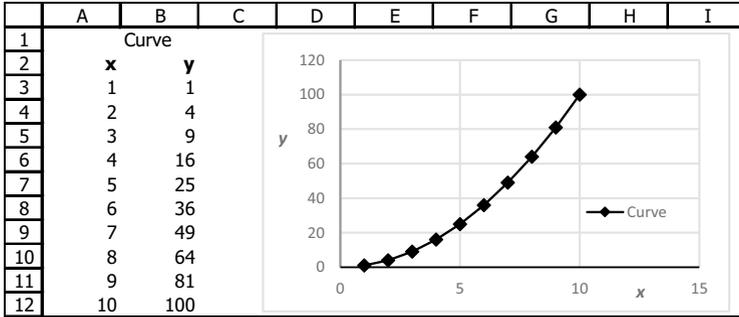


Fig. 4.37 (S) Diagram of the data in columns A and B

```

1 Sub Macro1()
2 ActiveSheet.Shapes.AddChart2(240, xlXYScatterLines).Select
3 ActiveChart.SeriesCollection.NewSeries
4 ActiveChart.FullSeriesCollection(1).Name = "=Diagram!$B$1"
5 ActiveChart.FullSeriesCollection(1).XValues = "=Diagram!$A$3:$A$12"
6 ActiveChart.FullSeriesCollection(1).Values = "=Diagram!$B$3:$B$12"
7 End Sub
    
```

Fig. 4.38 (P) Instructions recorded by the macro recorder when the diagram in Fig. 4.37 (S) was created

with the programs in Fig. 4.38 (P) and Fig. 4.40 (P). Your task is to analyze the VISUAL BASIC programs and redraw the diagram with our standard FigStd and plt (matplotlib.pyplot) commands in Python.

Plotting a diagram

- Of what type is the diagram in Fig. 4.37 (S) (LINE, BAR, or SCATTER)?
- What is the equation for y ?
- How do you create the arrays x and y in Python?
- With SUB MACRO1 in Fig. 4.38 (P), retrace how the chart was created and interpret the program lines 3 to 6!
- (Python) What does a header in FigStd (numpy and matplotlib) look like when leading to a diagram like that in Fig. 4.37 (S)?
- SUB MACRO2 in Fig. 4.39 (P) has recorded the instructions executed to format the data series in the diagram of Fig. 4.37 (S). Interpret the formatting instructions! How do you implement them in the plot command of plt.plot(?) of the plotlib library?
- SUB MACRO3 in Fig. 4.40 (P) has recorded the commands executed to format the x -axis in the diagram of Fig. 4.37 (S). Interpret the instructions that follow the two WITH SELECTION commands!

8	Sub Macro2()	.ForeColor.ObjectThemeColor _	13
9	ActiveChart.FullSeriesCollection(1).Select	= msoThemeColorText1	14
10	Selection.MarkerStyle = 2	.Weight = 1.25	15
11	Selection.MarkerSize = 7	End With	16
12	With Selection.Format.Line	End Sub	17

Fig. 4.39 (P) SUB MAKRO2 contains the commands that were recorded by the macro recorder when the data series for the diagram in Fig. 4.37 (S) was formatted

18	Sub Macro3()	ActiveChart.Axes(xlValue).Select	25
19	ActiveSheet.ChartObjects("Chart 1").Activate	With Selection.Format.Line	26
20	ActiveChart.Axes(xlCategory).Select	.ForeColor.ObjectThemeColor \	27
21	With Selection.Format.Line	'= msoThemeColorBackground1	28
22	.ForeColor.ObjectThemeColor \	.ForeColor.Brightness = -0.5	29
23	'= msoThemeColorText1	End With	30
24	.Weight = 1#	End Sub	31

Fig. 4.40 (P) SUB MACRO3 contains commands recorded by the macro recorder when the diagram in Fig. 4.37 (S) was formatted

12. Write a Python program that produces a similar diagram with FigStd (numpy and matplotlib) ! It should include formatting the data series in the function header and changing the thickness of the x-axis within FigStd.

Text processing in VISUAL BASIC and Python

The following three questions refer to both Visual Basic for EXCEL and Python. The variable Tx contains the text “We are cutting.”

- 13. With which instructions do you get the first, the last, and the 4th letters of Tx?
- 14. How do you copy the fragment “re cu” from Tx to a new variable Wd?
- 15. Of what type are the variables A, B, C, and D in the commands $A = Split(B)$ and $C = Join(D)$?

Ψ Loop2i; continue counting in the loop!

- 16. Write a macro SUB XY1() that writes all products x.y from x = 1 to 10 and from y = 1 to 5 successively into a spreadsheet, with x and y being integers!
- 17. Do the same in another macro SUB XY2() for x and y being half-integers (1, 3/2, 2, 5/2, ...)!
- 18. Do the same as in SUB XY2() in a new macro SUB XY3(), but insert a blank line after every third entry into the spreadsheet!
- 19. Create a similar Python program using the .append method and nested loops running over arrays x and y created by np.linspace! For the equivalent of inserting an empty row into a spreadsheet, append 'None' to the list!
- 20. Make a hand-drawn sketch of straight-line segments in the xy plane, with the x values in line 1 and the y values in line 2 of Fig. 4.41 (S)!

	A	B	C	D	E	F	G	H	I	J	K	L
1	x	0,5	1,0		1,5	2,0		2,0	1,5		1,0	0,5
2	y	1,5	2,0		2,0	1,5		1,0	0,5		0,5	1,0

Fig. 4.41 (S) Coordinates for straight-line segments in the *xy* plane

Rep-log procedure

In Fig. 4.42a, a circle is represented, calculated in the spreadsheet in Fig. 4.43 (S) in columns G:I. The coordinates of the centers of these circles are taken from the table in Fig. 4.41 (S). Figure 4.42b shows four circles whose coordinates have been obtained from the table in Fig. 4.43 (S) with a rep-log procedure that systematically changes the center point and the radius.

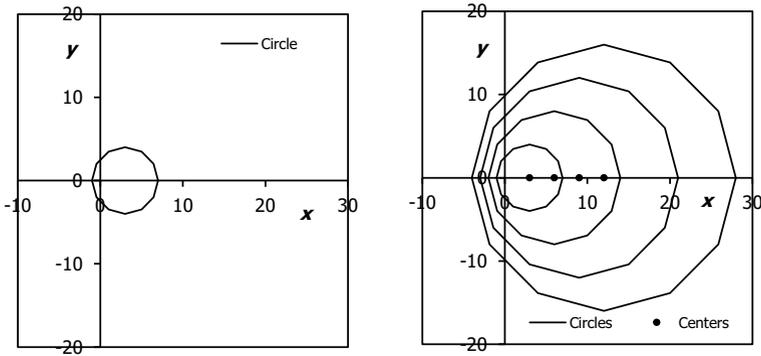


Fig. 4.42 a (left) A circle with $r_0 = 4$ and $x_0 = 3$. b (right) The circle from a was enlarged three times and shifted along the *x*-axis; in the diagram, all four circles are represented as one data series

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	r.0	4.00		c.s	4.00										
2	x.0	3.00		v.F	3.00										
							Circle			Circles				Centers	
3															
4		30 °													
5	dphi	0.524													
6							phi	x	y		x	y		xC	yC
7							0.00	7.00	0.00		7.00	0.00		3	0
8							0.52	6.46	2.00		6.46	2.00		6	0
9							6.28	7.00	0.00		7.00	0.00			

Fig. 4.43 (S) Spreadsheet calculation for Fig. 4.42. The variable c_s contains the factors with which the radius of the circle is increased. The variable v_F is the velocity with which the circle center is shifted on the *x*-axis

21. What are the radii and the coordinates of the circles' center points in Fig. 4.42b?
22. The coordinates of the circle are to be calculated four times by changing the parameters in the table and then to be stored successively in columns Q and R (column indices 17 and 18, respectively) of the spreadsheet and graphically displayed in a diagram as one data series (to yield Fig. 4.42b). The centers' coordinates are in columns N and O (column indices 14 and 15, respectively). The circle radius should increase with $r_0 = v_s \cdot t$, and the center point should be shifted on the x -axis with $x_0 = v_F \cdot t$. Write such a log procedure in Visual Basic!
23. For the answer of Question 23 in Python, create empty lists Q and R and extend them with the `.append` method in a nested loop! Instead of an empty line, insert "None"! Finally, plot (Q, R) with a correct label and observe whether four separated circles show up!

Formula-generating routine

In Fig. 4.44b, you see a parabola connected to the horizontal axis by vertical lines. The spreadsheet calculation for the coordinates is shown in Fig. 4.44a (S).

24. The y -value of the parabola is calculated in the usual way with named cell ranges (see the formula in B5). Every tenth point of the parabola is connected to the horizontal axis with a vertical line ("Dashes"). Write a routine that writes the formulas for the vertical lines' coordinates into columns D and E!
25. Create a corresponding Python program with two variants. (a) The vertical lines are plotted one after the other in a loop. (b) The coordinates of all vertical lines are stored in lists named D and E separated by empty cells, and plotted as one data series.

	A	B	C	D	E
1	a.1	2.00		c.1	4.00
2	b.1	4.00		dx	0.10
3					
4		$y=2(x-4)^2+4$			Dashes
5		$=a.1*(x-b.1)^2+c.1$			
6	x	y	x.s	y.s	
7	0.00	36.00	0.00	36.00	
8	0.10	34.42	0.00	0.00	
9	0.20	32.88			
10	0.30	31.38	1.00	22.00	
11	0.40	29.92	1.00	0.00	
12	0.50	28.50			
13	0.60	27.12	2.00	12.00	
107	10.00	76.00			

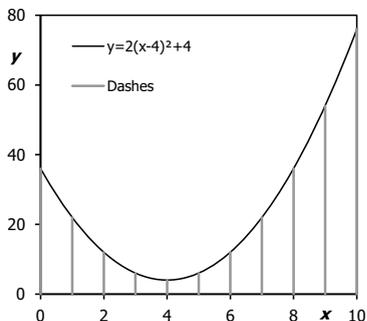


Fig. 4.44 a (left, S) Polynomial $y = a_1(x - b_1)^2 + c_1$ in columns A and B; coordinates x_s, y_s for the vertical lines of Fig. b in columns D and E. b (right) Display of the data from a

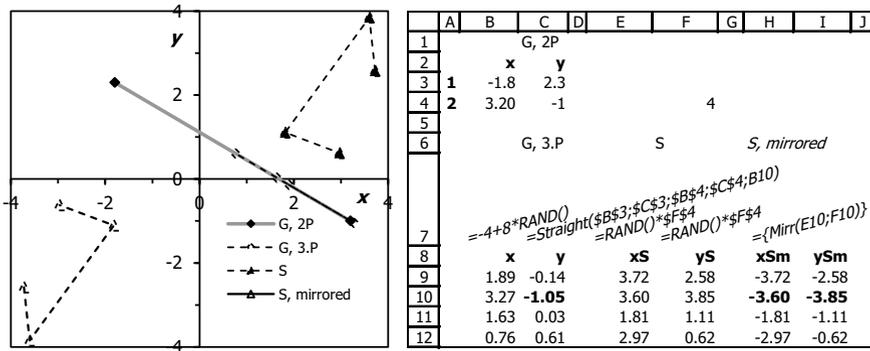


Fig. 4.45 a (left) Two points “G, 2P” define a straight line (18a); the points in the third quadrant are mirrored at the zero point into the first quadrant (18b). **b** (right, S) Spreadsheet calculation for **a**; the coordinates for the points on the straight line are in columns B and C; the coordinates in columns H and I are the mirror images of the coordinates in columns E and F

User-defined spreadsheet functions

In Fig. 4.45a, the results of two user-defined spreadsheet functions are displayed. One function adds additional points, “G, 3.P”, onto a straight line “G, 2P” defined by two points. The other function mirrors points in the first quadrant at (0, 0) into the third quadrant. The coordinates of the points in Fig. 4.45a are calculated in the spreadsheet of Fig. 4.45b (S).

- 26. What are the names of the two user-defined functions reported in line 7 in Fig. 4.45b (S)?
- 27. Write a function (VISUAL BASIC or Python) of the type $y_3 = f(x_1, y_1, \dots)$ that calculates the y -value y_3 of a third point from the coordinates of the two defining points of a straight line and the x -value x_3 of the third point!
- 28. Write a function (VISUAL BASIC or Python) of the type $(x_{sp}, y_{sp}) = f(x, y)$ that mirrors the coordinates x and y of a point at the origin of the coordinate system!

Macros

- 29. You want to trigger a macro whenever a slider named SCROLLBAR1 is changed. What is the name of the associated macro?⁵¹
- 30. At a mail-order company, some data from all outgoing packets are entered in the spreadsheet of Fig. 4.46 (S). Write a protocol routine (Visual Basic or Python) that reads some data from Fig. 4.46 (S) and enters it into a table as in Fig. 4.47 (S)! The packets have to be numbered consecutively.

⁵¹ SUB SCROLLBAR1_CHANGE().

	A	B	C	D	E	F	G	H	I
1	Name		Mary B.						
2	Running number		46						
3									Protoc
4	Width (cm)	w	5	Volume (l)	V	1.80	$=w*l*h/1000$		
5	Length (cm)	l	18	Surface (m ²)	S	0.11	$=(w*l+w*h+l*h)*2/10^4$		
6	Hight(cm)	h	20	Time		07.09.2020 17:21	$=NOW()$		

Fig. 4.46 (S) Table section in which the width, length, and height of packages, as well as the sender's name, are to be entered

	K	L	M	N	O	P	Q	R	S
6	12 next free row								
7	Number	Name	Time	Width/cm	Length/cm	Height/cm	Volume/l	Surface/m²	
8	43	Otto L.	7.9.20 17:03	17	17	14	4.05	0.15	
9	44	James L.	7.9.20 17:16	20	10	10	2.00	0.10	
10	45	Henry M.	7.9.20 17:16	18	28	8	4.03	0.17	
11	46	Mary B.	7.9.20 17:21	5	18	20	1.80	0.11	

Fig. 4.47 (S) The data from Fig. 4.46 (S) are to be reorganized in this way

With the methods learned in Chaps. 2–4 (list processing, programming constructs for, if, sub/def), we practice differentiation, integration, and calculating with vectors. We get to know a new technique: solving systems of linear equations with matrix calculation.

5.1 Introduction: Calculus, Vectors, and Linear Algebra

Solutions of Exercises 5.3 (Excel), 5.5 (Python), 5.6 (Python), and 5.7 (Excel) can be found at the internet address: go.sn.pub/VYYbJL.

Straight-line segment

Straight-line segments are central for vector calculation, and also for calculus, because we approximate all curves by sequences of such elements. In the introductory Exercise 5.2, we calculate vector entities related to a straight-line segment, e.g., line vector and mid-perpendicular, as well as length, slope, and area enclosed with the x -axis.

Differentiation and integration

We get to know simple techniques with which the first and second derivatives of a function can be obtained (Exercise 5.3) and with which a function can be integrated. With integration, the area between the curve of a function and the x -axis (Sect. 5.4.1) and the length of a curve (Sect. 5.4.2) can be calculated. We pay special attention to the x -value over which the results must be plotted: at the beginning, in the middle, or at the end of the interval for which the elementary calculation was done.

Vectors in the plane

We discuss the addition and the scalar product of two vectors and convert planar Cartesian and planar polar coordinates one into the other (Exercise 5.5). In Exercise 5.6, the tangents to and the perpendiculars on a polynomial are drawn. As a physically relevant example, we calculate the forces acting on a car running through a banked curve (Exercise 5.7). In Exercise 5.8, we calculate a mobile's equilibrium by using the mathematical construction of a weighted average.

Systems of linear equations

Systems of linear equations (with small rank) are solved by bringing them into matrix form and forming the inverse matrix of the coefficient matrix (Exercise 5.9). With this method, we get the coefficients of a polynomial from a given set of points on the curve and determine currents in electrical networks with Kirchhoff's rules. In Python, we also use functions from the `numpy.linalg` library for these tasks.

Mathematical functions

In Sect. 5.10, some useful mathematical functions are listed, both in EXCEL and numpy notation, together with short descriptions.

5.2 Straight-Line Segment Under a Magnifying Glass

A straight-line segment is specified by its two endpoints. Its center, line vector, and perpendicular vector are obtained with vector operations. The basic pieces of calculus, dx , dy , dA (the area under the curve), are calculated.

5.2.1 Under a Magnifying Glass

The functions we are investigating in this textbook are all continuous. They are approximated as polylines, i.e., as sequences of straight-line segments. Therefore, all operations of calculus, such as differentiation, integration, and integration along a line, are based on such properties of segments as length, slope, line vector, and mid-perpendicular.

In this exercise, we specify a straight segment in a plane by its two endpoints A , B formulated as positional vectors. The unit line vector \underline{AB} of the segment and its perpendicular \underline{AB}_\perp to be erected at the midpoint \underline{AB}_C are calculated with matrix operations. An example is shown in Fig. 5.1.

The equation of the line passing along the segment is

$$y = y_A + m \cdot (x - x_A) \tag{5.1}$$

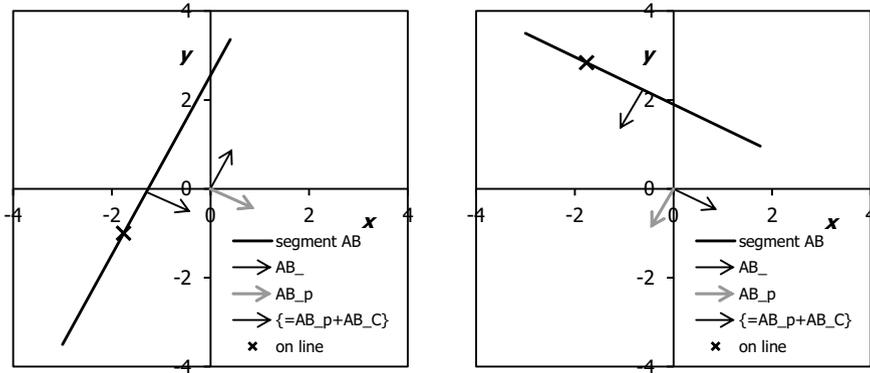


Fig. 5.1 Line vector and mid-perpendicular of a straight-line segment. The cross is drawn with a function based on the coordinates of the two endpoints, **a** (left) with the settings from Fig. 5.2 (S), **b** (right) other settings

with m being the slope of the segment and (x_A, y_A) the coordinates of point A . In Fig. 5.1a, the point (y_s, x_s) on this line for $x_s = -1.76$ is represented by a cross. We use vector calculation, e.g., $\underline{P} \cdot \underline{AB}$ to obtain the unit vector perpendicular to the segment with \underline{P} , the 90° rotation matrix, and \underline{AB} , the unit line vector.

5.2.2 Data Structure and Nomenclature

A, B	the two endpoints of a straight segment in the plane
x_A, y_A	coordinates of point A
x_B, y_B	coordinates of point B
$dx = (x_B - x_A)$	their distance in the x-direction
$dy = (y_B - y_A)$	their distance in the y-direction
$ds = \sqrt{dx^2 + dy^2}$	length of the segment
m	slope dy/dx
$y = y_A + m \cdot (x - x_A)$	function describing the line through the segment
$\Delta A = (y_A + y_B)/2 \cdot dx$	area between the segment and the x-axis
$\underline{A} = (x_A, y_A)$	vector representation of point A
$\underline{B} = (x_B, y_B)$	vector representation of point B
$\underline{AB_C} = (\underline{A} + \underline{B})/2$	center of the segment
$\underline{AB} = (\underline{B} - \underline{A})/ds$	unit line vector
$\underline{P} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$	90° - rotation matrix.
$\underline{AB_p} = \underline{P} \cdot \underline{AB}$	unit vector perpendicular to the segment.

5.2.3 Spreadsheet Calculation

In the spreadsheet calculation of Fig. 5.2 (S), the coordinates of the starting point A are determined with sliders in A2:B3, ranging from 0 to 100. The values in the linked cells are transformed into coordinates ranging from -4 to 4 (G2:G3). The coordinates of the endpoint B are typed directly into cells J2:J3.

The cells G2:G3 get multiple identifiers: x_A and y_A refer to single cells, whereas A refers to the whole range and can be processed as a column vector. The same applies to B .

The length ds of the segment is calculated in A6 from the coefficients of A and B . It can also be calculated with the matrix formula $\{=SQRT(SUMXMY2(B:A))\}$. $SUMXMY2$ stands for “Sum of all individual $(x - y)^2$ ”. The innermost operation $A - B$ involves two matrices, but the output is only a scalar. So, we have to enclose the formula in curly brackets and finish with the magic chord Ψ *Ctrl* + *Shift* + *Return*.

The result of this calculation is shown in Fig. 5.1a. The perpendicular unit vector AB_p is also drawn from the center AB_C of the line to $AB_C + AB_p$ (J6:J7).

In Fig. 5.3 (S), we calculate the primary segments of calculus dx , dy , and the area dA between the segment and the x -axis. Furthermore, we set up the equation

	A	B	C	D	E	F	G	H	I	J	K
1							A			B	
2	<input type="text" value="55.00"/>	<input type="text" value="92.00"/>				x_A	0.40	=(D2-50)/12.5	x_B	-3.00	
3	<input type="text" value="92.00"/>	<input type="text" value="92.00"/>				y_A	3.36	=(D3-50)/12.5	y_B	-3.50	
4	=SQRT((yA-yB)^2+(xB-xA)^2)										
5	ds										
6	7.66		0.00	0.44		0.00	0.90	=D7		-1.30	-0.40
7			0.00	0.90		0.00	-0.44	=D6		-0.07	-0.51
8											

Fig. 5.2 (S) Coordinates of points A and B , line vector AB and center position vector AB_C ; the perpendicular vector AB_p is obtained from AB through matrix multiplication with P , the 90° rotational matrix P presented in Fig. 5.3 (S)

	L	M	N	O	P	Q	R	S	T
5	area	dA	0.24	=(yA+yB)/2*dx		P			
6		dx	-3.40	=xB-xA		0	-1		
7		dy	-6.86	=yB-yA		1	0		
8	slope	m	2.02	=dy/dx					
9									
10	on line	x	-1.76		28	<input type="text" value="28"/>			
11		y	-1.00	=yA+m*(x-xA)					

Fig. 5.3 (S) Continuation of Fig. 5.2 (S). Characteristics of a straight-line segment important for calculus, and the equation for the line running along the segment

$y = y_A + m \cdot (x - x_A)$ of the line along the segment. The value of x is determined with a slider so that the cross in Fig. 5.1 runs along this line.

5.2.4 Plotting Vectors with Python Matplotlib

To plot vectors with Python, we have to use the function `arrow` of the library `matplotlib.pyplot`, to be imported with `import matplotlib.pyplot as plt`. We integrate this function into a user-defined function *ArrowP* with standard formatting parameters (see Table 5.1).

In EXCEL, arrowheads are a design feature of a line: ... LINE/END ARROW TYPE.

The constructor arguments of `plt.arrow` comprise, among others, the keyword arguments:

<code>width</code>	float (default: 0.001) width of full arrow tail
<code>fill</code>	bool
<code>linestyle</code> or <code>ls</code>	{ '-', '—', '-.', ':', '...', (offset, on-off-seq), ... }
<code>linewidth</code> or <code>lw</code>	float or None
<code>head_length</code>	float or None (default: 1.5 * <code>head_width</code>) length of arrow head
<code>overhang</code>	float (default: 0, triangular) fraction that the arrow is swept back

Some arrows are plotted in Fig. 5.4 to demonstrate the effect of the constructors. The arrow pointing upwards is drawn with our standard function *ArrowP* (`overhang = 1`) in Table 5.1.

5.3 Differentiation

We learn how to approximate the first and second derivatives of a function $f(x)$ numerically with difference quotients between neighboring calculation

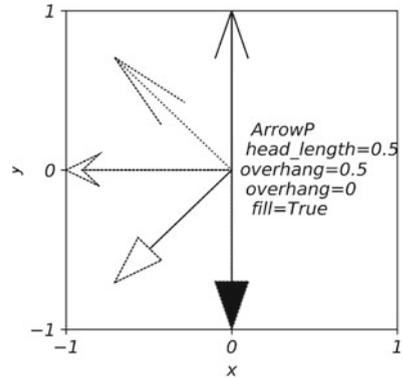
Table 5.1 User-defined function *ArrowP* for drawing an arrow from point P_0 to point P_1 in a plane; the argument list of `plt.arrow` does not contain all possible keyword arguments (similar to Tables 5.3, 5.4, 5.5 and 5.6)

```

1 def ArrowP(P0,P1,c="k",ls='-',lw=1,hw=0.2):
2     (x0,y0)=P0 #Decomposes the foot position vector
3     (x1,y1)=P1 #Decomposes the tip position vector
4     print(lw,hw)
5     #c has to be given as c="k", not c='k'
6     plt.arrow(x0,y0,x1-x0,y1-y0,
7               length_includes_head=True,
8               head_width=hw,overhang=1,fill=False,
9               linestyle=ls, color=c, linewidth=lw)

```

Fig. 5.4 Arrows plotted with different constructor arguments in the procedure head of `plt.arrow`



points. The first derivative must be plotted over the middle between two grid points, the second derivative over the central of three grid points.

5.3.1 First and Second Derivative

The first derivative of a curve is the slope of its tangent to the curve at the specified x -value. The derivative of a function $f(x)$ with respect to x is defined as

$$\frac{df(x)}{dx} = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \quad (5.2)$$

More specifically, Eq. 5.2 is called the right derivative.

Such an approach to the limit cannot be carried out for discrete functions; they are specified as a list of x -values ($x = [\dots x_i \dots]$) and y -values ($y = [\dots y_i \dots]$). Instead, we calculate the difference quotient of neighboring grid points i and $i + 1$:

$$\frac{df\left(\frac{x_i + x_{i+1}}{2}\right)}{dx} \approx \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \quad (5.3)$$

For our discrete functions, the first derivative in the center of a segment is approximated by $\Delta y / \Delta x$, the slope of the segment between adjacent points. The values are to be plotted over the center of the interval. It is evident that the smaller Δx is, the better the accuracy of the approximation.

The second derivative of a function is the derivative of the first derivative, thus,

$$\frac{d^2 f(x)}{dx^2} = \frac{d}{dx} \left(\frac{d}{dx} f(x) \right) \tag{5.4}$$

which is equivalent to applying Eq. 5.2 twice. The second derivative describes the change of the slope, and is thus a measure of the curvature of the curve. The difference equation for grid points x_i at equal intervals Δx is

$$\begin{aligned} \frac{d^2 f(x)}{dx^2} &\approx \frac{1}{\Delta x} \left(\frac{f(x_{i+1}) - f(x_i)}{\Delta x} - \frac{f(x_i) - f(x_{i-1}))}{\Delta x} \right) \\ &= \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1}))}{\Delta x^2} \end{aligned} \tag{5.5}$$

The second derivative at point x can be calculated directly without a detour via the first derivative with function values at the grid points x , $x - dx$ and $x + dx$. It must be plotted over point x (x_i in Eq. 5.5), the coordinate of the middle grid point.

Sine function

To give an example, we differentiate the sine function, knowing beforehand that its first and second derivatives are the cosine and the negative sine, respectively. So, we can check whether our numerical calculations reproduce this result. This is indeed confirmed in Fig. 5.5, with 100 calculation points in one period 2π . The shape of a cosine is clearly visible in the numerically calculated y_{1d} in Fig. 5.5a, and that of a negative sine in y_{2d} in Fig. 5.5b.

Oscillation of a mass-spring-system

Consider the oscillation $z(t)$ of a mass-spring system. The second derivative \ddot{z} with respect to time is the acceleration a , which, in turn, is proportional to the restoring

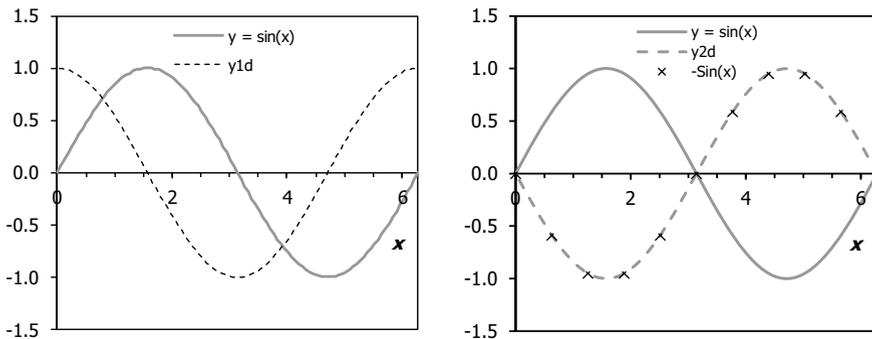


Fig. 5.5 Derivatives of a sine function **a** (left) first derivative, **b** (right) second derivative, numerically (dashed line) and theoretically (x) calculated, $dx = 2\pi/100$

force $-kz$ of the spring. The curvature of the displacement is thus proportional to the force. This leads to the simple equation of a harmonic oscillator based on Newton's law:

$$F = -k \cdot z = m \cdot a = m \cdot \ddot{z}$$

or

$$\ddot{z} = -\left(\frac{k}{m}\right) \cdot z \quad (5.6)$$

The curvature of the displacement is thus proportional but opposed to the force. A sine function is a solution to this differential equation, as can be seen in Fig. 5.5b.

Question

Let $y = \sin(x)$ in Fig. 5.5 be the displacement of an oscillator. At which positions x is the speed of the oscillator maximum, and at which positions is it zero?¹

Composite function

We are now considering a composite function, i.e., for which the argument is not just an independent variable, but also a function. For the numerical derivative, there is no difference from a simple function. We build the two arrays x and $y = \sin(f(x))$ and proceed with $\Delta y/\Delta x$.

In Fig. 5.6a, the function $y = A \cdot \sin(kx)$ with $A = 0.1$ and $k = 2\pi$ is differentiated first once and then twice. The results are again the cosine and the negative sine, however, with different amplitudes, $0.1 \cdot 2\pi$ and $-0.1 \cdot (2\pi)^2$, respectively.

Questions

What is the amplitude A of the curve $A \cdot \sin(kx)$ in Fig. 5.6a?²

Let A be 10 times bigger than it is in Fig. 5.6a. How do you have to change the scale of the left and the right y-axes to get the same appearance in the figure?³

Why might it be advantageous to display a function $f(t) = \sin(2\pi t)$ instead of $f(x) = \sin(x)$?⁴

¹ The speed is maximum at zero crossings ($x = 0, \pi, 2\pi$ in Fig. 5.5, at the extrema in y_{1d}) and zero at turning points ($x = \pi/4, 3\pi/4$ in Fig. 5.5).

² $A = 0.1$.

³ Both axes also have to be scaled by a factor of 10: -4 to 4 and -40 to 40 .

⁴ Then, a period duration has the length 1, a quarter period ($\pi/2$) the length 0.25, and a half period (π) the length 0.5, thus always at simple rational numbers that are clearly visible on the x-axis.

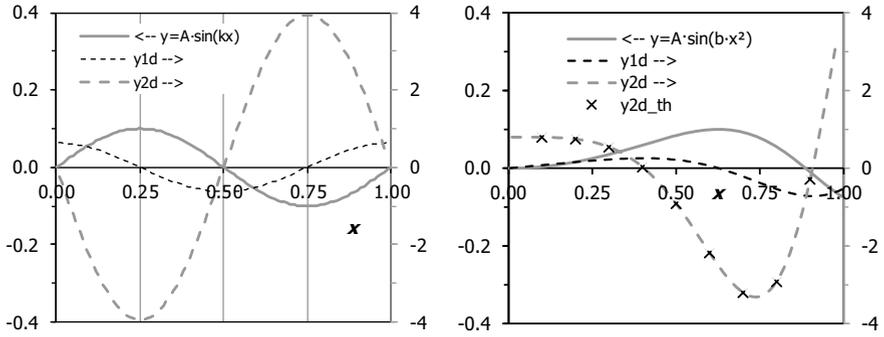


Fig. 5.6 **a** (left) $y=A \cdot \sin(2\pi x)$ and its first and second derivative. **b** (right) $y = A \cdot \sin(bx^2)$ with $A = 0.1$ and $b = 4$, its first and second numerical derivative, together with the analytical second derivative Eq. 5.7. For the derivatives, the right y-axis is valid.

Why might it be advantageous to divide the values of $f''(t)$, the second derivative of $\sin(2\pi t)$, by $4\pi^2$?⁵

In Eq. 5.7, we have chosen a parabola as an argument for the sine:

$$y = A \cdot \sin(bx^2) \tag{5.7}$$

The function $y = A \cdot \sin(bx^2)$ and its numerically obtained derivatives are displayed in Fig. 5.6b with continuous lines.

Analytical derivatives with the chain rule

The derivatives of a composite function $y = f(z)$, $z = g(x)$ are obtained with the chain rule:

$$\frac{dy}{dx} = \frac{df}{dz} \cdot \frac{dz}{dx} \tag{5.8}$$

For $y = \sin(2\pi x)$, we get

$$y' = -\cos(2\pi x) \cdot 2\pi \text{ and } y'' = -\sin(2\pi x) \cdot (2\pi)^2 \tag{5.9}$$

and for $y = \sin(b \cdot x^2)$:

$$\begin{aligned} y' &= -\cos(bx^2) \cdot 2bx \\ y'' &= -\sin(bx^2) \cdot (2bx)^2 + \cos(bx^2) \cdot 2b \end{aligned} \tag{5.10}$$

⁵ Then, we would expect an amplitude -1 , which is easier to verify in the diagram.

	A	B	C	D	E	F		A	B	C	D	E	F
3							2	A	0.10				
4	dx	0.0628	=2*PI()/100				3	k	6.28				
							4	dx	0.01				
5	=A7+dx	=SIN(x)	=((x+A9)/2)	=((A9-x)/(y1d-D7))	=((x-C7)/(y1d-D7))	=SIN(x)							
6	x	y	xc	y1d	y2d	-sin(x)	5	=A7+dx	=A*SIN(k*x)	=((x+A9)/2)	=((B9-y)/(y1d-D7))	=((A9-x)/(y1d-D7))	=((x-C7)/(y1d-D7))
7	0.00	0.00	0.03	1.00		0.00	6	x	y	xc	y1d	y2d	
8	0.06	0.06	0.09	1.00	-0.06	-0.06	7	0.00	0.00	0.01	0.63		
9	0.13	0.13	0.16	0.99	-0.13	-0.13	8	0.01	0.01	0.02	0.63	-0.25	
105	6.16	-0.13	6.19	1.00	0.13	0.13	105	0.98	-0.01	0.99	0.63	0.49	
106	6.22	-0.06	6.25	1.00	0.06	0.06	106	0.99	-0.01	1.00	0.63		
107	6.28	0.00			0.00		107	1.00	0.00				

Fig. 5.7 (S) Spreadsheet layout for calculating the first (y_{1d}) and second (y_{2d}) derivatives of the sine function **a** (left) for $\sin(x)$ and **b** (right) for $\sin(kx)$

Some points of y'' are also displayed in Fig. 5.6b with crosses. They lie on the curve obtained with numerical differentiation, indicating that our procedure is correct, especially our decision to plot the result over the middle of the three points from which the second derivative is obtained.

5.3.2 Data Structure and Nomenclature

- x, y arrays specifying the points of the curve, x values dx apart,
- x_C array with the x -values of the center of the segments,
- y_{1d} array containing the values of the first derivative,
- y_{2d} array containing the values of the second derivative.

5.3.3 Spreadsheet Layout

A spreadsheet layout for the differentiation of the sine function is given in Fig. 5.7 (S).

Questions

concerning Fig. 5.7 (S):

Interpret the formula in C5, valid for C8!⁶

What are the lengths of the arrays x , x_C , y_{1d} , and y_{2d} ?⁷ Also, compare with Table 5.2!

⁶ The spreadsheet formula in C5 calculates the x -value of the centre of the interval.

⁷ $\text{len}(x) = R107 - R6 = 101$, $\text{len}(x_C) = 100$, $\text{len}(y_{1d}) = 100$, $\text{len}(y_{2d}) = 99$ (see also Table 5.2).

Table 5.2 **a** (top) Function *deri* for determining the derivative of $y = f(x)$; **b** (bottom left) First and second derivatives of $y = \sin(2\pi x)$; **c** (bottom right) Lengths of the arrays x , $y1d$, and $y2d$

1	def deri(x,y):		
2	dx=x[1:]-x[:-1]		
3	dy=y[1:]-y[:-1]		
4	xC=(x[1:]+x[:-1])/2	#x	Center of segment
5	return xC,dy/dx		
6	x=np.linspace(0,1,101, endpoint=True)	len(x)	101
7	y=np.sin(2*np.pi*x)	len(y1d)	100
8	x1,y1d=deri(x,y)	len(x1)	100
9	x2,y2d=deri(x1,y1d)	len(y2d)	99
		len(x2)	99

The formula in D8 is $D8 = [(B9-y)/(A9-x)]$. Which value does EXCEL take for x and y ?⁸

How do you get a selection of $-\sin(x)$ in column F of 10 points, as presented in Fig. 5.5b?⁹

We plot (x_c, y_{1d}) . Show that the formulas in E5 (valid for E8) is correct!¹⁰

5.3.4 Python Program

In the Python program corresponding to Fig. 5.7 (S), the derivatives are built with a function *deri*, shown in Table 5.2a, requiring x and y as input and returning x_C , the center of the intervals, and $\Delta y/\Delta x$. The Python program in Table 5.2b treats $y = \sin(2\pi x)$.

The quantities dx , dy , and x_C are obtained by slicing the arrays x and y . Their length is, by construction, one less than that of x and y , as can be verified in Table 5.2c. We no longer have to think about the values of x for which the derivative is a good approximation, because this is already done within the function and returned as x_C . For the second derivative, we apply *deri* twice.

Composite function

In Table 5.3, we are considering the composite function $y = A \cdot \sin(bx^2)$ (Eq. 5.10).

The results for the 10 x -values specified in line 7 are shown in Fig. 5.6b. The numerically calculated values (marked –) and the theoretical ones (marked with x)

⁸ For x and y in C8 and D8, the values in the same (8th) row are taken from the column ranges with names x (A8) and y (B8).

⁹ With a VBA procedure comprising a loop of the type: FOR R = 7 TO 107, ..., R2 = R2 + 1.

¹⁰ Row $n-1$: $x-dx, x_c = x-dx/2, y1d$.

Row n : $x, x_c = x + dx/2, y1d, y2d$.

To get y_{2d} at the position x , we have to calculate the difference quotient for y at $x + dx/2$ and $x-dx/2$.

Table 5.3 First and second derivatives of $A \cdot \sin(bx^2)$

```

1 x=np.linspace(0,1,101,endpoint=False)
2 A=0.1
3 b=4
4 yf=A*np.sin(b*x**2)
5 x1,yf1d=deri(x,yf)
6 x2,yf2d=deri(x1,yf1d)
7 xx=np.linspace(0,1,10)
8 yf2d_th=(A*np.cos(b*xx**2)*2*b
9          -A*np.sin(b*xx**2)*(2*b*xx)**2)
10          #Theoretical second derivative

```

of the second derivative coincide, indicating that our simple numerical recipe yields sufficiently accurate derivatives.

Question

Over which arguments, type x or x_c , must the function y_{f2d-th} in Table 5.3 be plotted so that it correctly represents the second derivative of the function y ?¹¹

5.4 Integration

We determine the area under a sine curve and the length of a polynomial.

5.4.1 Area Under a Curve

Integral function, definite integral

The integral of a function $f(x)$ between x_1 and x_2 corresponds, in the simple cases we are dealing with, to the area limited by the curve, the x -axis, and two vertical boundaries. It can be positive or negative. The area under a curve between two adjacent interpolation points is calculated using the trapezoid rule visualized in Fig. 5.8a. The area of a trapezoid of width Δx is

$$F_{\Delta x}(x) = \frac{f(x - \Delta x) + f(x)}{2} \cdot \Delta x \quad (5.11)$$

The integral from x_1 to x_2 is the sum of all trapezoids in that region. It must be represented in the diagram over x_2 , the end of the integration interval.

¹¹ The function y_{f2d-th} has to be plotted over xx , the equivalent of x , because it is the analytically determined second derivative of $A \cdot \sin(bx^2)$ at position xx .

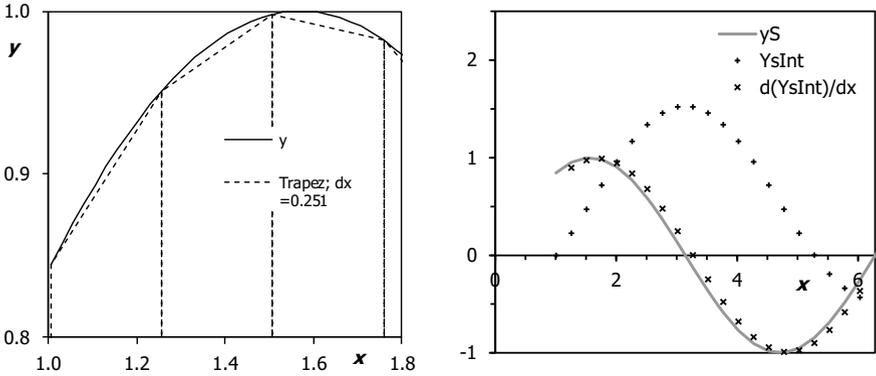


Fig. 5.8 **a** (left) Numerical calculation of an integral with the trapezoidal rule, $dx = 0.251$. **b** (right) Integral of the sine function and differentiation of the integral displayed over x (column A in Fig. 5.11 (S))

Figure 5.9 shows the integral function (a) of a sine function and (b) of a polynomial. The integral function of a *sine* is a *cosine*. The integral in Fig. 5.9a is theoretically $F(x) = (1 - \cos(x))$. The integral function of a polynomial of the n th order is a polynomial of the $(n + 1)$ th order.

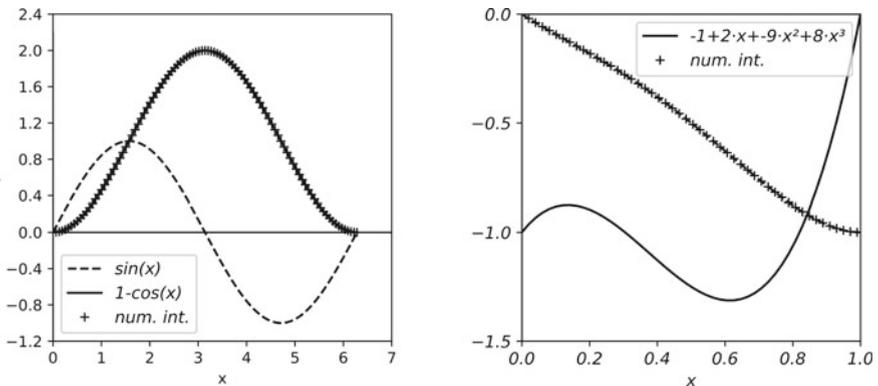


Fig. 5.9 Curves obtained with the Python program in Table 5.4. **a** (left) Integration of the sine function. **b** (right) Integration of a 3rd order polynomial (lower curve) (here, negative values of the integral)

Questions

What is the analytical integral function of a *sine*?¹²

What is the area under a *sine arc*?¹³

In Fig. 5.8b, the numerically obtained derivative of the integral function is shifted with respect to the original curve, contrary to the fundamental theorem of calculus. What, do you suspect, is the reason for this?¹⁴

5.4.2 Length of a Curve

The length Δs of a straight segment is easily calculated as

$$\Delta s = \sqrt{(\Delta x^2 + \Delta y^2)} \quad (5.12)$$

$$ds = \sqrt{1 + \left(\frac{dy}{dx}\right)^2} \cdot dx \quad (5.13)$$

The length of a curve may be approximated by the length of the sequence of segments used to approximate it, i.e., the sum of the lengths of all segments.

How is the function "length of a curve" correlated with the derivative of that curve? Looking at Fig. 5.10a, we may state that the slope of the length is always positive; and the bigger the *absolute value* of the derivative of the function, the bigger the derivative of the length. Looking at Eq. (5.13), we see that the slope ds/dx of the length is equal to the absolute value of the slope of $y(x)$ if $(dy/dx)^2 \gg 1$.

Question

Where does the curve "Length of a circle" as a function of the y value of the circle in Fig. 5.10b cross the x -axis?¹⁵

5.4.3 Data Structure and Nomenclature for the Arrays in the Integration

dx horizontal distance between the vertices

¹² $\int \sin(x)dx = \cos(x)$.

¹³ The area under a *sine arc* is 2, as can be seen from the value of the integral function at $x = \pi$ in Fig. 5.9a, based on the theorem $\int_a^b f(x)dx = F(b) - F(a)$.

¹⁴ The derivative is wrongly plotted over the end of the intervals, not correctly over their center.

¹⁵ The "Length of a circle" crosses the x -axis at $x = 0$ (for $y = 0$), $x = \pi$ (for $y = 0$ after having gone through a half circle), and $x = 2\pi$ (for $y = 0$, after having gone through a full circle).

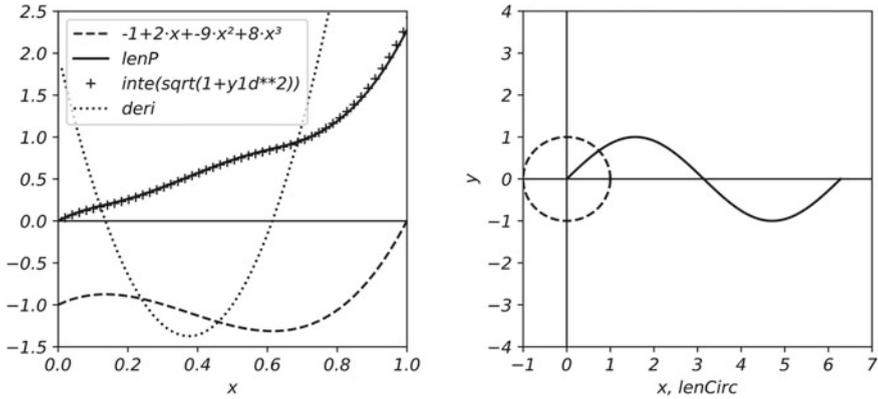


Fig. 5.10 **a** (left) Length of a polynomial, calculated with Eq. (5.12); with def `lenCurve` and a second time as `len2` in line 12 in Table 5.5. **b** (right) Length of a circle (reported on the x axis) as a function of the y -value of the circle; attention: the horizontal axis represents the dependent variable

- x_P x coordinates of a polynomial
- y_P y coordinates of the polynomial
- Y_{PInt} integral function of the polynomial
- len_P length of y_P
- x x coordinates of a sine and a cosine
- y_S y coordinates of a sine function
- y_C y coordinates of a cosine function
- Y_{SInt} integral function of the sine function
- len_{Circ} length of the circle defined by polar coordinates: $x = y_C$ and $y = y_S$.

5.4.4 Python Program

Area under a curve

A Python function, def `inte`, for performing the integration of a discretized function $y = f(x)$ with the trapezoid approximation is shown in Table 5.4. The returned array `integ` has the same length as `x`. Its first element is zero, because that was introduced by `np.zeros(len(x))`, and the first element is not overwritten. With the Python program in the next cells, we perform integrations of a *sine* function (middle cell) and a 3rd order polynomial (bottom cell). The results are displayed in Fig. 5.9.

Length of a polynomial

The approximation of Eq. 5.12 is implemented in the Python function def `lenCurve(x,y)` in Table 5.5, returning the cumulated sum of the individual elements `ds(np.cumsum(ds))` as a function of x .

In the second cell of Table 5.5, the length of the polynomial $y_P(x)$ of Table 5.4 is calculated, to be displayed in Fig. 5.10a. First, with `lenP = lenCurve(x,yP)` and then (line 12) as `len2`, applying Eq. (5.13).

Table 5.4 First cell: function for performing an integration of $y = f(x)$; second cell: $f(x) = \sin(x)$; third cell: $f(x)$ is a 3rd order polynomial

```

1 def inte(x,y):
2     dx=x[1:]-x[:-1]
3     yC=(y[1:]+y[:-1])/2 #y Center of segment
4     integ=np.zeros(len(x))
5     integ[1:]=np.cumsum(yC*dx)
6     return integ
7
7 x=np.linspace(0,2*np.pi,101, endpoint=True)
8 ys=np.sin(x)
9 yc=np.cos(x)
10 YsInt=inte(x,ys)
11
11 x=np.linspace(0,2,100,endpoint=True)
12 a,b,c,d=-1,2,-9,8
13 yP=a+b*x+c*x**2+d*x**3
14 YpInt=inte(x,yP)

```

Table 5.5 Function returning the length of the curve $y = f(x)$; x and y_P are defined in Table 5.4

```

1 def lenCurve(x,y):
2     ds=np.zeros(len(x))
3     dx2=(x[:-1]-x[1:])**2
4     dy2=(y[:-1]-y[1:])**2
5     ds[1:]=np.sqrt(dx2+dy2)
6     return(np.cumsum(ds))
7
7 FigStd('x',0,1,0.2,'y',-1.5,2.5,0.5)
8 plt.plot(x,yP,'k--')
9 lenP=lenCurve(x,yP)
10 plt.plot(x,lenP,'k-')
11 xC,yld=deri(x,yP) #Derivative
12 len2=inte(xC,np.sqrt(1+yld**2)) #Length accord. to formula
13 plt.plot(x[:-1],len2,'k+')
14 plt.show

```

Questions

Express lines 3 and 4 of Table 5.4 in one instruction! An `nd.array` is to be returned!¹⁶

Design suitable labels for the plots in lines 8, 10, and 13 of Table 5.5!

Length of a circle

In Table 5.6, we calculate the length of a circle. The full length is reported in the second cell. It is within 0.2% of the value of 2π . The length `lenCirc` (value on the x -axis) as a function of the y coordinate y_C of the circle (regarded as the independent variable) is displayed in Fig. 5.10b.

¹⁶ `Integ = np.array([0, *np.cumsum(yC*dx)])`

Table 5.6 Calculating the length of a circle and plotting it as a function of the y-values of the circle

```

1 phi=np.linspace(0,2*np.pi,101, endpoint=True)
2 yS=np.sin(phi)           #x-coordinates of the circle
3 yC=np.cos(phi)          #y-coordinates of the circle
4 lenCirc=lenCurve(yC,yS)
5 FigStd('x, lenCirc',-1,7,1,'y',-4,4,1)
6 plt.plot(yC,yS,'k--')
7 plt.plot(lenCirc,yS,'k-')
lenCirc[-1] 6.282
(2*np.pi) 6.283
    
```

5.4.5 Spreadsheet Solution

Area under the curve

A spreadsheet solution corresponding to Table 5.4 for the *sine* function is given in Fig. 5.11a. The integration is performed in column C by operating on the individual cells of y and taking the constant dx for all $x_{i+1} - x_i$. If the horizontal distance between the vertices is not constant, we have to replace dx in D8 by (x-A7), and so on.

The integration in Fig. 5.11a (S) starts at $x = 1.005$ (A7). The corresponding start value for the integration, zero, was entered into C7. The following cells accumulate the areas of the trapezoids. The resulting integral Y_{sInt} is shown in Fig. 5.8b, together with y_S , this, however, for $dx = 0.251/10$.

Question

According to the first fundamental theorem of calculus, the derivative of the integral over $f(x)$ should again yield the function $f(x)$. In Fig. 5.8b, however,

	A	B	C	D	E	F	G	
4	dx	0.251						Sub Trapez()
5	$=A7+dx$	$=SIN(x)$	$=C7+(y+B7)/2*dx$	$=C7+(y+B7)/2*dx$				r2 = 7
6	x	yS	YsInt	d(YsInt)/dx				For r = 7 To 11 Step 1
7	1.005	0.844	0		1.005	0.000		Cells(r2, 6) = Cells(r, 1) 'x
8	1.256	0.951	0.23	0.90	1.005	0.844		Cells(r2, 7) = 0
9	1.508	0.998	0.47	0.97	1.256	0.951		Cells(r2, 7) = 0
10	1.759	0.982	0.72	0.99	1.256	0.000		r2 = r2 + 1
11	2.010	0.905	0.96	0.94	1.256	0.951		Cells(r2, 6) = Cells(r, 1) 'x
12	2.262	0.771	1.17	0.84	1.508	0.998		Cells(r2, 7) = Cells(r, 2) 'y
107	26.138	0.844	0.00	0.76				r2 = r2 + 1
								Cells(r2, 6) = Cells(r + 1, 1) 'x.next
								Cells(r2, 7) = Cells(r + 1, 2) 'y.next
								r2 = r2 + 1
								Next r
								End Sub

Fig. 5.11 a (left, S) Integration of $y = \sin(x)$, columns F and G contain the data series for the trapezoids shown in Fig. 5.8a. b (right, P) VBA procedure for writing the coordinates of the trapezoids into columns F and G

$d(Y_{sInt})/dx$ is shifted relative to the function $y_{s(x)}$ to the right. Inspecting Fig. 5.11a, find out why that's the case!¹⁷

In column D, the integral is differentiated, and the result is also displayed in Fig. 5.8b as a function of x listed in column A. In this plot, the derivative $d(Y_{sInt})/dx$ is shifted to the right with respect to the original function. The reason for this flaw is that the derivative is plotted over x (D8 is plotted over A8), the end of the interval in which the derivative is built, instead of over the center of the interval.

Coordinates of the trapezoids by a scan-log procedure

Figure 5.11b (P) shows the program code for a procedure that writes the coordinates for the trapezoids from the table data for x and dY_{sInt}/dx in Fig. 5.11a into the columns F and G of Fig. 5.11a (S). Rows 7–12 of columns A and C are scanned with the loop index r , and three points of a trapezoid are transferred with each loop cycle to F and G, with the running index r_2 being incremented three times in each run. The result “Trapez” is shown in Fig. 5.8a.

5.5 Vectors in the Plane

Polar and Cartesian coordinates are converted one into the other. Two vectors are added, and their scalar product is built. Perpendicular bisectors are erected on line segments. Arrows representing forces are attached to application points in the xy -plane.

5.5.1 Vectors

Vectors in polar and Cartesian coordinates

Vectors have a magnitude l and a direction that, in the plane, can be determined by the angle α to the positive x -axis. Alternatively, a vector can be defined by Cartesian coordinates (V_x, V_y) . The two coordinate systems can be transformed one into the other by

$$\begin{aligned} V_x &= l \cdot \cos(\alpha) \\ V_y &= l \cdot \sin(\alpha) \end{aligned} \quad (5.14)$$

¹⁷ The derivative $d(Y_{sInt})/dx$ is plotted versus x listed in column A. The derivative calculated for a segment is, however, to be plotted over the horizontal center of this segment.

$$l = \sqrt{V_x^2 + V_y^2}$$

$$\alpha = \text{arcus tangens}(V_x, V_y) \tag{5.15}$$

Attention: The order of the arguments in *arcus tangens* is different in the EXCEL and Python functions:

EXCEL	Python
ATAN2(x, y)	np.arctan2(y, x)

Vector addition

Two vectors are added by adding their Cartesian coordinates individually:

$$\underline{W} = \underline{V} + \underline{U}, \quad (W_x, W_y) = (V_x + U_x, V_y + U_y).$$

Vector addition is illustrated for forces in Fig. 5.12a with the axes F_x and F_y scaled in units of N (Newton). The vectors are represented as arrows with their bases in the origin of the coordinate system and the coordinates of their head points being the coordinates of the vector. The resulting vector \underline{W} points to the corner of the parallelogram spanned by \underline{V} and \underline{U} .

In Fig. 5.12b, the plane xy is displayed, scaled in units of m with the arrows representing the vectors of Fig. 5.12a attached at a point of application, here, (2, 4), after being scaled with a scalar with the physical dimension m/N (here, 0.8 m/N) to get the same physical unit as the axes, namely, m (meter). The scaling factor is chosen so that arrows of convenient length result that fit into the chart.

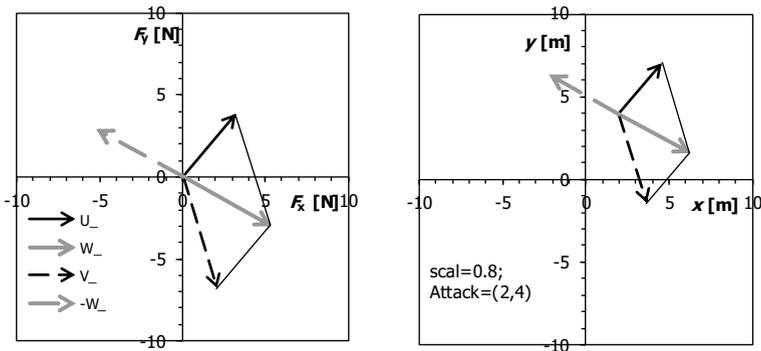


Fig. 5.12 a (left) Vector addition of forces in the (F_x, F_y) plane with axes in physical units N; two vectors \underline{U} and \underline{V} are added to produce the resulting vector \underline{W} . b (right) Vector arrows attached to a point in the xy -plane with axes in physical units m

Question

What is the physical unit of the scaling factor for forces in the xy -plane?¹⁸

Scalar product

The scalar product of two vectors \underline{V} and \underline{U} can be calculated in two ways:

- by multiplying their Cartesian coordinates and summing up the products:

$$\underline{U} \cdot \underline{V} = U_x \cdot V_x + U_y \cdot V_y \quad (5.16)$$

- or using polar coordinates with the included angle $\gamma = \alpha_U - \alpha_V$:

$$\underline{U} \cdot \underline{V} = l_U \cdot l_V \cdot \cos(\gamma) \quad (5.17)$$

5.5.2 Data Structure and Nomenclature

$\underline{U}, \underline{V}, \underline{W}$	vectors with two components
l_U, l_V, l_W	length of $\underline{U}, \underline{V}$, and \underline{W}
$\alpha_U, \alpha_V, \alpha_W$	angles of $\underline{U}, \underline{V}$, and \underline{W} to the x-axis
<u>Attack</u>	point of application in the plane, vector with two components (x and y)
$\underline{U}_{at}, \underline{V}_{at}, \underline{W}_{at}$	coordinates of heads of arrows attached to the point of application.

5.5.3 Spreadsheet Layout

Fig. 5.13 (S) shows the spreadsheet layout for the drawings in Fig. 5.12.

The lengths and the angles of the two vectors are specified with sliders (scroll bars). These polar coordinates are transformed into Cartesian coordinates in rows 7 and 8, where the scalar product of the two vectors is also calculated in two ways ($\underline{U} \cdot \underline{V}$ and Scp), according to Eqs. (5.16) and (5.17), respectively. \underline{W} is the sum of $\underline{U} + \underline{V}$, written into C11:C12 with the matrix formula in curly brackets $\{=U_ + V_ \}$. The Cartesian coordinates are transformed into the polar coordinates (l_W, α_W) in I10:I12. The length l_W is calculated with $\sqrt{W(1)^2 + W(2)^2}$.

Questions

concerning Fig. 5.13 (S):

¹⁸ The physical unit of the scaling factor is m/N, so that the length of the arrows is in m.

	A	B	C	D	E	F	G	H	I	J
1	←			→	50		length	IU	5.00	=E1/10
2	←			→	140		angle	aU	0.87	=(E2-90)/180*PI()
3	←			→	71			IV	7.10	=E3/10
4	←			→	17			aV	-1.27	=(E4-90)/180*PI()
5										
6			U_			V_				
7	.x	0.00	3.21	=IU*COS(aU)	0.00	2.08	=IV*COS(aV)	U·V	-19.33	=SUMPRODUCT(U_,V_)
8	.y	0.00	3.83	=IU*SIN(aU)	0.00	-6.79	=IV*SIN(aV)	Scp	-19.33	=IU*IV*COS(aU-aV)
9										
10			W_			-W_		IW	6.0614	=SQRT(INDEX(W_,1)^2
11		0.00	5.29	{=U_+V_}	0.00	-5.29	{=-W_}			+INDEX(W_,2)^2)
12		0.00	-2.96		0.00	2.96		aW	-0.51	=ATAN2(C11;C12)

Fig. 5.13 (S) Specification of two vectors \underline{U} and \underline{V} by their length and their angle to the x -axis with sliders (rows 1–4); transformation to Cartesian coordinates (C7:F8); their scalar product (twice in H7:I8); their sum (C11:D12); and the polar coordinates of the sum (I10:I12)

What is the apparent range of the numbers generated by the slider in A2:D2?¹⁹

How do you get positive and negative angles between $-\pi/2$ and $\pi/2$ from an always positive output of a slider with a range of 0–180?²⁰

Where are the length and angle of the vector $\underline{W} = \underline{U} + \underline{V}$ calculated?²¹

With which instruction do you get the first entry in the named range $W_$?²²

In rows 11 and 12 of Fig. 5.13 (S), the vector sum of the two vectors is built with a matrix formula $\underline{W}_ = \{ \underline{U} + \underline{V} \}$. Addition is not possible in polar coordinates; if the vectors are specified by length and angle, they must be converted to Cartesian coordinates before addition.

Figure 5.14 (S) contains the extension of the calculation for obtaining the coordinates of arrows representing vectors applied at a point (*Attack*) in the plane. *Attack* is specified in Z4:Z5. The vector arrows in the xy -plane go from *Attack* to *Attack* + $\underline{U} \cdot scal$, and so on. All four arrows can be entered into a chart together as one series by specifying AB4:AL4 as SERIES X-VALUES and AB5:AL5 as SERIES Y-VALUES, because the respective ranges in the spreadsheet are separated by empty cells.

Ψ Empty cells separate curves.

¹⁹ The slider in A2:D2 ranges from 0 to 180, angles in degree.

²⁰ With a formula as in I2 = [= (E2-90)/180*PI()], one can get positive or negative values.

²¹ The length and angle of the new vector \underline{W} are calculated in I10 (I_W) and I12 (a_W).

²² INDEX(W_,1); in EXCEL, the first entry is indexed as 1, contrary to PYTHON where indexing starts with 0.

	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM
2	scal	0.8	$\{=U_*scal+Attack_\}$ $\{=V_*scal+Attack_\}$ $\{=W_*scal+Attack_\}$ $\{=-W_*scal+Attack_\}$											
3	Attack		U_ V_ W_ -W_											
4	2.00		2.00	4.57		2.00	3.66		2.00	6.23		2.00	-2.23	
5	4.00		4.00	7.06		4.00	-1.43		4.00	1.63		4.00	6.37	

Fig. 5.14 (S) Calculating the coordinates of arrows representing the vectors \underline{U} , \underline{V} , \underline{W} in the xy -plane of Fig. 5.12a

5.5.4 Python Program

The Python program corresponding to Fig. 5.13 (S) is given in the first two cells of Table 5.7. The results of the calculation are shown in the third cell (bottom right) of that table. They correspond exactly to the values in Fig. 5.13 (S). Lines 16-19 calculate the coordinates of the vector arrows in the plane, similar to Fig. 5.14 (S). $W_.shape = (2,)$ indicates that \underline{W} is an array with two elements. The values for the scalar product and the polar coordinates of \underline{W} are exactly the same as in Fig. 5.13 (S).

Notice! The argument order of *arcus tangens* is different in the EXCEL and Python functions (see Sect. 5.5.1).

The main program in Table 5.8 calls *ArrowP* (Table 5.1 in Sect. 5.2.4) four times. It is a continuation of Table 5.7 and can refer to the data, e.g., \underline{U}_{at} , specified therein. The resulting diagram is shown in Fig. 5.15.

Table 5.7 a (top) Python program for defining two vectors with their length and their angle to the x -axis, being transformed into Cartesian coordinates and their scalar product being built; **b** (bottom left) the sum of the two vectors is built and the coordinates of all vectors are calculated when attached to a point of attack in the plane; **c** (bottom right) reports values and shapes of some variables

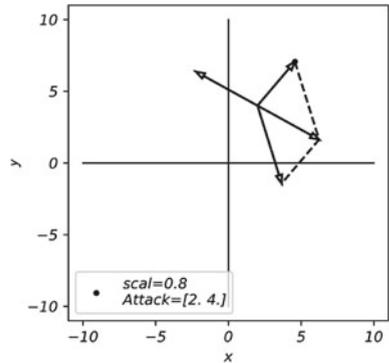
1	<code>lU=5.0</code>	<code>#Length of vector U</code>	
2	<code>aU=0.8727</code>	<code>#Angle to x-axis</code>	
3	<code>U_=lU*np.array([np.cos(aU),np.sin(aU)])</code>		
4	<code>lV=7.10</code>		
5	<code>aV=-1.274</code>		
6	<code>V_=lV*np.array([np.cos(aV),np.sin(aV)])</code>		
7			
8	<code>Scp=lU*lV*np.cos(aU-aV)</code>		
9	<code>UV=U_@V_</code>	<code>#Dot product</code>	
10	<code>W_=U_+V_</code>		UV -19.33
11	<code>lW=np.sqrt(np.sum(W_**2))</code>		Scp -19.33
12	<code>aW=np.arctan2(W_[1],W_[0])</code>		
13			
14	<code>Attack_=np.array([2.00,4.00])</code>		W_ 5.29 -2.96
15	<code>scal=0.8</code>		W_.shape (2,)
16	<code>U_at=U_*scal+Attack_</code>		lW 6.06
17	<code>V_at=V_*scal+Attack_</code>		aW -0.51
18	<code>W_at=W_*scal+Attack_</code>		
19	<code>W_at_opp=-W_*scal+Attack_</code>		

Table 5.8 Continuation of Table 5.7; the program plots four arrows and the parallelogram of forces, *ArrowP* from Table 5.1

```

20 FigStd('x', -10, 10, 5, 'y', -10, 10, 5)
21 ArrowP(Attack_, U_at, lw=1.5)
22 ArrowP(Attack_, V_at, lw=1.5)
23 ArrowP(Attack_, W_at, lw=1.5)
24 ArrowP(Attack_, W_at_opp, lw=1.5)
25 plt.plot([U_at[0], W_at[0], V_at[0]],
           [U_at[1], W_at[1], V_at[1]], 'k--')
           #Parallelogram, 3 sides
26 plt.axis('scaled')
           #Axis lengths to scale
    
```

Fig. 5.15 Vector diagram corresponding to Fig. 5.12b, but drawn with the Python program in Table 5.8



Questions

How do you produce the labels shown in Fig. 5.12b (EXCEL) and Fig. 5.15 (Python)?²³

What is the effect of the statement `plt.axis('scaled')`?²⁴

²³ Excel: “scal = ”&scal&”; Attack = (“&Index(Attack_1)&”, “&Index(Attack_2)&”).

Python: lbl = "scal = " + str(scal) + "\nAttack = " + str(Attack_).

²⁴ The lengths of the axes in the figure correspond to the scaling of the axes specified in the program (see Fig. 5.15).

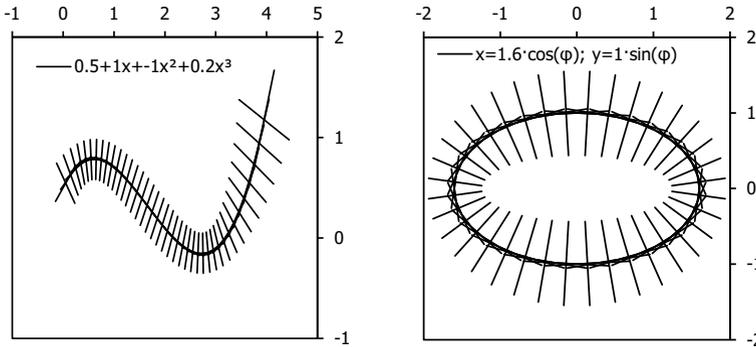


Fig. 5.16 Tangents and perpendiculars, **a** (left) for a polynomial as obtained from the spreadsheet in Fig. 5.18 (S), and **b** (right) on an ellipse constructed using polar coordinates (see legend)

5.6 Tangents to and Perpendiculars on a Curve

Tangents are fixed to a curve and perpendiculars are erected on it, taking a third-degree polynomial as an example. All calculations are performed with vectors. Python uses a loop to draw a multitude of segments. In contrast, in the EXCEL realization, the curve is calculated in a spreadsheet, and the segments' coordinates are generated with a scan-log procedure.

5.6.1 At/On a Polynomial and an Ellipse

We determine and display tangents to and perpendiculars on a polynomial, first with a Python program and then through the combination of a spreadsheet calculation and a VBA procedure. An example is shown in Fig. 5.16a. In Fig. 5.16b, the same construction is shown for an ellipse.

The coordinates of the ellipse are obtained with

$$x = a_X \cdot \cos(\phi); y = a_Y \cdot \sin(\phi) \quad (5.18)$$

Questions

What are the coefficients a_X and a_Y of Eq. 5.18 for the ellipse in Fig. 5.16b²⁵

²⁵ From the legend, we infer $a_X = 1.6$ and $a_Y = 1$.

Table 5.9 Program for determining the endpoints of segments representing the tangents to and the perpendiculars on a curve

```

1  #Define polynomial of 3rd order
2  a,b,c,d=0.5,1.0,-1,0.2
3  dx=0.1
4  x=np.arange(0,4+dx,dx)
5  y=a+b*x+c*x**2+d*x**3
6
7  #Segments, their centers and lengths
8  xC=(x[1:]+x[:-1])/2
9  yC=(y[1:]+y[:-1])/2
10 Dx=x[1:]-x[:-1]
11 Dy=y[1:]-y[:-1]
12
13 scal=5      #Scaling the length of arrows
14 #Tangentials          19 #Perpendiculars
15 xL=xC[1:]-Dx[1:]*scal  20 xpL=xC[1:]-Dy[1:]*scal
16 xR=xC[1:]+Dx[1:]*scal  21 xpR=xC[1:]+Dy[1:]*scal
17 yL=yC[1:]-Dy[1:]*scal  22 ypL=yC[1:]+Dx[1:]*scal
18 yR=yC[1:]+Dy[1:]*scal  23 ypR=yC[1:]-Dx[1:]*scal
    
```

In Fig. 5.16a, the perpendiculars do not seem to be perpendicular to the curve, contrary to Fig. 5.16b. What is the reason for this?²⁶

5.6.2 Data Structure and Nomenclature

- a, b, c, d coefficients of the polynomial $y = a + b \cdot x + c \cdot x^2 + d \cdot x^3$
- x x -coordinates of the vertices, dx apart
- y y values of the function
- x_C, y_C center of the segments
- dx, dy lengths of the segments in the x and y directions
- x_L, y_L left coordinates of the tangential segments
- x_R, y_R right coordinates of the tangential segments
- x_{pL}, y_{pL} left coordinates of the perpendicular segments
- x_{pR}, y_{pR} right coordinates of the perpendicular segments.

5.6.3 Python Program

A Python program for our task is given in Table 5.9. The segments’ centers and lengths are elegantly obtained with one instruction each through slicing, as are, again, the arrays for the coordinates of the tangential and perpendicular segments.

²⁶ In Fig. 5.16a, the lengths of the x - and y -axes do not conform to the scaling (-1 to 5) and (-1 to 2). In Fig. 5.16b, the lengths of the axes have been adjusted to the scaling.

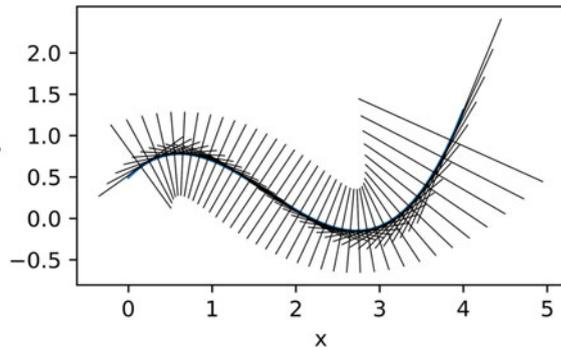
Table 5.10 Program for plotting the results of Table 5.9

```

1 FigStd('x',-1,5,1,'y',-1,2,0.5)
2 plt.plot(x,y)
3 for i in range(0,len(xC)-1):
4     plt.plot([xL[i],xR[i]],
5              [yL[i],yR[i]],'k-',lw=0.5)
6 for i in range(0,len(xpL)-1):
7     plt.plot([xpL[i],xpR[i]],
8              [ypL[i],ypR[i]],'k-',lw=0.5)
9 plt.axis('scaled') #Only effective downstream (at the end)
10 plt.savefig('PhEx 5.5 polynomial.png',dpi=1200)

```

Fig. 5.17 Tangents to and perpendiculars on a polynomial; the coordinates are calculated in Table 5.9 and drawn with the program in Table 5.10



Tangents are drawn through the center (x_c, y_c) of a segment ranging from

$$(x_c - dx \cdot scal, y_c - dy \cdot scal) \text{ to } (x_c + dx \cdot scal, y_c + dy \cdot scal).$$

Scal is a scalar chosen to give a suitable length to the segments representing the tangents and perpendiculars in the figure. No explicit for-loop is necessary to determine the various coordinates, contrary to the VBA code in Sect. 5.6.4. In Python, this is implicitly done with slicing.

The result of the calculation in Table 5.9 is plotted by the program in Table 5.10 with the resulting chart in Fig. 5.17. Here, looping over all tangents and all perpendiculars is chosen, contrary to Fig. 5.18b (S), where all coordinates are written into one column for x and another one for y , with empty cells separating segments so that they can be entered as one series into the figure. With the statement `plt.axis('scaled')`, the axes' lengths are adapted to the axes' scaling. Here, the perpendiculars are visibly orthogonal to the curve, contrary to Fig. 5.16a. With the last statement in Table 5.10, the diagram in Fig. 5.17 is stored as a *png* file.

5.6.4 Spreadsheet Solution

In the spreadsheet of Fig. 5.18a (S), the coefficients of a polynomial of the 3rd degree are specified, and 41 points (x, y) on that curve are calculated similar to

	A	B	C	D	E
1	a	0.50		cc	-1.00
2	b	1.00		d	0.20
3			0.5+1x+-1x ² +0.2x ³		
4	dx	0.10			
5					
6			=a+b*x+cc*x ² +d*x ³		
7	x	y		by VBA procedure	
8	0.0	0.50		-0.15	0.36
9	0.1	0.59		0.25	0.73
10	0.2	0.66			
47	3.9	1.05		0.66	0.98
48	4.0	1.30		0.64	0.58

1	Sub TangVert()	Cells(r2, 5) = yC - dy	16
2	scal = 5	r2 = r2 + 1	17
3	r2 = 8	Cells(r2, 4) = xC + dx	18
4	For r = 8 To 47	Cells(r2, 5) = yC + dy	19
5	xA = Cells(r, 1)	r2 = r2 + 2	20
6	yA = Cells(r, 2)	'perpendiculars	21
7	xB = Cells(r + 1, 1)	Cells(r2, 4) = xC - dy	22
8	yB = Cells(r + 1, 2)	Cells(r2, 5) = yC - dx	23
9	'center	r2 = r2 + 1	24
10	xC = (xA + xB) / 2	Cells(r2, 4) = xC + dy	25
11	yC = (yA + yB) / 2	Cells(r2, 5) = yC + dx	26
12	dx = (xB - xA) * scal	r2 = r2 + 2	27
13	dy = (yB - yA) * scal	Next r	28
14	'tangentials	End Sub	29
15	Cells(r2, 4) = xC - dx		30

Fig. 5.18 a (S, left) Table for calculating the x and y coordinates of a 3rd order polynomial; the columns D and E from row 8 contain the outputs of the VBA procedure in (b). b (P, right) VBA procedure for calculating the coordinates of the segments representing the tangents and perpendiculars to the curve $y = f(x)$ and storing them all in columns D and E

Table 5.9. The coordinates of the segments, representing the tangents at and the perpendiculars on the curve, are calculated in a VBA procedure SUB *TangVert* in Fig. 5.18b (P).

The subroutine is of the type *scan-log* and applies the construction *loop2i*. The loop index r runs down A8:A47, scanning the values in columns 1 and 2. The index r_2 specifies the row of the output in columns D and E and is incremented in lines 17, 20, 24, and 27. In each cycle, the coordinates of the tangent and those of the perpendicular are calculated and written one after the other into the same columns, always separated by blank lines so that they are plotted as isolated segments when entered as one series into a chart.

5.7 Banked Curve

We calculate the forces acting on a vehicle running through a banked curve. The gravitational and centrifugal forces are decomposed into vectors on the road and perpendicular to it. The components are combined to get the forces pressing the vehicle onto the road and pushing it perpendicular to its track on the road.

5.7.1 Cross-Section of the Road

Turns on roads are generally banked so that the surface is inclined towards the inside of the turn. The reason for this is that the car is in less danger of being pushed out of the turn if the centrifugal force C is not fully working parallel to the road's surface. In addition, the gravitational force G also has a component parallel

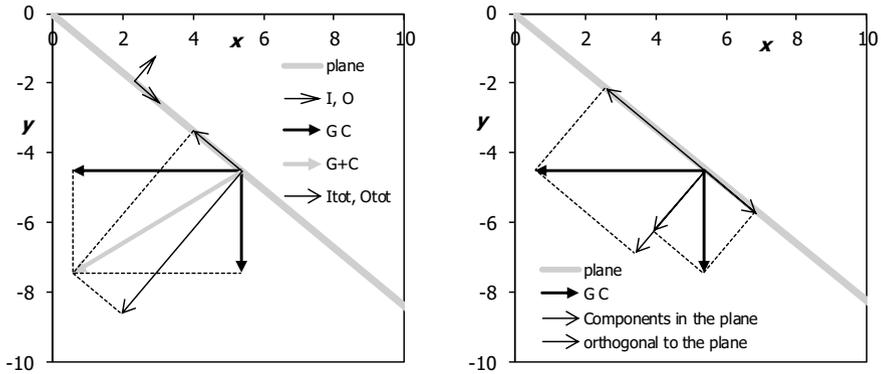


Fig. 5.19 **a** (left) The total force $\underline{G} + \underline{C}$ is decomposed into components parallel and orthogonal to the line (cross-section of the road). **b** (right) The two vectors \underline{G} and \underline{C} are decomposed individually into components parallel and orthogonal to the bank line. Inclination angle $\alpha = 40^\circ$

to the surface towards the inside of the turn. We represent the situation with a straight line ("bank line") describing the road's cross-section.

We treat the task with 2-dimensional vectors, namely, unit vectors parallel and orthogonal to the bank line and vectors representing the forces acting on the car, gravitational mg and centrifugal $-mv^2/r$, where m , v , and r are the mass and velocity of the vehicle and the radius of the curve, respectively. The signs are valid for the situation of Fig. 5.19. As the mass occurs in both forces, we need not consider it explicitly when interested in comparisons between the two forces. So, we use accelerations with sizes $G = g$ and $C = v^2/r$. The components of the accelerations in the bank line and perpendicular to it are calculated with scalar products with the in-plane and out-of-plane unit vectors.

The acceleration parallel to the road drives the car out of its track. The acceleration perpendicular to the bank line presses the vehicle onto the road, thus determining the frictional force.

Questions

What are convenient physical units of \underline{G} and \underline{C} ?²⁷

Is it sensible to interpret \underline{G} and \underline{C} as accelerations?²⁸

²⁷ As we have defined these quantities as accelerations, their unit is N/kg.

²⁸ Open to debate. Pro: The mass does not play any role. But does the frictional force depend on the mass?

5.7.2 Data Structure and Nomenclature

α	inclination of the track towards the horizontal
$\underline{I} = [\cos(\alpha), -\sin(\alpha)]$	unit line vector
$\underline{O} = [\sin(\alpha), \cos(\alpha)]$	unit vector orthogonal to the bank line
$\underline{G} = [0, 9.81]$	gravitational acceleration [N/kg] = [m/s ²]
rr	radius of the curve
v	speed of the car
$\underline{C} = [-v^2/rr, 0]$	centrifugal acceleration
<u>Attack</u>	point of attack of the forces
$\underline{G_I}, \underline{C_I}$	components in the plane
$\underline{I}_{tot} = \underline{G_I} + \underline{C_I}$	total force in the plane
$\underline{G_O}, \underline{C_O}$	components orthogonal to the plane
$\underline{O}_{tot} = \underline{G_O} + \underline{C_O}$	total force orthogonal to the plane.

5.7.3 Python Program

In the first cell of Table 5.11, the four basic vectors, line vector \underline{I} , orthogonal vector \underline{O} , gravitational acceleration \underline{G} , and centrifugal acceleration \underline{C} , are determined from the parameters of the exercise: inclination angle α , gravitational acceleration g , speed v of the car and radius rr of the curvature of the road. In the second and third cell of that table, vector operations are applied to decompose the accelerations into components parallel and orthogonal to the bank line. $\underline{G@I}$ is a matrix multiplication, for one-dimensional vectors equivalent to the scalar product. The resulting values are listed in Table 5.12. They have the same values as those in the spreadsheets in Sect. 5.7.4.

In Table 5.13, the coordinates of the arrows, representing the vectors in the xy -plane with suitable length, are calculated. The starting point Attack of all arrows is given as multiple *PoA* of the unit line vector, starting at the origin (0, 0) of

Table 5.11 First cell: specifying line vectors \underline{I} (in-plane) and \underline{O} (orthogonal to plane), and gravitational \underline{G} and centrifugal \underline{C} acceleration

1	<code>a=40</code>	<code>#Angle of inclination, degrees</code>
2	<code>lbl_1="α="+str(a)+"°"</code>	
3	<code>a*=np.pi/180</code>	<code>#Angle in radian</code>
4	<code>rr=100</code>	<code>#[m], Radius of curve</code>
5	<code>v=40</code>	<code>#[m/s], Speed of car</code>
6		
7	<code>I_=np.array([np.cos(a), -np.sin(a)])</code>	
8	<code>O_=np.array([-I_[1], I_[0]])</code>	
9	<code>G_=np.array([0, -9.81])</code>	<code>#Gravitational acceleration</code>
10	<code>C_=np.array([-v**2/rr, 0])</code>	<code>#Centrifugal acceleration</code>
11	<code>GI_=G_@I_*I_</code>	<code>14 GO_=G_@O_*O_</code>
12	<code>CI_=C_@I_*I_</code>	<code>15 CO_=C_@O_*O_</code>
13	<code>Itot_=GI_+CI_</code>	<code>16 Otot_=GO_+CO_</code>

Table 5.12 Numerical values of the vectors specified in Table 5.11, values for GO_* and CO_* are the same as in Fig. 5.21 (S)

I_*	[0.77 -0.64]	GI_*	[4.83 -4.05]
O_*	[0.64 0.77]	CI_*	[-9.39 7.88]
G_*	[0.00 -9.81]	$Itot_*$	[-4.56 3.83]
C_*	[-16.00 0.00]	$Otot_*$	[-11.44 -13.64]

Table 5.13 Calculating the coordinates of the arrows in Fig. 5.19; PoA is a scalar determining the point of attack on the bank line

1	$PoA=7$	7	$CI=CI_*scal+Attack_*$
2	$scal=0.3$	8	$Itot=Itot_*scal+Attack_*$
3	$Attack=np.array(I_*PoA)$	9	$GO=GO_*scal+Attack_*$
4	$G=G_*scal+Attack_*$	10	$CO=CO_*scal+Attack_*$
5	$C=C_*scal+Attack_*$	11	$Otot=Otot_*scal+Attack_*$
6	$GI=GI_*scal+Attack_*$	12	$GpC=(G+C)*scal+Attack_*$

Table 5.14 Plotting the arrows that represent the various accelerations in Fig. 5.19a with *ArrowP*

```

1 FigStd('x',0,10,2,'y',-10,0,2)
2 PoA=7
3
4 plt.plot([0,20*I[0]],[0,20*I[1]],c='0.7',lw=3,ls='-')
5                                     #Plane
6 At_=I_*3                             #Attack point of the unit vectors
7 scal=1
8 Arrow(At_[0],At_[1],
9       At_[0]+I_[0]*scal, At_[1]+I_[1]*scal)
10 Arrow(*At_,*(At_+O_*scal))        #* Decomposes the array
11
12 At_=Attack_*
13 Arrow(*At_,*G_)
14 Arrow(*At_,*C_)
15 Arrow(*At_,*GpC_)
16 Arrow(*At_,*Itot_)
17 Arrow(*At_,*Otot_)
17 #Parallelogram of forces:
18 plt.plot([C[0],GpC[0],G[0]],[C[1],GpC[1],G[1]],'k:')
19 plt.plot([Itot[0],GpC[0],Otot[0]],
20          [Itot[1],GpC[1],Otot[1]],'k:')

```

the bank line. The endpoints are obtained with instructions like $G = \underline{G}_*scal + \underline{Attack}$. Plotting the arrows is achieved with Table 5.14.

Question

Interpret the statement in Table 5.11: $G_@I_*I_!$ ²⁹

²⁹ $G_@I_*$ is a matrix multiplication of two vectors, equivalent to $G_{[0]}*I_{[0]} + G_{[1]}*I_{[1]}$, yielding a scalar. $G_@I_*I_*$ is the multiplication of the vector I_* with this scalar.

5.7.4 Spreadsheet Solution

A spreadsheet solution of the banked-curve problem is shown in Fig. 5.20 (S).

We specify the line vectors \underline{I} and \underline{O} and the forces \underline{G} and \underline{C} as column vectors with two components. We have to pay attention to operator precedence when calculating the centrifugal force $-v^2/rr$. The values of the coordinates are the same as in Table 5.12, left cell.

- ▶ In EXCEL, the *sign* operator $-$ has precedence over the *power* operator \wedge . We therefore have to enter $= -(v^2/rr)$. In Python, $-v**2/rr$ will accomplish the task.

In Fig. 5.21 (S), \underline{G} and \underline{C} are decomposed into components parallel and orthogonal to the banked curve’s characteristic line. The scalar product is obtained with SUMPRODUCT. In Fig. 5.22 (S), the components of the corresponding arrows in the plane are calculated.

Attack is the starting point of all arrows, and $G, C, GI, CI, I_{tot}, GO, CO, O_{tot}$ are the endpoints of the arrows representing the various accelerations in the xy -plane.

We can create a living figure by introducing sliders to vary the parameters α , rr , and v .

5.8 Weighted Average

Inclination of road	angle	40 °	\underline{I}	\underline{O}	\underline{G}	\underline{C}
	a	0.70 radian	0.77 =COS(a)	0.64 =SIN(a)	0	-16 =-(v^2/rr)
Radius of curve	rr	100 m	-0.64 =SIN(a)	0.77 =-COS(a)	-9.81	0
velocity of car	v	40 m/s				
		144 km/h				
					plane	
					0	15.32 {=I_*20}
					0	-12.86

Fig. 5.20 (S) Specifying line vectors \underline{I} and \underline{O} , and gravitational \underline{G} and centrifugal \underline{C} acceleration, same specifications as in Table 5.11, same values as in Table 5.12

Components in the plane			orthogonal to the plane		
$=SUMPRODUCT(G_{:1})*I_{:1}$ $=SUMPRODUCT(C_{:1})*I_{:1}$ $=GI_{:1}+CI_{:1}$			$=SUMPRODUCT(G_{:2})*O_{:2}$ $=SUMPRODUCT(C_{:2})*O_{:2}$ $=GO_{:2}+CO_{:2}$		
GI_	CI_	Itot_	GO_	CO_	Otot_
4.83	-9.39	-4.56	-4.83	-6.61	-11.4
-4.05	7.88	3.83	-5.76	-7.88	-13.6

Fig. 5.21 (S) Continuation of Fig. 5.20 (S), decomposing the gravitational and centrifugal forces into components parallel and orthogonal to the bank line

poA	7	scal	0.3								
$=_1 * poA$	$\{=G_*scal+Attack_\}$										
Attack_	G	C	GI	CI	Itot	GO	CO	Otot	G+C		
5.36	5.36	0.56	6.81	2.55	3.99	3.91	3.38	1.93	0.56		
-4.50	-7.44	-4.50	-5.72	-2.14	-3.35	-6.23	-6.86	-8.59	-7.44		

Fig. 5.22 (S) Continuation of Fig. 5.21 (S), calculating the coordinates of the vector arrows to be drawn in the xy-plane; the values of the coefficients are to be compared with the corresponding values in Table 5.12, right cell

The equilibrium of a mobile with two arms is calculated with the law of the lever. The mathematical construct is a weighted average. The calculation uses vectors for forces and arms.

5.8.1 A Mobile with Two Arms

Figure 5.23a shows a mobile with one horizontal crossbar and weights attached at its left and right ends, balanced by a counter-force applied at the center of gravity of the construction. The crossbar is supposed to be weightless. The equilibrium is calculated with the law of the lever:

$$g \cdot m_L \cdot x_L = g \cdot m_R \cdot x_R$$

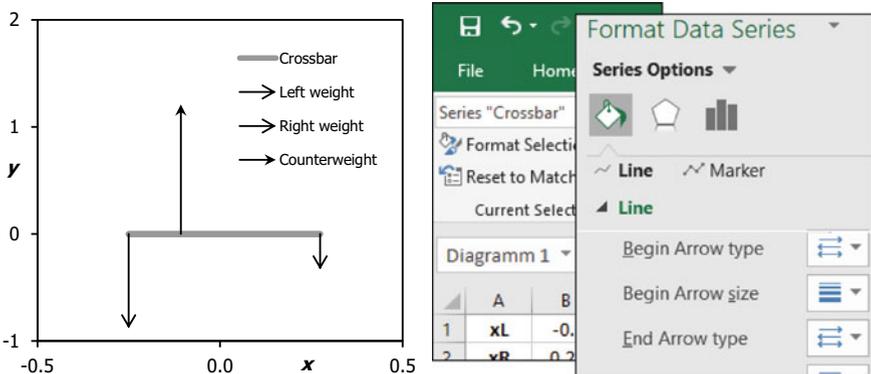


Fig. 5.23 a (left) Mobile with one horizontal crossbar. b (right) EXCEL menu for editing the series “Crossbar” in a; the arrowheads are introduced by FORMAT DATA SERIES/SERIES OPTIONS/FILL&LINE/END ARROW TYPE

where m_L and m_R are the masses attached at the ends of the bars and x_L and x_R their distance to the fulcrum.

We use the vector formulation for the equilibrium of torques:

$$\underline{W}_L \times \underline{X}_L = \underline{W}_R \times \underline{X}_R$$

The construction is to be done with all positions and forces specified as two-dimensional vectors. Weights and counter-force are to be represented as arrows in the xy -plane.

The x -coordinate x_C of the equilibrium point, where the counter-force has to be applied to hold the mobile in equilibrium, is calculated as a weighted average with the masses as weights:

$$x_C = \frac{m_L \cdot x_L + m_R \cdot x_R}{m_L + m_R} \quad (5.1)$$

5.8.2 Data Structure and Nomenclature

<u>L</u>	left arm of the crossbeam
<u>R</u>	right arm of the crossbeam
x_L, x_R	length of the arms of the crossbeam from $x = 0$
m_L, m_R	masses at the crossbeam
<u>w_L</u>	weight at the left arm
<u>w_R</u>	weight at the right arm
<u>CoG</u>	center of gravity ($x_C, 0$)
x_C	horizontal position of center of gravity, from $x = 0$
<u>w_{Anti}</u>	upward counter force applied at <u>CoG</u> .

All underlined entities are two-dimensional vectors represented in the programs with an underscore at the end, e.g. $L_$.

5.8.3 Python Program

In Table 5.15, the horizontal positions (distance to $x = 0$) of the end of the arms of the crossbar on the horizontal $y = 0$ and the forces are specified.

In Table 5.16, the arrows representing the weights at the crossbar's ends and the counter-force to be applied at the equilibrium point are plotted with *ArrowP* from Sect. 5.2.4. A diagram like the one in Fig. 5.23a results (lines 16–18 ignored).

Table 5.15 Specifying the vectors (arms of the crossbar, weights and counter-force) of the mobile in Fig. 5.23a

1	<i>#Scalars</i>	7	<i>#Vectors</i>
2	<code>xL = -0.25</code>	8	<code>L_ = np.array([xL,0])</code>
3	<code>xR = 0.274</code>	9	<code>R_ = np.array([xR,0])</code>
4	<code>mL = 0.88</code>	10	<code>CoG_ = np.array([xC,0])</code>
5	<code>mR = 0.33</code>	11	<code>wL_ = np.array([0,-mL])</code>
6	<code>xC = (mL*xL+mR*xR) / (mL+mR)</code>	12	<code>wR_ = np.array([0,-mR])</code>
		13	<code>wAnti_ = np.array([0,mL+mR])</code>

Table 5.16 Plotting the vectors of Table 5.15 as arrows in the plane of Fig. 5.23a

```

14 FigStd('x',-0.5,0.5,0.5,'y',-1,2,1)
15 plt.plot([xL,xR],[0,0],'k-')
16 plt.plot(*(L_+wL_), 'kd', fillstyle="none")
17 plt.plot(*(R_+wR_), 'kd', fillstyle="none")
18 plt.plot(*(CoG_+wAnti_), 'ks', fillstyle="none")
19 ArrowP(L_,L_+wL_,hw=0.03)
20 ArrowP(R_,R_+wR_,hw=0.03)
21 ArrowP(CoG_,CoG_+wAnti_,hw=0.03)

```

Questions

concerning Table 5.16:

What are the arguments transferred by `*(L_+wL_)` to `plt.plot(...)`?³⁰

What do the instructions in lines 16–18 do (result not appearing in Fig. 5.23a).³¹

5.8.4 Spreadsheet Calculation

A spreadsheet calculation corresponding to the Python program in Table 5.15 is laid out in Fig. 5.24 (S). All vectors are specified as column vectors with two coordinates, one for the x and the other for the y direction. In the lower half from left to right, we have the arms \underline{L} and \underline{R} of the crossbar, the arrows representing the weights at the left and the right end, and the counter-force applied at the center of gravity.

³⁰ $\underline{L}_+ + \underline{wL}_ = [xL, -mL]$ (an array), $*[xL, -mL] = xL, -mL$ (the elements of an array).

³¹ The instructions plot diamonds at the tips of the weights and a square at the tip of the counter-force.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	xL	-0.25													{=-(wL_+wR_)}
2	xR	0.274													
3	mL	0.88													
4	mR	0.33													
5															
6	xC	-0.107 =(mL*xL+mR*xR)/(mL+mR)													
7															
8															
9															
10	x														
11	y														
12															

Fig. 5.24 (S) Defining the elementary vectors and calculating the anti-force

Question

How do you enter the three arrows into a figure as in Fig. 5.23a?³²

5.9 Systems of Linear Equations

We treat two problems resulting in a system of linear equations solved with matrix inversion: calculating the standard form of a polynomial of the 3rd degree from four points and determining the currents in a network of resistances, voltage, and current sources with Kirchhoff’s rules.

5.9.1 Polynomial and Electrical Network

Polynomial

A third-degree polynomial is specified by 4 coefficients:

$$y = f(x) = a + b \cdot x + c \cdot x^2 + d \cdot x^3 \tag{5.19}$$

They can be calculated if 4 points (x^P_i, y^P_i) are specified in a plane, leading to 4 linear equations:

$$y_i = a \cdot 1 + b \cdot x_i + c \cdot x_i^2 + d \cdot x_i^3 \quad i = 1, 2, 3, 4 \tag{5.20}$$

Two examples are shown in Fig. 5.25.

³² G10:N10 as an X-series and G11:N11 as a Y-series.

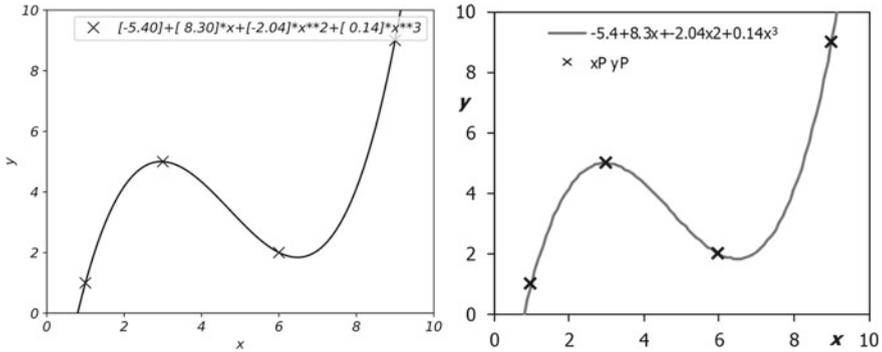


Fig. 5.25 3rd-order polynomials generated, **a** (left) with the Python program in Sect. 5.9.4 and **b** (right) with the EXCEL spreadsheet in Sect. 5.9.3

Matrix of powers of x

The system of linear equations is put into matrix form, with the matrix \underline{P} composed of the powers (0, 1, 2, 3) of x_i (“matrix of powers”) and the known y_i as a vector on the right side of the equation:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ & & \cdot & \\ & & \cdot & \\ 1 & x_4 & x_4^2 & x_4^3 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \tag{5.21}$$

$$\underline{P} \cdot \underline{coeff} = \underline{y}_P$$

The vector of unknowns contains the coefficients: $[a, b, c, d]$. The coefficients may be obtained by applying \underline{P}^{-1} , the inverse of \underline{P} , to \underline{y}_P :

$$\underline{coeff} = \underline{P}^{-1} \cdot \underline{y}_P \tag{5.22}$$

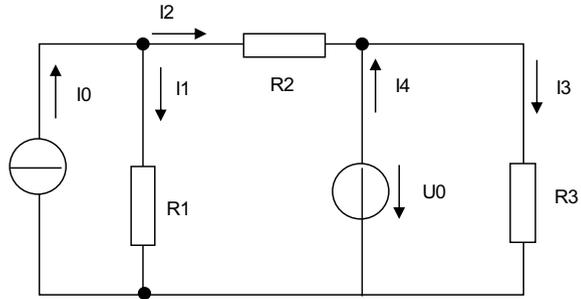
The matrix of powers has to be non-singular; otherwise, its inverse cannot be built. A square matrix is nonsingular if and only if its determinant is nonzero. The determinant is obtained with MDET (EXCEL) or `np1.det` (Python).

Question

What is the determinant ($< 0, = 0, > 0$) of the matrix of powers if two x values of the four defining points are identical?³³

³³ The determinant is zero because it is impossible to draw a polynomial through two points when one is above the other one.

Fig. 5.26 Electrical circuit comprising a current source I_0 , a voltage source U_0 , and three ohmic resistors R_1 , R_2 , and R_3



Electrical Networks

In Fig. 5.26, an electrical network with three ohmic resistors R_1 , R_2 , and R_3 , a constant current source I_0 , and a constant voltage source U_0 is shown. We are going to calculate the currents I_1 , I_2 , I_3 , and I_4 in the different branches. The directions of these currents (i.e., the directions of the arrows in the figure) are arbitrary, but they must be chosen at the very beginning. The signs of the currents are with respect to these directions.

Kirchhoff's rules as matrix equation

The four linear equations for the four unknown currents I_1 , I_2 , I_3 , and I_4 are obtained with the help of Kirchhoff's rules, namely, with two mesh rules (the voltage around a closed circuit must be zero):

$$R_3 \cdot I_3 - U_0 = 0 \text{ and } -R_1 \cdot I_1 + R_2 \cdot I_2 + U_0 = 0 \quad (5.23)$$

and two junction rules (the currents flowing into a junction must equal the currents flowing out of the junction):

$$-I_1 - I_2 + I_0 = 0 \text{ and } I_2 + I_4 - I_3 = 0 \quad (5.24)$$

These four equations are transformed so that the known source voltage U_0 (or 0) is on the right side of the mesh rules and the known source current I_0 (or 0) is on the right side of the junction rules so that they can be expressed as a matrix equation:

$$\underline{Res} \cdot \underline{I} = \underline{Srcs} \quad (5.25)$$

\underline{Res} is a matrix with components 0 or 1 or the resistances of the circuit, \underline{I} is the vector of the unknown currents, and \underline{Srcs} is a vector whose coefficients are zero or the known source voltages and currents. An example for the network in Fig. 5.26 is given in the spreadsheet of Fig. 5.27 (S).

Questions

concerning Fig. 5.27 (S):

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2				1	1	0	0		1.1		1.0			
3				0	0	R.3	0	▪	1.2	=	U.0			
4				-R.1	R.2	0	0		1.3		-U.0			
5				0	1	-1	1		1.4		0			
6														
7	I.0	0.5 A	Res						Currents		Srcs			
8	U.0	5 V	1	1	0	0			1.1		0.5			
9	R.1	100 Ω	0	0	90	0		▪	1.2	=	5			
10	R.2	68 Ω	-100	68	0	0			1.3		-5			
11	R.3	90 Ω	0	-1	1	-1			1.4		0			
12														

Fig. 5.27 (S) Eq. (5.25) applied to the network of Fig. 5.26. The matrix equation is presented in rows 1–5 in general form and in rows 8–11 with the concrete values from column B. The vertical lines, and the dots in column H, are intended to indicate matrix calculation. They have no computational function in the spreadsheet

Verify that the matrix calculation in rows 8–11 is identical to Eqs. 5.23 and 5.24!

What are the formulas in D10:G10?³⁴

5.9.2 Data Structure and Nomenclature

Polynomial

- x_p, y_p coordinates of 4 points in the plane, specified as two vectors
- \underline{A} 4×4 matrix (“matrix of powers of x ”), powers of x_p in rows
- \underline{A}_{Inv} the inverse of matrix A
- \underline{coeff} solution a, b, c, d of the system of linear equations, coefficients of the polynomial

Electrical circuit

- R_1, R_2, R_3, R_4 four ohmic resistances
- I_0 current of a constant-current source
- U_0 voltage of a constant-voltage source
- \underline{Res} square matrix containing resistances, zeros, and ones
- \underline{ResInv} inverse of Res
- \underline{Srcs} vector containing the voltage and current sources or zero
- $\underline{Currents}$ vector containing the currents to be determined with the system of linear equations.

³⁴ D10 = [=R.1], E10 = [=R.2], F10 = [0], G10 = [0].

We need functions to build the inverse of a square matrix and to multiply matrices.

EXCEL: MINVERSEMMULT
 Python: np.inv@ operator (or np.matmul)

5.9.3 Spreadsheet Solutions

Polynomial

The solution of Eq. 5.22 is implemented in the spreadsheet of Fig. 5.28 (S). The coordinates of the defining points are stored in column vectors named \underline{x}_P and \underline{y}_P . The matrix \underline{A} is composed of column vectors obtained as powers of \underline{x}_P . To solve for the coefficients of the polynomial, we have to build the inverse \underline{A}_{inv} of \underline{A} to be applied to \underline{y}_P in a matrix multiplication.

Electrical network

For the network of Fig. 5.26, the spreadsheet layout in Fig. 5.27 (S) applies, in rows 2–4 in general form and in rows 8 to 11 with the values from A7:B11. The indices of the resistances and the constant current and constant voltage are separated from the letters by a dot, because a name like R1 would be interpreted as a cell address.

The unknown currents are obtained by forming the inverse matrix to \underline{M} to be applied to the vector \underline{S} of sources:

$$\underline{M}^{-1} \cdot \underline{M} \cdot \underline{I} = \underline{M}^{-1} \cdot \underline{S} \text{ or } \underline{M}^{-1} \cdot \underline{S} = \underline{I}$$

The solution for the matrix equation in Fig. 5.27 (S), applying the EXCEL matrix functions MINVERSE to get the inverse matrix and MMULT to perform matrix multiplication, is given in Fig. 5.29 (S) where, in column N, we check whether Kirchhoff's rules are satisfied.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
2																
3		\underline{x}_P	\underline{y}_P		$=x^P^3$	$=x^P^2$	$=x^P$	1		{=MINVERSE(A)}						
4				A					Ainv					coeff		
5		1	1	1	1	1	1		-0.01	0.03	-0.02	0.01		0.14	dd	
6		3	5	27	9	3	1		0.23	-0.44	0.29	-0.07		-2.04	cc	
7		6	2	216	36	6	1		-1.24	1.92	-0.87	0.19		8.30	bb	
8		9	9	729	81	9	1		2.03	-1.50	0.60	-0.13		-5.40	aa	

Fig. 5.28 (S) Spreadsheet implementation of Eq. 5.22; \underline{A}_{inv} is the inverse matrix of \underline{A} ; the resulting polynomial is presented in Fig. 5.25b

	C	D	E	F	G	H	I	J	K	L	M	N	O
12													
13	{=MINVERSE(Res)}						{=MMULT(ResInv;Srcs)}						
14	ResInv				Srcs				Cur				
15	0.40	0.00	-0.01	0.00	1.0	0.23	1.1	-5.00	=-R.1*1.1+R.2*1.2				
16	0.60	0.00	0.01	0.00	U.0	0.27	1.2	-0.50	=-1.1-1.2				
17	0.00	0.01	0.00	0.00	-U.0	0.06	1.3	5.00	=R.3*1.3				
18	-0.60	0.01	-0.01	-1.00	0	-0.21	1.4	0.00	=1.2+1.4-1.3				
19													

Fig. 5.29 (S) Solving the matrix equation in Fig. 5.27 (S) with the inverse of Res. In column N, Kirchhoff's rules are checked

Table 5.17 Calculating the coefficients *coeff* of a polynomial of the 3rd order when 4 points (in arrays (*x_p*, *y_p*)) are given

<pre> 1 import numpy.linalg as npl 2 xP=np.array([1,3,6,9]) #Defining points 3 yP=np.array([[2,6,2,2]]) 4 xP0_ = np.array([1,1,1,1]) 5 xP1= xP 6 xP2= xP**2 7 xP3= xP**3 8 9 A=np.array([xP0_,xP1,xP2,xP3]).transpose(1,0) 10 yPt_ = yP.transpose(1,0) 11 coeff_ = npl.solve(A,yPt) </pre>		
A	coeff	yPt
[[1 1 1 1]	[[-4.00]	[[2]
[1 3 9 27]	[7.67]	[6]
[1 6 36 216]	[-1.78]	[2]
[1 9 81 729]]	[0.11]]	[2]]

Question

The calculations in column N (reported in column O) of Fig. 5.29 (S) are intended to check whether Kirchhoff's rules are fulfilled. Which of the four equations Eq. 5.23 left or right and Eq. 5.24 left or right are checked in N15–N18?³⁵

5.9.4 Python Programs

Polynomial from defining points

In Table 5.17, the coordinates of the defining points are specified in the arrays *x_p* and *y_p*. The matrix *A* is put together in line 9, first as a list of the rows *x_{p0}*, *x_{p1}*, *x_{p2}*, *x_{p3}* that is then transposed to get a form as in Eq. (5.21); see also the printout in the lower cell. Likewise, the vector *y_p* is generated. In line 11, the system of linear equations

³⁵ N15: Eq. 5.23 right; N16: Eq. 5.24 left; N17: Eq. 5.23 left; N18: Eq. 5.24 right.

Table 5.18 Python program for plotting the polynomial specified by the coefficients in Table 5.17

```

1 x=np.linspace(0,10,100)
2 y=coeff[0]*x**3+coeff[1]*x**2+coeff[2]*x+coeff[3]
3 FigStd('x',0,10,2,'y',0,10,2)
4 plt.plot(x,y,'k-')
5 plt.plot(xP,yP,'kx')      #Defining points of the parabola

```

is solved with the function `solve` of the `numpy.linalg` library. Alternatively, the coefficients can be obtained by applying the inverse of the "powers of x" matrix to the vector of pre-specified y values:

$$\underline{coeff}_2 = \underline{P}_{\text{Inv}} @ \underline{y}_p$$

The results of the two methods are identical. The coefficients are used to generate the coordinates of the polynomial shown in Fig. 5.25a. The entities in the lower cell of Table 5.17 are arranged so that they correspond to the equation:

$$\underline{P} \cdot \underline{coeff} = \underline{y}_{\text{Pt}}$$

The entity $\underline{y}_{\text{Pt}}$ is the transposed row vector \underline{y}_p . At least the first coefficient of $\underline{y}_{\text{Pt}}$ can be checked by mental calculation.

Questions

The coefficients in Fig. 5.28 (S) are the same as in the Python solution in Table 5.17, however, in reverse order. What is the reason for this?³⁶

What is the more versatile version: (a) $y = \text{coeff}[0]*x**3 + \dots$ or (b) $y = \text{coeff}[0] + \text{coeff}[1]*x$?³⁷

The program for realizing the plot in Fig. 5.25a is shown in Table 5.18.

Electrical network

A Python program for solving Kirchhoff's equations corresponding to the spreadsheet layout in Fig. 5.27 (S) and Fig. 5.29 (S) is presented in Table 5.19. The matrix \underline{Res} is defined as in the spreadsheet in Fig. 5.27 (S), whereas the vector of the sources is first defined as a row vector \underline{Src} and then transposed into a column vector \underline{Srcs} equal to the one in Fig. 5.27 (S) and to \underline{Srcs} in column K of Fig. 5.29 (S). Lines 9 and 10 can be combined into one instruction.

³⁶ The matrix of powers \underline{P} in Fig. 5.28 (S) contains the columns in reverse order to Table 5.17.

³⁷ Version (b) is more versatile, because it can easily be extended to higher order, $\dots + \text{coeff}[3]*x**4 + \dots$

Table 5.19 Solving Kirchhoff’s equation

<pre> 1 import numpy.linalg as npl 2 R1,R2,R3=100,68,90 3 I0,U0=0.5,5 4 Res=np.array([[1,1,0,0], 5 [0,0,R3,0], 6 [-R1,R2,0,0], 7 [0,-1,1,-1]]) 8 Src=np.array([[I0,U0,-U0,0]]) </pre>	<pre> 9 Srcs=Src.transpose(1,0) 10 ResInv = npl.inv(Res) 11 Cur=ResInv@Srcs 12 CurS=npl.solve(Res,Srcs) </pre>
<p>ResInv:</p> <pre> [[0.40 -0.00 -0.01 -0.00] [0.60 0.00 0.01 0.00] [0.00 0.01 0.00 0.00] [-0.60 0.01 -0.01 -1.00]] </pre>	
<p><i>Cur with ResInv (inverse matrix)</i></p> <pre> [[0.23] [0.27] [0.06] [-0.21]] </pre>	<p><i>CurS with linalg (solve)</i></p> <pre> [[0.23] [0.27] [0.06] [-0.21]] </pre>

The system of linear equations is solved in two ways, in line 11 with the inverse matrix *ResInv* as in the spreadsheet calculation and in line 12 with the function `solve` of the `numpy.linalg` library. Both methods yield the same result (second cell in Table 5.19) as the spreadsheet procedure.

5.10 Some Mathematical Functions

Numpy, np.*	EXCEL	
<i>Basic numeric information</i>		
abs	ABS(x)	absolute value
sign	SIGN(x)	sign (+1, -1 or 0)
<i>Basic mathematical operations</i>		
sum	SUM(x; x ₂)	$x + x_2$ (no matrix function)
prod	PRODUCT(x;x ₂)	$x \cdot x_2$ (no matrix function)
	QUOTIENT(x;x ₂)	integer portion of x/x_2 , no matrix function
mod	(x; x ₂)	Modulo, remainder from x/x_2
sqrt	SQRT(x)	Positive square root
<i>Rounding functions</i>		
round	ROUND(x; N)	Up or down, to N digits
	ROUNDUP(x; N)	Up to bigger absolute value, to N digits
	ROUNDDOWN(x; N)	Down to smaller absolute value, to N digits
	MROUND(x; x ₂)	Up or down to a multiple of x_2 , no matrix function

Numpy, np.*	EXCEL	
Ceil	CEILING(x; x ₂)	Up versus ∞ to a multiple of x ₂
Floor	FLOOR(x; x ₂)	Down versus −∞ to a multiple of x ₂
Rint	INT(x)	Down to the next integer
fix	TRUNC(x)	Towards zero to the next integer
<i>Trigonometric functions</i>		
pi	PI()	Constant value of pi
rad2deg	DEGREES	Converts radians to degrees
deg2rad	RADIANS	Converts degrees to radians
cos	COS(α)	Cosine of a given angle (rad)
arccos	ACOS	Inverse cosine [−1, 1] → [0, π]
sin	SIN(α)	Sine of a given angle (rad)
arcsin	ASIN	Inverse sine [−1, 1] → [−π/2, π/2]
tan	TAN(α)	Tangent of a given angle (rad)
arctan	ATAN	inverse tangent (<i>arcus tangens</i>) [−∞, ∞] → [−π/2, π/2]
arctan2 (y, x)	ATAN2(x;y)	Angle of a given pair of x and y coordinates; attention: Order of arguments is different in EXCEL and Python!
<i>Exponents and logarithms</i>		
Exp	EXP(x)	e raised to the power of x
Log	LN(x)	Natural logarithm, inverse of exp
	LOG(x;b)	Logarithm of x to base b LOG(X; B ₁) = LOG(X; B ₂) * LOG(B ₂ ; B ₁)
log10	LOG10	Base 10 logarithm
log2	LOG(x; 2)	Base 2 logarithm
<i>Sums</i>		
Footnote 38	SUMIF	Adds the cells in a supplied range that satisfy a given criterium
Footnote ^a	SUMIFS	Adds the cells in a supplied range that satisfy multiple criteria
sum(x * y)	SUMPRODUCT(x;y)	x * y over arrays
sum(x * x)	SUMSQ(x)	x ² over arrays
sum(x ² −y ²)	SUMX2MY2(x;y)	x ² minus y ² over arrays
sum(x ² + y ²)	SUMX2PY2(x;y)	x ² plus y ² over arrays
sum((x−y) ²)	SUMXMY2(x;y)	(x − y) ² r arrays
Cumsum	Footnote ^b	Cumulated sum
<i>Functions of linear algebra</i>		
np.linalg.det	MDETERM	Determinant of a square matrix
np.linalg.inv	MINVERSE	Inverse of a square matrix
np.matmul (A, B) or A@B	(A;B)	Matrix product of two matrices

Numpy, <code>np.*</code>	EXCEL	
<code>np.dot(x,y)</code>	SUMPRODUCT(x;y)	Dot product of two vectors

^a Can be achieved with list comprehension, as in the following example: `Rng = np.linspace(1,20,20)`; `x = [x1 for x 1 in Rng if x1 > 3 if x 1 < 6]`; `sum(x)`

^b In a spreadsheet operation, `B1 = [=A1]`; `B2 = [=B1 + A2]`; ... cumulates in column B the values in column A

5.11 Questions and Tasks

1. How do you get the anchor points of the inverse function of $y = y(x)$?³⁸

Differentiation and integration

2. You have calculated the difference quotient between the boundaries of an interval. Over which value do you plot the result?
3. What is the formula for the numerical second derivative of a function, specified at equidistant positions with distance dx ?
4. Over which positions of an interval (beginning, middle, end) are values for the first derivative, the second derivative, and the integral to be plotted?
5. What does the trapezoidal rule of integration stand for?

Vectors

6. What are the Cartesian coordinates of vectors of length 1 (one) pointing in the x direction, the negative y direction, and 45° from the x -axis?
7. What are the components of a vector pointing from $(\underline{x}_1) = (4, 5)$ to $(\underline{x}_2) = (6, 3)$?
8. What are the lengths of the two vectors pointing from $(0, 0)$ to $(1, 1)$, and from $(0, 0)$ to $(3, 4)$?
9. How do you form a scalar product of two vectors \underline{A} and \underline{B} in two different ways, (i) component by component, and (ii) using the angle ϕ between the two vectors?
10. What is the angle between two vectors if the scalar product is zero?
11. What are the coordinates of a unit vector in the direction of the segment from $[x_1, y_1]$ to $[x_2, y_2]$?
12. What are the coordinates of the perpendicular to the vector (x, y) ?

³⁸ The inverse function is $x = x(y)$. To get its interpolation points, you only have to swap the two columns for x and y in the spreadsheet or the two lists in Python.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	A			B			D															
2	1	0	1		1	0	0		1	2	3	=MMULT(E2:G4;I2:K4)										
3	0	1	0		0	1	0		4	5	6											
4					0	0	1		7	8	9				=MMULT(A2:C3;I2:K4)							
5																						

Fig. 5.30 (S) Five matrix ranges, named cell ranges: $A = [A2:C3]$, $B = [E2:G4]$, $C = [I2:K4]$, $D = [I2:J4]$, $x = [I2:K2]$, $y = [I3:K3]$

System of equations

- 13. Which arguments does the EXCEL function MMULT(?) have, which multiplies two matrices? What is the equivalent in Python? What are the relations between the widths and heights of the two matrices?
- 14. You have set up a system of equations $\underline{M} \cdot \underline{I} = \underline{I}$, with known matrix \underline{M} and known source vector \underline{S} , but unknown vector \underline{I} . By which instructions of type matrix operation do you get the coefficients of \underline{I} in EXCEL and Python?³⁹

Functional expressions

Calculate the values of the following expressions in EXCEL:

- 15. ARCTAN2(1;1), LOG10(0.001),
- 16. PRODUCT(2;3;4;5), POWER(10;3).
- 17. ROUND(3.74638,2), INT(17.453), REMAINDER(127; 2).

The functions in Questions 18 and 19 refer to Fig. 5.30 (S). What values result for:

- 18. SUMPRODUCT(x;y) , SUMXMY2(B;C)? What are the corresponding expressions in Python?
- 19. MMULT(B;C), MMULT(A;D)? What are the corresponding expressions in Python?
- 20. You are to determine the derivative dy/dx of a function defined by $x = np.linspace(0, 10, 11)$ and $y = x**2$. How do you determine dy and dx with list slicing? What are the first two elements of dy/dx, and over which values do they have to be plotted?
- 21. Define the arrays \underline{A} , \underline{B} , and \underline{C} , in Python corresponding to the three ranges A2:C3, E2:G4, and I2:K4, respectively, in Fig. 5.30 (S)! Calculate $\underline{B}@\underline{C}$ and $\underline{A}@\underline{C}$!
- 22. What are the two arguments in the function to calculate the angle of the vector (1, 2) to the x axis in EXCEL and Python?

³⁹ EXCEL: $I = MMULT(M;S)$. Python: $I = npl.inv(M)@S$; npl stands for numpy.linalg, M and S are of type np.array.

Superposition of Movements

6

We learn how to compose complicated movements from simple ones, namely translations and rotations in a plane. The exercises honor famous scientists: Bernoulli (cycloid), Foucault (pendulum), and Steiner (moment of inertia). In spreadsheets, we systematically use sliders and macros with which we have familiarized ourselves in previous chapters.

6.1 Introduction: Translations and Rotations

Solutions of Exercises 6.2 (Python), 6.3 (Excel), 6.4 (Excel), and 6.5 (Python) can be found at the internet address: go.sn.pub/or1CXF.

Simple movements

In this chapter, we put together movements in a plane from two simple movements:

Translations T , straight-line movements in one direction, generally defined by a two-dimensional velocity vector (v_x, v_y) ;

Rotations R , rotations in the xy -plane, described by an angular velocity ω_z and the radius r of the trajectory.

Polar coordinates

We use polar coordinates (r, ϕ) to describe rotations and convert them, e.g., for graphical representation in charts, to Cartesian coordinates (x, y) :

$$x = r \cdot \cos(\phi); y = r \cdot \sin(\phi) \quad (6.1)$$

Projectile motion, T-T (Exercise 6.2)

A projectile trajectory is composed of two linear movements. If friction is not taken into account, these are a vertical one accelerated by gravity and a uniform horizontal

one. We attach a velocity vector and its vertical and horizontal component to the trajectory at a freely chosen point.

Cycloid, rolling curve, T-R (Exercise 6.3)

We calculate the trace of a point on a wheel rolling on a plane, resulting from a translation of the wheel axis on a straight line and a rotation about the wheel axis, with translation speed and angular speed being dependent on each other.

The resulting curve, called a *rolling curve* or *cycloid*, represents the brachistochrone (the fastest path to fall from one point to a lower point), as shown by Johann Bernoulli. This will be treated in the follow-up book *Physics with Excel and Python, Using the Same Data Structure. Applications* in the chapter “Calculus of Variations”.

Foucault’s pendulum, T-R (Exercise 6.4)

What is the trace of a swinging pendulum on a rotating surface? It is obtained by superposing the movement of the linear oscillation in the laboratory system with a rotation of the base table on which the motion is recorded. Oscillation and rotation are independent of each other.

This experiment has historical significance. Michel Foucault demonstrated with a 67-m long pendulum suspended from the dome of the Pantheon in Paris that the earth rotates against the fixed stars.

Swinging anchor, R-R (Exercise 6.5)

We consider an anchor in the form of a hanging T with three mass points attached, one at each end of the T and one at the junction of the two lines. The mass points are supposed to be connected by massless struts. We hang the rigid anchor at the upper end of the stem or hold it at its center of gravity.

The anchor’s motion results from a superposition of the rotation of a selected point of the anchor, e.g., the center of gravity, around the suspension point, and a rotation of the anchor around the selected point. A rotational matrix describes this motion.

We calculate the center of gravity of the anchor and, with Steiner’s theorem,¹ its moment of inertia when rotating around the upper end of the stem and when rotating around its center of gravity.

Sound emitted from a moving source, T-T (Exercise 6.6)

In the last exercise of this chapter, we investigate the circular wavefronts emitted from a moving acoustic source.

¹ Also known as the *parallel axis* theorem or *Huygens-Steiner* theorem.

Animations

The movements treated in this chapter are well suited to be animated. The basic technique for this with `FuncAnimation` of the `matplotlib.animation` library is explained in Sect. 6.2.5 in connection with the projectile trajectory.

6.2 Projectile Trajectory with Velocity Vectors (T-T)

We calculate and plot the trajectory of a projectile composed of two linear motions (T), a uniform horizontal one and a vertical one accelerated by gravity. The parameters are launch height, angle, and speed. Vectors for the horizontal, vertical, and total velocities are attached at three points to the trajectory.

6.2.1 Projectile Trajectory and Velocity Vectors

Trajectory and attached arrows

In Fig. 6.1a, a projectile trajectory is shown with arrows representing velocity vectors attached for three different time points. The parameters are launch height and angle, and speed. In Fig. 6.1b, two trajectories for different launch angles are displayed. The trajectories are, in all cases, downwardly open parabolas. Figure 6.1b shows that maximum height and maximum distance depend on the launch angle.

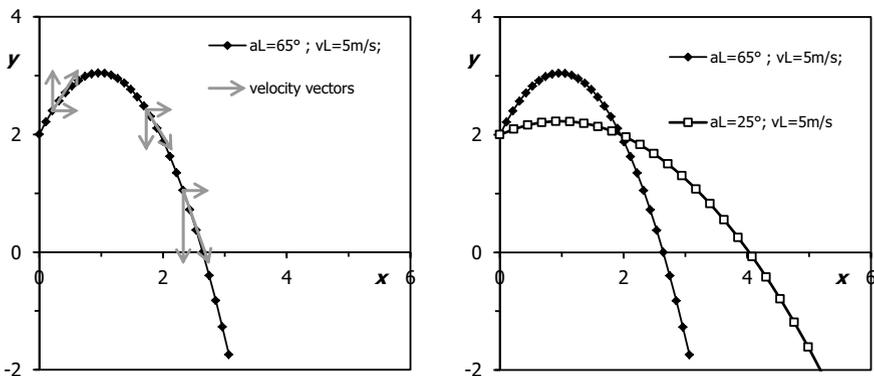


Fig. 6.1 **a** (left) Projectile trajectory, coordinates at different equidistant times; velocity vectors at three different times. **b** (right) Two trajectories for two different launch angles

Our task is to plot a projectile trajectory with given values for launch height y_L , angle α_L , and velocity v_L . In Fig. 6.1a, we have chosen $y_L = 2$ m, $\alpha_L = 65^\circ$, and $v_L = 5$ m/s. The time distance dt between the calculation points is 0.05 s.

Coordinates of the projectile trajectory

The coordinates of the parabola are calculated from

$$\begin{aligned} v_x &= v_{Lx} \quad \text{with} \quad v_{Lx} = v_L \cdot \cos(\alpha_L); \\ v_y &= v_{Ly} - g \cdot t \quad \text{with} \quad v_{Ly} = v_L \cdot \sin(\alpha_L) \end{aligned} \quad (6.2)$$

as

$$x(t) = v_{Lx} \cdot t \quad (6.3)$$

$$y(t) = y_L + v_{Ly} \cdot t - \frac{1}{2}g \cdot t^2 \quad (6.4)$$

In the x -direction, there is a uniform motion with the horizontal initial velocity v_{Lx} . The movement in the y -direction is the sum of a uniform motion with velocity v_{Ly} , determined by the initial velocity and a downward motion accelerated by gravity.

► **Tasks** For a given speed and launch height, change the angle such that (a) the height and (b) the width reached will be maximum!

Calculate the impact velocity for the two trajectories in Fig. 6.1b!

Determine the maximum height and width analytically and compare them with the value of the simulation!

Attach the tangential vector of the velocity and the decomposition of this vector into x and y components at the point of the trajectory corresponding to a specific time $t = tt$! The arrow representing the velocity vector (v_{Xtt} , v_{Ytt}) is drawn in the xy -plane

$$\text{from } (x_{tt}, y_{tt}) \text{ to } (x_{tt}, y_{tt}) + (V_{Xtt}, V_{Ytt}) \cdot \text{scal} \quad (6.5)$$

The lengths of the arrows in the figure are to be adapted to the diagram with a scaling factor $scal$.

6.2.2 Data Structure and Nomenclature

y_L, α_L, v_L	launch height, angle, speed
v_{Lx}, v_{Ly}	horizontal and vertical components of v_L
t	equidistant (dt) series of instants of time
$x(t), y(t)$	trajectory as a function of t
tt	list of three specific times
x_{tt}, y_{tt}	position at tt

	A	B	C	D	E	F		B	C	D	E	F
1	Time interval	dt	0.05 s				11	="aL="&aL&"^o ; vL="&vL&"m/s;"				
2	Prespecifications						12	aL=65° ; vL=5m/s;				
3	Launch height	yL	2 m				13	=B15+dt =vLx*t =yL+vLy*t-g/2*t^2				
4	Launch angle	aL	65 °				14	t	x	y		
5	Launch speed	vL	5.00 m/s				15	0	0.00	2.00		
6	Gravitational acceleration	g	9.81 m/s ²				16	0.05	0.11	2.21		
7	Deduced therefrom						44	1.45	3.06	-1.74		
8		vLx	2.11 m/s									
9		vLy	4.53 m/s									

Fig. 6.2 a (left, S) The parameters for the task are defined and used to calculate the horizontal, v_{Lx} , and vertical, v_{Ly} , initial velocities. The launch angle α_L is set with a slider. **b** (right, S) Continuation of **a**. Coordinates of the parabola (t, x, y) in equidistant time steps dt in columns B, C and D; the label for the figure is generated in B12 with the formula reported in B11

- v_{Xtt}, v_{Ytt} velocity vectors at tt (3 instances)
- $scal$ scaling factor [s] for the velocity vectors in the xy -plane
- x_{Att}, y_{Att} coordinates of the tips of the arrows representing the vectors.

6.2.3 Spreadsheet

Trajectory

A possible spreadsheet calculation is shown in Fig. 6.2 (S).

The freely selectable parameters initial height, angle, and speed of the launch are in C3:C6. From this, the initial horizontal and vertical velocities, v_{Lx} , and v_{Ly} , are calculated in C8:C9.

Questions

- Which formulas are in C8 and C9 of Fig. 6.1a (S)?²
- How is the legend “aL = ...” in Fig. 6.2b (S) generated?³

Velocity-vector coordinates

► **Task** Calculate the x - and y -components of the velocity for a given time tt ! In Fig. 6.3 (S), this is done for $tt = 0.818$ s, a time set with a slider.

In Fig. 6.3a, the coordinates of the arrows representing the velocity vector at $t = tt$ and its horizontal and vertical components are calculated according to Eq. 6.2. This is repeated in Fig. 6.3b for two other time points. The resulting arrows are displayed in Fig. 6.1a. As of EXCEL 2007, line segments in charts can be provided with arrowheads (EXCEL 1019: FORMAT DATA SERIES/FILL & LINE/END ARROW TYPE), and consequently our vector arrows.

² $v_{Lx} = C8 = [= vL*cos(aL/180*pi())]$; $v_{Ly} = C9 = [= vL*sin(aL/180*pi())]$.

³ See cell B11! Concatenation of text and variables.

	H	I	J	K	L		H	I	J	K	
14	velocity vectors						27	0.1	0.21		2.40
15	scaling factor			scal	0.2		28	0.1	0.63		3.11
16	tt	xAtt		yAtt			29				
17	818						30		0.21		2.40
18	0.818	1.73	=vLx*tt	2.42	=yL+vLy*tt-g/2*tt^2		31		0.63		2.40
19	0.818	2.15	=I18+scal*vLx	1.73	=K18+scal*(vLy-g*tt)		32				
20							33		0.21		2.40
21		1.73	=I18	2.42	=K18		34		0.21		3.11
22		2.15	=I19	2.42	=K18		35				
23							36	1.1	2.32		1.05
24		1.73	=I18	2.42	=K18		37	1.1	2.75		-0.20
25		1.73	=I18	1.73	=K19		38				
26							39		2.32		1.05

Fig. 6.3 a (left, S) Coordinates (in I18:K19) of an arrow representing a velocity vector and its vertical and horizontal components (I21:K25) at time tt , defined in H18; tt , x_{Att} , and y_{Att} are the names for the areas H18:H42, I18:I42, and K18: K42, containing the coordinates of the three vectors in Fig. 6.1a. **b** (right, S) Continuation of **a**. Another velocity vector at another time, this time set directly in H27 without a slider; the formulas are structurally the same as those reported in J18:J25 and L18:L25, but with references to different cells

Questions

concerning Fig. 6.3 (S):

The time at which velocity vectors are calculated and attached to the trajectory is set with a slider. What is the LINKED CELL of this slider, and what are probable MIN and MAX? What are the formulas in H18 and H19?⁴

Change the time tt with the slider so that the height $y = 0$ is reached for the discharge height and speed in Fig.6.1a and a launch angle 65° . At what time does this occur, and at what speed does the projectile hit the ground?⁵

What is the purpose of the quantity $scal$ in the formulas in row 15? How big is it, and what physical unit does it have?⁶

An Excel trick

When you want to specify vectors for several instants of time, you can copy the range H18:K25 in Fig. 6.3a if the formulas are written with relative and absolute cell references so that they remain valid when copied. In Fig. 6.3 b (S), the formulas have been copied into the area H27:K34. Regarding cells H27:H28, corresponding

⁴ In Fig. 6.3a (S), H17 is the cell linked to the slider (LINKED CELL). MIN = 0, MAX = 1500, as can be estimated from the position of the rider and the number in H17. H18 = H17/ 1000; H19 = H18.

⁵ The projectile reaches the ground at $t_1 = 1.25$ s and hits with $v = 8.01$ m/s, calculated with.

$$v = \sqrt{v_{0x}^2 + (v_{0y} - g \cdot t_1)^2}$$

⁶ $Scal = 0.2$ s. This parameter determines the length of the velocity vectors in their representation in the plane (x [m], y [m]); see also Exercise 5.5. It occurs in an equation of the kind x [m] = x_0 [m] + $scal * v$ [m/s]; $scal$ has the unit [s].

to H18:H19, the desired time is entered directly into H27 and H36. As the range H18:K25 has been copied twice, vectors are attached at a total of three points.

6.2.4 Python

Projectile trajectory

In the left cell of Table 6.1, the parameters of the exercise are specified. A label *lbl1* for the legend in a figure is generated. In the right cell, initial horizontal and vertical velocities are determined and the projectile trajectory ($x(t)$, $y(t)$) is calculated.

Drawing arrows that represent vectors

Table 6.2 draws a figure that is similar to Fig. 6.1a. The time instants for the three velocity vectors are specified in list t_2 . The coordinates of the arrows representing these vectors are calculated in a for-loop.

In Python, in order to draw an arrow, we have to make use of the function `plt.arrow`, which requires, among others, the initial position (x_0 , y_0) and the length of the vectors in the x - and y -directions as input. As we prefer to enter begin- and end-points, we have defined a new function *ArrowP*, reported in Table 6.3, with two positional arguments P_0 and P_1 , and some keyword arguments with default values.

ArrowP has two positional arguments, tail point P_0 and head point P_1 , and three keyword arguments, ls = line style, lw = line width, and hw = head width. In the current situation, the default head width is too large and we have specified $hw = 0.1$ in the function calls (lines 30–32 in Table 6.2).

Table 6.1 Projectile trajectory, with the same data structure as in the spreadsheet of Fig. 6.2

1	<code>dt=0.05 #Time interval</code>	8	<code>#Deduced</code>
2	<code>#Prespecified</code>	9	<code>aL=np.pi/180 #Radian</code>
3	<code>yL=2.0 #Launch height</code>	10	<code>vLx=vL*np.cos(aL)</code>
4	<code>aL=65 #Launch angle</code>	11	<code>vLy=vL*np.sin(aL)</code>
5	<code>vL=5.0 #Launch speed</code>		<code>#Projectile trajectory:</code>
6	<code>g=9.81 #Gravit. accel.</code>	12	<code>t=np.arange(0,1.45+dt,dt)</code>
7	<code>lbl1="aL="+str(aL)+";</code>	13	<code>x=vLx*t</code>
	<code>vL="+str(vL)</code>	14	<code>y=yL+vLy*t-g/2*t**2</code>

Table 6.2 Python program for drawing arrows representing velocity vectors at the trajectory; the loop in the right cell corresponds to Fig. 6.3

```

15 FigStd('x',0,6,1,'y',-2,4,1)
16 plt.plot(x,y,'kD-',ms=2,label=lbl1)
17 plt.legend()
18
19 #Velocity vectors at three time instants t2
20 t2=[0.1,0.8,1.1]
21 scal=0.2
22 for tt in t2:
23     vXtt=vLx                                #Constant horizontal velocity
24     vYtt=vLy-g*tt                            #Uniform vertical motion
25     xtt=vXtt*tt
26     ytt=yL+vLy*tt-g/2*tt**2
27     P1=[xtt,ytt]                             #Foot position of arrow
28     Ax=xtt+vXtt*scal                         #Tip position of arrow
29     Ay=ytt+vYtt*scal
30     ArrowP(P1,[Ax,Ay],hw=0.1)
31     ArrowP(P1,[Ax,ytt],hw=0.1)
32     ArrowP(P1,[xtt,Ay],hw=0.1)

```

Table 6.3 User-defined function for drawing arrows

```

1 def ArrowP(P0, P1, c="k", ls='-',lw=1, hw=0.4):
2     #c has to be given as c="k", not c='k' (2020)
3     (x0,y0)=P0
4     (x1,y1)=P1
5     plt.arrow(x0,y0,x1-x0,y1-y0,
6             length_includes_head=True,
7             head_width=hw, fill=False,
8             linestyle=ls, color=c, linewidth=lw)

```

6.2.5 Animation of Figures with FuncAnimation⁷

We are going to set up an animated version of Fig. 6.1a by extending the program presented in Table 6.1 that provides all data $x(t)$, $y(t)$, $v_x(t)$, and $v_y(t)$ that are accessed in the following program as global arrays.

Creating a figure and a subplot object

In Table 6.4, the sublibrary *animation* is imported from matplotlib. A figure object *fig* is set up and its default font size is set to 7 points (lines 4 and 5). In general, an array of $r \times c$ subplots can be introduced into the frame of a figure. The instruction is `add_subplot(rcn)` with r and c specifying the number of rows and columns. The index n indicates the individual subplot, starting at 1 in the upper left

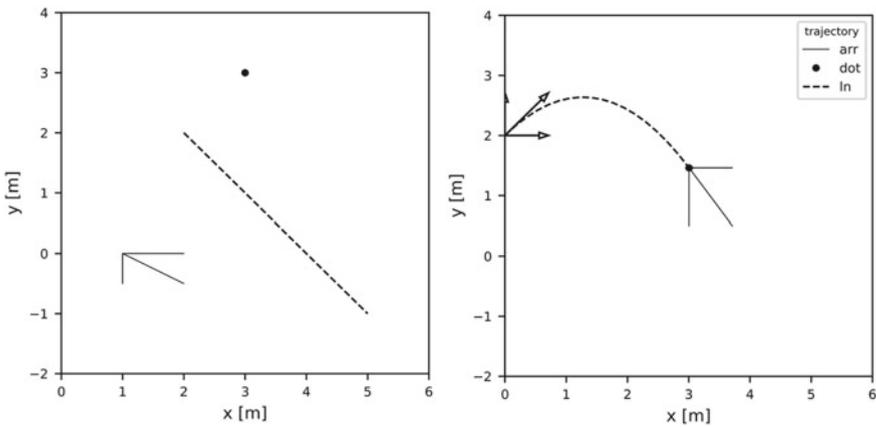
⁷ Matplotlib.pyplot.subplots—Matplotlib 3.4.1 documentation

Table 6.4 Setting up a figure and a subplot object

```

1  %matplotlib notebook
2  import matplotlib.animation as animation
3  cm = 1/2.54 #Centimeter in inches
4  fig=plt.figure(figsize=(9*cm,9*cm))
5  plt.rcParams.update({'font.size': 7})
6  ax=fig.add_subplot(111) #1 row, 1 column, 1=top left
7  ax.set_xlim(0,6)
8  ax.set_ylim(-2,4)
9  ax.set_xlabel('x [m]',size=9)
10 ax.set_ylabel('y [m]',size=9)

```

**Fig. 6.4** Presenting the objects *arr*, *dot*, and *ln* **a** (left) with toy data, **b** (right) at the end of the animation

corner and increases to the right. We need only one subplot, so the instruction is `add_subplot(111)`. Resulting figures are shown in Fig. 6.4 with different plot objects.

`%matplotlib` is a magic function that renders the figure in a notebook instead of displaying a dump of the figure object.⁸

Creating plot objects

We first create some toy data. In Table 6.5, three two-dimensional print objects *arr*, *dot*, *ln* are created. In Table 6.5a, the lists provided for the *x* and *y* data are empty but the styles (line or dot) are already specified. Furthermore, a legend object *leg* is created with captions for the three plot objects. In Table 6.5b, the three plot objects are provided with some toy data. Running this cell, yields Fig. 6.4a.

⁸ <https://stackoverflow.com/questions/43027980/purpose-of-matplotlib-inline>.

Table 6.5 Creating plot objects **a** (top) with empty lists **b** (bottom) with toy data, result shown in Fig. 6.4a

```

11  #Creating plot objects
12  arr,=ax.plot([], [], 'k-', lw=0.5)
13  dot,=ax.plot([], [], 'ko', ms=3)
14  ln, =ax.plot([], [], 'k--', lw=1)
15  leg = ax.legend(['arr', 'dot', 'ln'])
16
17  leg.set_title('trajectory', prop={'size':6})
18  arr,=ax.plot([1,2,1,1,1,2],[0,0,0,-0.5,0,-0.5],
                'k-', lw=0.5)
19  dot,=ax.plot([3],[3], 'ko', ms=3)
20  ln, =ax.plot([2,3,4,5],[2,1,0,-1], 'k--', lw=1)
21  plt.savefig("PhEx 6-2 Trajectory initial", dpi=1200)

```

Making an animation

We get a “living figure” by applying the function `FuncAnimation` that creates an animation by repeatedly calling a function.⁹ Its variables are `FuncAnimation(fig, func, frames = None, interval = 200, init_func = None, ...)`. *Fig* is a *figure* object. *func* is a *callable*, or, more precise, a function that refreshes the plot objects for every frame. The first argument in *func* will be the next value in *frames*. *Interval* specifies the delay between successive frames in ms.

In Table 6.4, we have specified the figure as *fig* and in Table 6.7a, the refresh function as *animDotLine*. The delay time is set to 100 ms. The frames parameter is given as an integer that gives the index of the position and velocity arrays up to a certain time. The final result is shown in Fig. 6.4b (from Table 6.6).

The refresh function *animDotLine* is given in Table 6.7a. The variable in the function header is taken as index for the global arrays for position and velocity.

The plot objects are refreshed with the instruction `set_data`. For the projectile (*dot*) it is just the current position (one point) . The trajectory itself (*ln*) is represented by the curve traversed so far; the data are obtained by appending the current position to the lists x_{Lin} and y_{Lin} . The data x_{Arr} and y_{Arr} for the velocity arrows are calculated in another function *coArr*, reported in Table 6.7b.

We take the plot objects in this exercise as prototypes for other exercises:

dot single point,
arr new picture,
ln continued curve.

⁹ https://matplotlib.org/stable/api/_as_gen/matplotlib.animation.FuncAnimation.html

Table 6.6 Creating an animation object recurring to the figure object *fig* and the function *animDotLine*; plotting arrows for indices in a list (here only one element)

```

1  xLin, yLin = [], []
2  xArr, yArr = [], []
3  scal=0.2
4  dotAnim=animation.FuncAnimation(fig, animDotLine,
      frames=int(len(t)*0.6), interval=100, repeat=False)
5
6  for k in [2,11,20]:
7      xArr,yArr=coArr(k)
8      for i in np.arange(0,5,2):
9          ArrowP([xArr[i],yArr[i]],[xArr[i+1],yArr[i+1]])
10 plt.show()
11 plt.savefig("PhEx 6-2 Trajectory Anim partly",dpi=1200)

```

Table 6.7 **a** (left) Refresh function in the animation **b** (right) Function called in *animDotLine* for each frame

1	def animDotLine(i):	10	def coArr(i):#Coeff. arrow
2	xLin.append(x[i])	11	xi,yi=x[i],y[i]
3	yLin.append(y[i])	12	vxi,vyi=vx[i],vy[i]
4	dot.set_data(x[i],y[i])	13	xArr=[xi,xi+vxi*scal]
5	ln.set_data(xLin,yLin)	14	#2 elements
6	xArr,yArr=coArr(i)	15	yArr=[yi,yi+vyi*scal]
7	arr.set_data(xArr,yArr)	16	xArr+=(xi,xi+vxi*scal)
8	#x,y from Table 6.1	17	#4 elements
9	#xLin,yLin from Table 6.6	18	yArr+=(yi,yi)
		19	xArr+=(xi,xi)
			#6 elements
			yArr+=(yi,yi+vyi*scal)
			return xArr, yArr

Questions

The font size for the axis titles in Fig. 6.4b seems to be too small. What is the reason and how can you change it?¹⁰

How to provide the velocity vectors with arrow heads?¹¹

¹⁰ The default font size was set with `plt.rcParams.update({'font.size':7})`. With such an instruction you can change the whole font, not only its size.

¹¹ Use unit-line and perpendicular vectors, see Chap. 5.

Does the time development of the animation reflect the true development?¹²

In this exercise and most easily in all other exercises, all coordinates of the plot objects are calculated in the main program and stored in arrays that are then accessed for the animation. The animation is thus an addition to the main program. The advanced programmer can calculate the coordinates on the run, individually and temporarily for every frame.

6.3 Cycloid, Rolling Curve (R-T)

We consider the movement of a point fixed on a wheel rolling along a straight horizontal line. Viewed from the laboratory system, it is composed of a rotation about the wheel axis and a uniformly progressing translation of the axis parallel to the line. The speed of the point depends on its current altitude above the line.

6.3.1 Trace of a Writing Point Fixed at a Rolling Wheel

We are going to examine the movement of a point (“writing point”) on a wheel that rolls along a straight line in a plane as, e.g., shown in Fig. 6.5. The movement of the rolling wheel is composed of a rotation (R) about its axis and a translation (T) of the axis parallel to the plane (“the road”). Rotation period T_W of the wheel and velocity v_A of its axis are related as follows:

$$v_A = \frac{2\pi r_W}{T_W} \text{ with } r_W = \text{radius of the wheel} \quad (6.6)$$

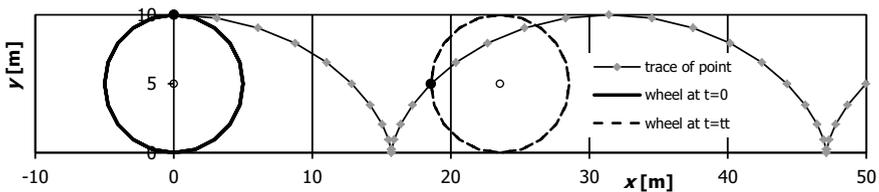


Fig. 6.5 Trace of a writing point on a wheel rolling along a straight line, always with the same time interval between two adjacent marks

¹² The time development of the animation reflects the true time development because the frames variable i is proportional to the time.

During a circulation period T_W , the wheel unrolls once on the road so that the axis covers a distance $2\pi r_W$, the length of the circumference.

For the coordinates (x_P, y_P) of the writing point and its speed v_T along its trajectory, the following equations hold:

$$\begin{aligned} x_P &= x_{ArA} + x_C = r_P \cdot \cos(\omega_W t) + x_A \\ y_P &= y_{ArA} = r_P \cdot \sin(\omega_W t) \end{aligned} \tag{6.7}$$

$$v_t = \frac{\Delta s}{\Delta t} = \frac{\sqrt{\Delta x^2 + \Delta y^2}}{\Delta t} \tag{6.8}$$

The index “ArA” designates rotation around the center (the axis), and x_A the translation of the axis.

► **Tasks** Represent the wheel and the writing point graphically at any time, to be set with a slider in the spreadsheet! Set-up a corresponding Python program with animation!

Show the trajectory $y = y(x)$ of the writing point in the same diagram!

Determine the writing point’s speed along its trajectory by numerical differentiation!

Speed of the writing point

The speed of a writing point on the wheel’s rim along its trajectory is shown in Fig. 6.6a, together with its average speed.

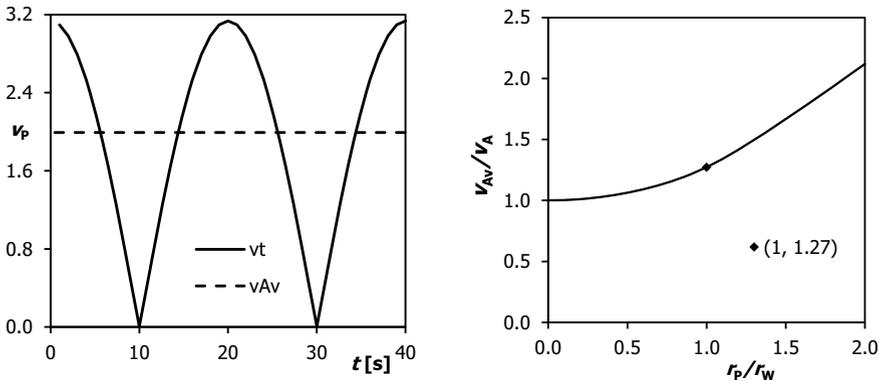


Fig. 6.6 **a** (left) Speed v_P of a point on the rim of the wheel in [m/s] as a function of time ($r_P = r_W = 5$ m). **b** (right) Mean trajectory velocity v_{Av} of the point relative to the velocity v_A of the axis, as a function of the distance r_P of the point P to the wheel axis relative to the radius r_W of the wheel

Questions

At which points of the wheel is the writing point's speed minimum and at which is it maximum?¹³

How do you calculate the speed $v_T(t)$ of the writing point along its trajectory numerically from $x_P(t)$ and $y_P(t)$ at time t , when the distance between the interpolation points is Δt ?¹⁴

Over which t value is the numerically calculated path velocity to be plotted?¹⁵

What is the orbital period of the wheel in Fig. 6.6a?¹⁶

What is the speed v_A of the axis in Fig. 6.6a?¹⁷

When the writing point touches the ground, its speed disappears, here, at $T = 10$ s and 30 s. Its speed is maximum ($v = 2v_{\text{Axis}} = \pi$ m/s) when it is at the rotating wheel's highest point.

The average speed of the writing point is indicated in Fig. 6.6a by the horizontal dashed line. To get the average speed numerically, it is essential to average over whole cycle times, e.g., over two cycle times, as in Fig. 6.6a.

The average speed depends on the distance of the writing point from the axis of the wheel. In Fig. 6.6b, the average speed of the point relative to the speed of the axis is plotted as a function of the distance of the writing point from the axis. When the writing point is on the axis, its average speed is equal to the speed of the axis. When it is on the rim, the speed ratio is $1.273 = 4/\pi$.

► **Task** Determine the point's mean path velocity as a fraction of the axis velocity for different distances of the point to the axis as in Fig. 6.6b!

Several trajectories with a procedure

► **Task** Vary the distance of the writing point r_P from the axis at time $t = 0$ systematically, e.g., as in Fig. 6.7. The distance can be greater than the radius, and also negative. In EXCEL, use a rep-log procedure; in PYTHON, define a function with r_P as the argument!

¹³ Minimum speed: point on the road ($v = 0$); maximum speed ($v = 2v_{\text{Axis}}$): at the highest point of the wheel.

¹⁴ $\vec{v}_T = (v_x; v_y) = ((x(t) - x(t - \Delta t))/\Delta t; (y(t) - y(t - \Delta t))/\Delta t)$ then $|v_T| = \sqrt{v_x^2 + v_y^2}$.

¹⁵ The speed is plotted over the center of the interval for which the velocity is numerically calculated.

¹⁶ $T = 20$ s, period of the speed profile, speed of the axis $= 2 \cdot \pi \cdot 5/20 = 1,57$ [m/s].

¹⁷ Speed of the axis $v_A = \frac{2\pi \cdot r_A}{T_w} = \frac{2\pi \cdot 5}{20} = \frac{\pi}{2}$ ($\frac{m}{s}$)

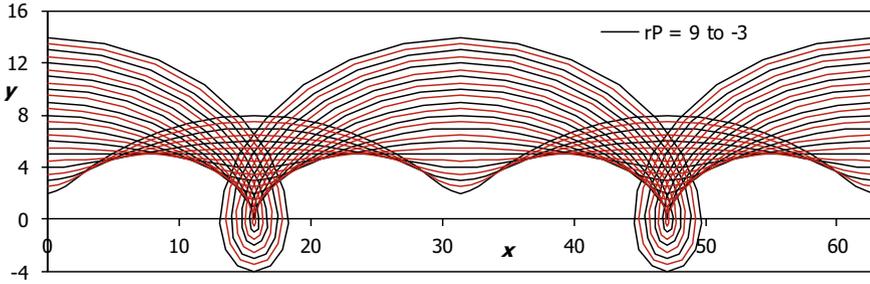


Fig. 6.7 Trajectories of writing points on a wheel with various distances r_P to the axis

Brachistochrone

Johann Bernoulli discovered, in 1696, that cycloids solve the brachistochrone problem. A brachistochrone curve is a trajectory on which a body in the homogeneous gravitational field of the earth glides fastest from a starting point to a lower endpoint.

To describe such trajectories with our data structure, we would have to set ω_W positive and $y_R = -r_R$, so that the wheel, hanging from a ceiling, unrolls to the right. The resulting curve is essentially part of one of the curves in Fig. 6.7 reflected at the x -axis. It indeed has the property of a brachistochrone, as we shall see in the chapter “Variational Calculus” of the follow-up textbook “Physics with Excel and Python. Applications”.

6.3.2 Data Structure and Nomenclature

r_W	radius of the wheel
r_P	distance of the writing point to the wheel axis
T_W, ω_W	cycle time, and angular velocity of the wheel
$v_A = (2\pi r_W) / T_W$	velocity of the axis
t	series of equidistant (dt) time instants
t_C	center of the time intervals of t
x_{ArA}, y_{ArA}	rotation of the point around the axis
x_A	horizontal position of the axis
x_P, y_P	trace of the writing point in the xy -plane
v_P	velocity of the writing point along its trajectory
tt	one specific time point.

	A	B	C	D	E	F
1	Prespecifications					
2	Radius of the wheel	rW	5.00 m			
3	Distance of the point from the axis	rP	5.00 m			
4	Cycle time of the wheel	TW	20.00 s			
5	Time interval	dt	1.00 s			
6	Calculated therefrom					
7	Angular frequency	wW	-0.31 1/s		$=-2*PI()/TW$	
8	Velocity of the axis	vA	1.57 m/s		$=2*PI()*rW/TW$	

Fig. 6.8 (S) Parameters for the rolling curve in Fig. 6.5

6.3.3 Excel

Parameters of the motion

In Fig. 6.8 (S), the movement parameters are specified in named cells, in particular, the radius r_W and the cycle time T_R of the wheel. From the given parameters, we derive angular frequency ω_W , height y_P , and velocity v_A of the axis. The distance of the writing point to the axis of the wheel r_P need not be equal to the radius r_R of the wheel. It may be bigger or smaller. The wheel will move on the plane to the right, rotating clockwise. The angular frequency ω_R , with which the polar angle is calculated, is therefore negative.

Questions

From which axis is the angle φ of the plane polar coordinates measured?¹⁸

What is the angle φ when the writing point is at the highest point of the wheel?¹⁹

What is the angle φ when the writing point is at the height of the axis?²⁰

What sign does ω_W have in a right-handed coordinate system when the wheel rotates clockwise?²¹ Compare with Fig. 6.8 (S)!

Trace of the writing point

In Fig. 6.9 (S) , the coordinates (x_P, y_P) of the writing point are calculated for 41 instants of time, with intermediate calculations for the rotation (x_{ArA}, y_{AA}) of the point about the axis of rotation (C:D) and for the translational motion (x_A) of the axis (column E).

¹⁸ The angle φ is measured from the positive x axis.

¹⁹ $\varphi = \pi/2 = 90^\circ, 90^\circ + 360^\circ, 90^\circ + 360^\circ + 360^\circ, \dots$

²⁰ $\varphi = 0 = 0^\circ.$

²¹ The angular frequency is negative when the point runs clockwise.

	B	C	D	E	F	G	H	I	J
9						vAv	1.99 m/s		=AVERAGE(vP)
10		wheel at t=0.5	trace of point				1.268		=vAv/vA
11		$=B15+dt$ $=rP * \cos(\omega t + \pi/2)$ $=rP * \sin(\omega t + \pi/2) + rW$ $=vA * t$ $=xArA + xA$ $=yArA$ $= \text{SQRT}((xP-F13)^2 + (yP-G13)^2) / dt$ $=(t+B13)/2$							
12		t	xArA	yArA	xA	xP	yP	vP	tC
13		0.5	0.78	9.94	0.79	1.57	9.94		
14		1.5	2.27	9.46	2.36	4.63	9.46	3.10	1
53		40.5	0.78	9.94	63.62	64.40	9.94	3.14	40

Fig. 6.9 (S) Coordinates of a writing point when unrolling the wheel; rotation of the point around the wheel axis (x_{ArA} ; y_{ArA}); translation of the axis of the wheel (x_A); addition of the two movements yields ($x(t)$; $y(t)$); speed along the path v_t

The corresponding equations are

$$x_{arA}(t) = r_P \cos(\omega t) + x_R; y_{arA}(t) = r_P \sin(\omega t) + r_W \tag{6.9}$$

Here, it is assumed that, at $t = 0$, the writing point is vertically above the axis. An example can be found in Fig. 6.5. You can extend the solution by allowing the selected point to assume any position at $t = 0$ by choosing the phase shift within the circular functions!

In column E, there is the displacement of the x -coordinate of the center of the wheel over time (translation in the x -direction). The coordinates of the rotating point in the laboratory system are in columns F and G, calculated from the superposition (component-wise addition of the Cartesian coordinates) of the rotation about the wheel axis (x_{arA} , y_{arA}) and the translation of the axis (x_M , 0):

$$x(t) = x_A + x_{arA} \quad y(t) = y_{arA} \tag{6.10}$$

The resulting trajectory is called a cycloid (rolling curve).

Question

In which cell of Fig. 6.9 (S) is the mean path velocity calculated?²²

► **Task** Determine the mean path speed of the point relative to the axis' speed for different distances of the point to the axis! A typical evaluation can be seen in Fig. 6.6a, b. It is best to use a VBA rep-log procedure that varies the distance and logs the average speed!

²² The mean speed v_{Av} is calculated in H9 of Fig. 6.9 (S).

Fig. 6.10 (S) Position of the point $(x(t_2); y(t_2))$ at time tt ; wheel at $t = 0$ (x_{rad}) and $tt = t_2$ (x_{t2}); tt is set with the slider. The formulas are the same as in Table 6.10

	K	L	M	N	O	P
10	point at t=tt		axis at tt		wheel at t=tt	
11	$=rP * \cos(wW * tt + PI() / 2) + xAtt$ $=rP * \sin(wW * tt + PI() / 2) + rW$ $=vA * tt$ $=xArA + xAtt$					
12	tt	xPtt	yPtt	xAtt	xWtt	
13	15	18.56	5.00	23.56	24.34	
14						25.83
15						27.10
53						24.34

Wheel at time tt

In Fig. 6.5, the wheel is shown at $t = 0$ and at a second time tt . In Fig. 6.10 (S), the coordinates of the wheel are determined for tt specified with a slider. The writing point has the coordinates (x_{Ptt}, y_{Ptt}) . The coordinates of the wheel are (x_{Wtt}, y_{Wtt}) , with y_W being the y -coordinates calculated for $t = 0$. If you put the pointer into the right part of the slider and keep it pressed down, the wheel rolls to the right.

6.3.4 Python

The Python program exhibits the same structure as the formula network in the spreadsheet. In Table 6.8, the parameters of the task are set.

In Table 6.9, the wheel’s coordinates at the start and the trace of the writing point are calculated. The velocity $v_P = ds/dt$ along the trace of the point is calculated in line 15. The length ds of the trace sections is obtained by slicing x_P and y_P .

The coordinates of the wheel and the writing point at a specific time $t = tt$ are obtained with the function *WheelAt* in Table 6.10, which has only tt as an argument and resorts otherwise to global variables.

The plot program for yielding a figure like Fig. 6.5, with the arrays calculated in Table 6.9, is given in Table 6.11. The coordinates for the wheel and the writing point at a specific time $t = tt$ are obtained by calling the function *WheelAt(...)* in Table 6.10.

Table 6.8 Setting the parameters for the rolling wheel, the same as in Fig. 6.8 (S)

```

1  #Prespecified:
2  rW=5.0           #Radius of the wheel
3  rP=5.0           #Distance of the point from the axis
4  TW=20            #Period of wheel rotation
5  dt=1.0           #Time increment
6  #Calculated therefrom:
7  wW=-2*np.pi/TW #Angular frequency
8  vA=2*np.pi*rW/TW #Velocity of the axis

```

Table 6.9 Calculation of the coordinates of the wheel at its start and of the trace of the writing point

```

8  #Coordinates of point vs. time:
9  t=np.arange(0,40+dt,dt)
   #Coordinates of the wheel at start:
10 xArA=rW*np.cos(wW*t+np.pi/2)
11 yArA=rW*np.sin(wW*t+np.pi/2)+rW
12 xA=vA*t           #From velocity of axis
13 xP=xArA/rW*rP+xA  #Writing point
14 yP=rP*np.sin(wW*t+np.pi/2)+rW
   #Veloc. along the trajectory of the point:
15 vP=np.sqrt((xP[1:]-xP[:-1])**2+(yP[1:]-yP[:-1])**2)/dt
16 tC=(t[1:]+t[:-1])/2 #Valid for vP

```

Table 6.10 Function for specifying the coordinates of the wheel and the writing point at $t = tt$

```

1  def WheelAt(tt):
   #Wheel and point at t = tt:
2     xAtt=vA*tt           #Position of axis
3     xPtt=rP*np.cos(wW*tt+np.pi/2)+xAtt
4     yPtt=rP*np.sin(wW*tt+np.pi/2)+rW
5     xWtt=xArA+xAtt      #x Coordinates of the rim
6     return xAtt,xPtt,yPtt,xWtt

```

Table 6.11 Plot program yielding a figure like Fig. 6.5

```

1  FigStd('x',-10,50,10,'y',0,10.0,2.5,xlength=12,ylength=4)
2  plt.plot(xArA,yArA,'k-') #Wheel at t = 0
3  plt.plot(xP[0],yP[0],'ko') #Point at t = 0
4  plt.plot(xP,yP,'k-')    #Trace of point
5
6  xAtt,xPtt,yPtt,xWtt=WheelAt(tt=15)
7  plt.plot([xP[0],xAtt],[rW,rW],'ko', fillstyle='none')
8                                     #Pos. of axis
9  plt.plot(xWtt,yArA,'k--') #Wheel
10 plt.plot(xPtt,yPtt,'ko')  #Selected point
11 plt.axis('scaled')

```

A meaningful animation could comprise the functions *dot* for the writing point, *ln* for the cycloid, *arr* for the wheel from Sect. 6.2.5.

6.4 Foucault's Pendulum (T-R)

We calculate the trace of a pendulum swinging in the laboratory system (T) on a rotating table (R).

6.4.1 A Lecture Experiment

In a lecture experiment about Foucault's pendulum, a thread pendulum swings over a rotating plate, writing a trace thereon. In the laboratory system, the pendulum swings in a plane, with its suspension point being located in the axis of rotation of the plate.

Figure 6.11a shows the trace of the pendulum for a period of oscillation of $T_p = 1.2$ s and a rotation time of the table of $T_r = 9$ s. The partial circle "Stylo" represents the trace of a pen, resting in the laboratory system, along the rotating plate to indicate the sense of rotation.

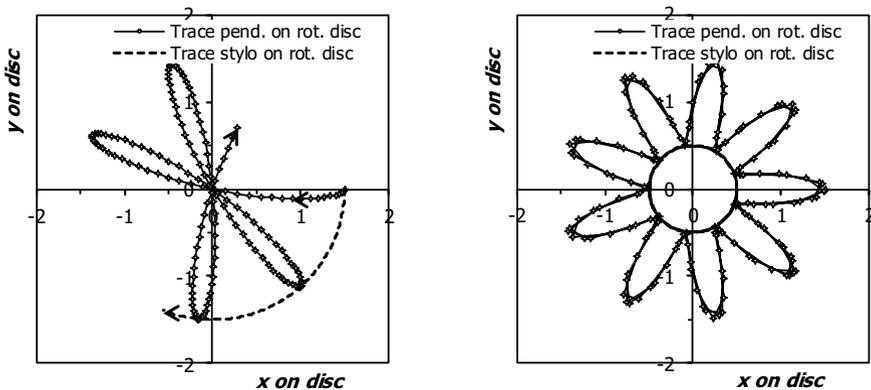


Fig. 6.11 **a** (left) Traces of a pendulum oscillating in the laboratory system ($T_p = 1.2$ s) and of a stylus at rest in the laboratory system on a rotating plate ($T_r = 9$ s), the suspension point of the pendulum being in the axis of rotation of the plate; the unit length is 1 cm, as explained in the main text. **b** (right) Closed track of a pendulum whose suspension point is not in the axis of rotation ($T_p = T_r/9$)

Questions

A thread pendulum is swinging with a period of 12.7 s. How long is the pendulum?²³

In what period of time does the earth rotate by 1°?²⁴

What are the amplitude of the pendulum and the horizontal displacement of the suspension point against the rotation axis in Fig. 6.11b?²⁵

Under what condition do closed tracks occur in the lecture experiment on Foucault's pendulum? What is the concrete condition in Fig. 6.11b?²⁶

For simplification, we assume that the trace of the pendulum on the plate at rest or, more generally, in the *laboratory system* is described by

$$x_p = A_p \cdot \cos(\omega_p t) \tag{6.11}$$

Then, we let the plate rotate around its vertical axis. The trace of the pendulum on the rotating table is composed of the oscillation in the *x*-direction (T, in the laboratory system) and an angular displacement on the table according to its rotation (R). The equations for the conversion of the coordinates (*x_L*, *y_L*) in the laboratory system into the coordinates (*x_T*, *y_T*) of the rotating table are

$$x_T = x_p \cdot \cos(\omega_T t) \quad \text{and} \quad y_T = x_p \cdot \sin(\omega_T t) \tag{6.12}$$

where ω_T is the angular frequency of the rotating table. These equations are a special case of the general form for a counter-clockwise rotation by ϕ applying a rotational matrix:

$$\begin{bmatrix} x_T \\ y_T \end{bmatrix} = \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} \cdot \begin{bmatrix} x_P \\ y_P \end{bmatrix} \tag{6.13}$$

Where have all the units gone?

► **Tim** In Fig. 6.11, we have not specified any physical units for the lengths.

► **Alac** That's no problem: Times in seconds, lengths in meters. That's standard.

²³ $T = 12,7 \text{ s}$, $\omega = 2\pi/T = \sqrt{g/l}$, \rightarrow length l of the pendulum = 40 m.

²⁴ The earth rotates by 360° in one day, \rightarrow in 4 min, by 1°; $\Delta t = 1^\circ/360^\circ * 24 * 60 * 60 \text{ s} = 240 \text{ s}$.

²⁵ The deflection of the pendulum is from 0.5 cm to 1.5 cm. The suspension point of the pendulum is shifted by a distance 1 cm against the center. The amplitude is 0.5 cm.

²⁶ The ratio of the period of the pendulum and the circulation time of the plate must be a natural number. In Fig. 6.11b, the ratio is 9 to 1. The pendulum makes nine oscillations during one turn of the plate.

- ▶ **Mag** Concerning calculation, everything is clear. But does that make sense physically? How long is the pendulum?
- ▶ **Tim** From the oscillation period, $T_P = 1.2$ s, as stated in the caption, it follows that $l = 36$ cm.
- ▶ **Mag** How does the maximum swing fit in with that?
- ▶ **Alac** I admit: A deflection of 1.8 m does not fit with the pendulum length. So, let's decide that the pendulum should be deflected by just 1.8 cm.
- ▶ **Mag** So, the unit of length in Fig. 6.11 is 1 cm. Indeed, this is experimentally difficult to record, but at least our calculation is consistent.
- ▶ **Tasks** Create a spreadsheet calculation/a Python program for the experiment described above and vary the pendulum's oscillation duration and the table's cycle time!

Check if the pendulum track is as expected (a) when the rotation period of the table is large compared to the period of oscillation and (b) when the oscillation period and rotation period are identical!

Change the calculation for the case in which the table's rotational axis is still in the plane of the swinging pendulum but no longer passes through the suspension point of the pendulum! An example can be seen in Fig. 6.11b.

6.4.2 Data Structure and Nomenclature

A_p	amplitude of the pendulum
T_p, ω_p	oscillation period of the pendulum and corresponding angular velocity
T_r, ω_r	rotation period of the table and corresponding angular velocity
x_{Sh}	shift of the suspension point with respect to the rotational axis
t	series of equidistant (dt) time instants
x_p	position of the pendulum at t in the lab system
x_T, y_T	trace of the pendulum on the table
x_{St}, y_{St}	trace of a stylus at rest in the lab system on the table, to check the direction of rotation of the table.

6.4.3 Excel

Setting the parameters

The parameters for the movement in Fig. 6.11 are specified in Fig. 6.12 (S). The quantity x_{Sh} (in C6) determines the displacement of the suspension point against the plate's axis of rotation.

	A	B	C	D	E
1	Prespecifications				
2	Amplitude of pendulum	Ap	1.50		0.50
3	Period of pendulum	Tp	1.20		1.00
4	Period of rotation	Tr	9.00		9.00
5	Time interval	dt	0.0173		0.09
6	Suspension point vs rot. axis	xSh	0.00		1.00
7	Calculated therefrom				
8	Angular frequency pendulum	wP	5.24	$=2*PI()/Tp$	
9	Angular frequency rotating disc	wR	-0.70	$=-2*PI()/Td$	

Fig. 6.12 (S) Specifications for the movement presented in Fig. 6.11; values in column C for partial picture **a**, those in column E for partial picture **b**

	B	C	D	E	F	G	H
12	$=B14+dt$	$=Ap * \cos(wP*t) + xSh$	$=xP * \cos(wR*t)$	$=xP * \sin(wR*t)$	$=Ap * \cos(wR*t)$	$=Ap * \sin(wR*t)$	
13	t	xP	xT	yT	xSt	ySt	
14	0.000	1.50	1.50	0.00	1.50	0.00	
15	0.017	1.49	1.49	-0.02	1.50	-0.02	
174	2.768	-0.52	0.18	0.49	-0.53	-1.40	

Fig. 6.13 (S) In column C, the pendulum motion $xP(t)$ is calculated in the laboratory system. In columns D and E, this movement is transformed (to xT, yT) into the coordinate system of the rotating table. In columns F and G, the coordinates on the rotating plate of a point (“Stylo”) fixed in the laboratory system are calculated

Question

How long is the pendulum in Fig. 6.12 (S) when the oscillation period T_P is given in seconds?²⁷

Trace of pendulum

The movement itself is calculated in Fig. 6.13 (S) for the pendulum swinging in the x -direction.

► **Task** Complete the diagram with two points representing the pendulum’s positions and the pen at a selectable time! In Fig. 6.11a, this was done for $t = 0.2249$ (arrow close to $x = 1$). A suggestion: Use a slider to select a row from 14 to 174 and copy the coordinates from that line to an area added to the diagram as a point. You can use the reference type INDIRECT for this purpose.

²⁷ $T = 1.2$ s, $\omega = 2\pi/T = \sqrt{g/l}$, length l of the pendulum = 0.36 m.

Table 6.12 Foucault's pendulum, specification of the parameters, the same as in Fig. 6.12 (S)

```

1  #Prespecified:
2  Ap=1.5           #Amplitude of the pendulum
3  Tp=1.2           #Period of the pendulum
4  Tr=9.0           #Period of the rot. disc
5  dt=0.0173       #Time interval
6  xSh=0.0          #Shift of suspension point
7  #Calculated therefrom:
8  wP=2*np.pi/Tp  #Circ. freq. of the pendulum
9  wR=-2*np.pi/Tr #Circ. freq. of the disc

```

Table 6.13 Setting up the arrays describing the motion of the pendulum in the laboratory system and its trace on the rotating table

```

1  t=np.arange(0,2.768+dt,dt)
2  #Traces:
3  xP=Ap*np.cos(wP*t)+xSh #Pendulum in lab
4  xT=xP*np.cos(wR*t)     #Trace of pendulum on table
5  yT=xP*np.sin(wR*t)
6  xSt=Ap*np.cos(wR*t)   #Trace of stylus on table
7  ySt=Ap*np.sin(wR*t)

```

6.4.4 Python

In Table 6.12, the parameters of the swinging pendulum are specified, with the same values as in Fig. 6.12 (S).

The arrays describing the motion of the pendulum in the laboratory system and its trace on the rotating table are set up in Table 6.13, together with the trace of a stylus fixed in the laboratory system

To get figures such as those in Fig. 6.11, we apply the program in Table 6.14. The parameters in Table 6.12 are for Fig. 6.11a. To get Fig. 6.11b, the parameters in column E of Fig. 6.12 (S) have to be inserted into Table 6.12.

Remember: Within brackets or parentheses, line breaks are allowed after punctuation marks, as is applied in Table 6.14, line 5. Explicit line breaks are possible after a backslash (\) as in line 2.

Questions

concerning Table 6.14:

How many positional arguments are in the header of *ArrowP*?²⁸

What is the first argument in the header of *ArrowP* in line 9?²⁹

²⁸ *ArrowP*(P_0, P_1, \dots) has two positional arguments, foot point P_0 and head point P_1 .

²⁹ $P_0 = [xT[-2], yT[-2]]$.

Table 6.14 Plotting the arrays obtained in Table 6.13 to get a picture like that in Fig. 6.11a, *ArrowP* from Table 6.3

```

1 FigStd('x',-2.0,2.0,0.5,'y',-2.0,2.0,0.5)
2 plt.plot(xT,yT,'k-x',ms=3,label='pend\
3         ulum on disc')
4 plt.plot(xSt,ySt,'k--',label='stylo on disc')
5 ArrowP([xSt[-2],ySt[-2]],[xSt[-1],
6         ySt[-1]],hw=0.1)
7 i=10
8 ArrowP([xT[i],yT[i]],[xT[i+1],yT[i+1]],hw=0.1)
9 ArrowP([xT[-2],yT[-2]],[xT[-1],yT[-1]],hw=0.1)
10 plt.legend()

```

To which arrows in Fig. 6.11a do the three calls of the function *ArrowP* correspond?³⁰

► **Task** Set up an animation in the laboratory system, with the pendulum swinging horizontally and the table rotating! The frames should be the equidistant time instants t to mimic the oscillation.

6.5 Anchor, Deflected Out of Its Rest Position (R-R)

The rotation of an anchor about its suspension point is described as a rotation (R) of the center of gravity about the suspension point and a rotation (R) of the anchor about the center of gravity (if the anchor is not suspended there). The moment of inertia is calculated using Steiner's theorem.

6.5.1 Deflected Anchor

Coordinates of the deflected anchor

We consider the rotation of an anchor about a point located in the origin of the coordinate system. In Fig. 6.14a, the anchor is held at the end S of the stem, in Fig. 6.14b, at its center of gravity C_g . The construction of the anchor is simplified with four mass points attached to the ends of a hanging T (see the inset in Fig. 6.14b).

³⁰ First: end of trace "stylo"; second: initial phase ($i = 10$) of trace "pend"; third: end of trace "pend".

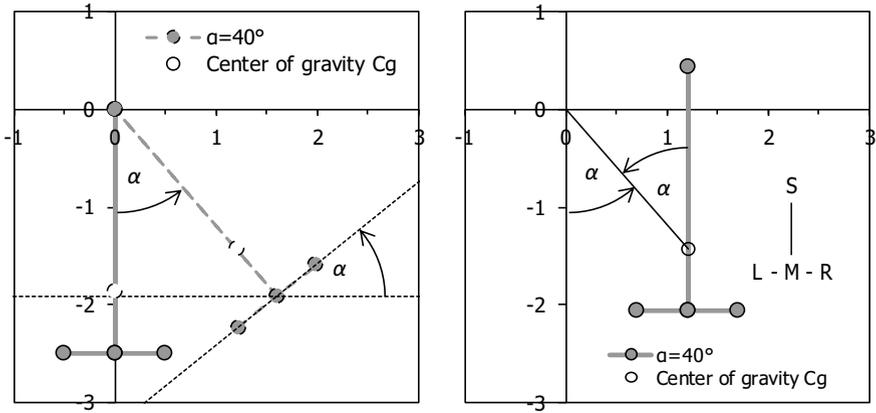


Fig. 6.14 An anchor is deflected by an angle $\alpha = 40^\circ$, held **a** (left) at the upper end of the stem, **b** (right) at its center of gravity. The four characteristic point masses are designated by the letters S, L-M-R. The center of gravity is marked by an open circle

In the following programs, the anchor's characteristic points are listed as [L, M, S, M, R] (Left, middle, top, middle, right; see Fig. 6.14b). We have doubled M because the anchor can then be drawn in one uninterrupted line. The distances L-M = M-R and S-M are specified, respectively, as r_A and l_A , so that the coordinates of the anchor at rest are defined as

$$\begin{aligned} x_A &= [-r_A, 0, 0, 0, r_A] \\ y_A &= [-l_A, -l_A, 0, -l_A, l_A] \end{aligned} \tag{6.14}$$

Rotation about the suspension point S

If the anchor is rotated about its top point S (Fig. 6.14a), the displacement of any point R of the anchor, with Cartesian coordinates $(-r_A, -l_A)$ at rest, may be considered the sum of two rotations:

$$\begin{aligned} \text{Rotation of point } R \text{ around } M &: (-r_A, 0) \rightarrow (-r_A \cdot \cos(\alpha), r_A \cdot \sin(\alpha)) \\ \text{Rotation of point } M \text{ around } S &: (0, -l_A) \rightarrow (l_A \cdot \sin(\alpha), -l_A \cdot \cos(\alpha)) \\ \text{With the sum :} & \quad (-r_A, -l_A) \rightarrow \begin{pmatrix} -r_A \cdot \cos(\alpha) + l_A \cdot \sin(\alpha), \\ r_A \cdot \sin(\alpha) - l_A \cdot \cos(\alpha) \end{pmatrix} \end{aligned}$$

The total rotation can be presented as a matrix multiplication:

$$= (-r_A, -l_A) \cdot [\text{RotMat}] \tag{6.15}$$

with the rotational matrix defined as

$$\text{RotMat} = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{6.16}$$

The rotational matrix can be applied to any point in the plane:

$$(x_R, y_R) = (x, y) \cdot [RotMat] \quad (6.17)$$

Here, we have chosen to present the coordinates as row vectors, because this is more convenient in spreadsheet calculations where coordinate vectors (a pair of two numbers) can more clearly be stored in successive rows.

► **Task** Calculate the coordinates of the anchor for the freely selectable parameters: length l_A of the stem (A-M), half the length r_A of the crossbar (M-R or M-L), and angle α !

Center of gravity as weighted sum

The center of gravity is defined as the sum over the coordinates of the characteristic points weighted with their mass:

$$\begin{aligned} x_G &= \frac{(\sum_i m_i \cdot x_i)}{\sum_i m_i} \\ y_G &= \frac{(\sum_i m_i \cdot y_i)}{\sum_i m_i} \end{aligned} \quad (6.18)$$

This is achieved in EXCEL with

$$y_G = \text{SUMPRODUCT}(y, m) / \text{SUM}(m)$$

and in Python with

$$y_G = \text{np.dot}(y, m) / \text{np.sum}(m)$$

The function `np.dot` returns the dot product of two arrays. For 1-D arrays, it is the inner product of the vectors, i.e., the sum of the products of the components, the equivalent of `SUMPRODUCT`. For 2-D arrays, it is equivalent to matrix multiplication.

The anchor is held at its center of gravity

In Fig. 6.14b, the anchor is held at its center of gravity by a rod (considered to be massless) that can rotate around S and is currently rotated by an angle α . The coordinates of the deflected anchor are the coordinates of the anchor at rest shifted by the coordinates of the center of gravity after rotation by α according to Eq. 6.17 applied to (x_G, y_G) . Now, only one rotation is effective.

Moment of inertia

The moment of inertia I is defined as

$$I = \sum_i r_i^2 \cdot m_i \quad (6.19)$$

where r_i is the distance to the axis of rotation, here, $r_i^2 = x_i^2 + y_i^2$.

It is calculated

in Python as: `np.sum((x**2+y**2)*m)`

in EXCEL as: `SUMPRODUCT(x; x; m) + SUMPRODUCT(y; y; m)`

We calculate the moment of inertia by applying the formulas:

- (a) to a rotation of the anchor about the point S (corresponding to Fig. 6.14a),
- (b) to a rotation of the anchor about its center of gravity,
- (c) to a rotation of the total mass, concentrated in the center of gravity, about S (corresponding to Fig. 6.14b).

The sum of (b) and (c) should be equal to (a), according to Steiner's law.

Animation

We set up an animation with the anchor swinging correctly with sinusoidal time dependence about its suspension point.

6.5.2 Data Structure and Nomenclature

r_A	distance L-M = M-R
l_A	distance M-S
x_A, y_A	coordinates of the anchor at rest (5 elements, combinations of r_A and l_A)
xy_A	= $[x_A, y_A]$ (2D range)
α	angle of rotation
x_R, y_R	coordinates of the deflected anchor, rotated around the suspension point
x_G, y_G	coordinates of the center of gravity for the anchor at rest
x_{Gr}, y_{Gr}	coordinates of the center of gravity rotated around S , at the origin of the coordinate system
<i>RotMat</i>	rotational matrix

Equivalence:

EXCEL: `[xR,yR] = MMULT(xyA,RotMat), xyA = [xA, yA]`

Python: `[xR,yR] = RotMat @ [xA,yA] or np.matmul (RotMat,[xA,yA])`

6.5.3 Excel

Specs

The parameters of the exercise are specified in columns A:E of Fig. 6.15 (S), with the rotation angle being adjusted with a slider. The coordinates of the anchor at rest are specified in columns G:H by inserting $\pm l_A$ or $\pm r_A$, or 0 when appropriate. The masses attached to these points are given in column I. The coordinates of point M appear twice in x and y , but the second time with zero mass so that the center of gravity and moment of inertia may be calculated in formulas in Fig. 6.16 (S), taking the whole arrays x_A, y_A , and m as input.

The anchor can be made to oscillate by running the time t with a slider and calculating the deflection angle α in C4 with $\alpha = A \cdot \cos(\omega \cdot t)$ with suitable A and ω .

Rotational matrix

The coordinates (x_R, y_R) of the deflected anchor are calculated by applying the rotational matrix *RotMat* in Fig. 6.16 (S) to the coordinates (x_A, y_A) of the anchor at rest.

Center of gravity

The center of gravity C_g and the moments of inertia are calculated in Fig. 6.17 (S). B5 = ["Deflection angle ="&C3&"°"]. The moment of inertia for a rotation about the suspension point S is calculated twice:

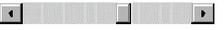
	A	B	C	D	E	F	G	H	I
1	Length of rod	lA	2.50 m				xA	yA	m
2	Half-length of cross bar	rA	0.50 m			L	-0.50	-2.50	1.0
3		220	40.00 °		=B3-180	M	0.00	-2.50	1.0
4		a	0.70 rad		=C3/180*PI()	S	0.00	0.00	1.0
5	Deflection angle $\alpha=40^\circ$					M	0.00	-2.50	0.0
6						R	0.50	-2.50	1.0

Fig. 6.15 (S) Anchor parameters in B1:C4; the deflection angle α is determined with the slider in A3 (linked cell = B3). The coordinates x, y of the line connecting the characteristic points are in columns G and H with the associated masses in column I, set to 1 except for the second reference to point M . B5 = ["Deflection angle ="&C3&"°"]

	K	L	M	N	O	P
1	RotMat			xR	yR	
2	0.77	0.64		1.22	-2.24	=MMULT(xyA;RotMat)
3	-0.64	0.77		1.61	-1.92	
4				0.00	0.00	
5	=COS(a)	=SIN(a)		1.61	-1.92	
6	=-SIN(a)	=COS(a)		1.99	-1.59	

Fig. 6.16 (S) Continuation of Fig. 6.15 (S). The rotational matrix *RotMat* is applied to the coordinates x_A and y_A , bound together in Fig. 6.15 (S) in one matrix $xy_A = [G2:H6]$ in Fig. 6.15 (S)

	AA	AB	AC	AD	AE
1	Center of gravity Cg			Moment of inertia for rotation about Cg	
2	xG	yG		5.19	=SUMPRODUCT(xA-xG;xA-xG;m)
3	0.00	-1.88	=SUMPRODUCT(yA;m)/SUM(m)		+SUMPRODUCT(yA-yG;yA-yG;m)
4	xGr	yGr		for rotation of Cg	
5	1.21	-1.44	=MMULT(xGyG;RotMat)	14.06	=(xG^2+yG^2)*SUM(m)
6					
7	Moment of inertia for rotation about S				
8		19.25	=SUMPRODUCT(x;x;m)	19.25	=AD2+AD5
9			+SUMPRODUCT(y;y;m)		

Fig. 6.17 (S) Continuation of Figs. 6.15 (S) and 6.16 (S). AA:AC: center of gravity, moment of inertia for rotation about the point $S = (0, 0)$; AD:AE: moment of inertia with Steiner’s theorem as the sum of two rotations calculated with the spreadsheet functions SUMPRODUCT and SUM

- in AB8, with the coordinates (x, y) of the anchor at rest,
- in AD2:AD8, as the sum of the moments of inertia for rotation about the center of gravity C_g and a rotation of the total mass in the center of gravity about $(0,0)$.

Both calculations should yield, according to Steiner’s theorem, the same result, and they do.

6.5.4 Python

Specs and rotational matrix

A Python solution corresponding to Figs. 6.15 (S) and 6.16 (S) is given in Table 6.15.

Table 6.15 Swinging anchor, specs as in Fig. 6.15 (S)

```

1  #Prespecified:
2  lA=2.5          #Length of stem
3  rA=0.5          #Half-length of crossbar
4  a=40           #Angle of deflection in °
5  a*=np.pi/180  #   in rad
6
7  #Coordinates of the anchor at rest:
8  #               L-M-S-M-R
9  xA=np.array([-rA,0,0,0,rA])
10 yA=np.array([-lA,-lA,0,-lA,-lA])
11
12 #Rotational matrix:
13 RotMat=np.array([[np.cos(a),-np.sin(a)],
14                  [np.sin(a), np.cos(a)]])
15 #Coordinates of the deflected anchor:
16 [xR,yR]=RotMat@[xA,yA]
```

Animation

In Table 6.16, a function *ancRot* is defined that is applied in the animation. It calculates the coordinates of the anchor and its center of gravity for an angle α . The function *anim* called within *FuncAnimation* converts the frame number i into the angle α through a sine function that mimics an oscillation.

Center of gravity

In Table 6.17, the coordinates of the center of gravity C_g are calculated, at rest and rotated. The dot product is used to calculate the nominator in the fraction for the center of gravity (Eq. 6.18).

A program for drawing the rotated anchor as shown in Fig. 6.14a is presented in Table 6.18.

In Table 6.19, the moments of inertia are calculated when the anchor is rotated:

- (a) about S, the upper end of the stem,
- (b) about C_g , the center of gravity; and
- (c) when the center of gravity is rotated about (0, 0).

Table 6.16 The anchor swings with sinusoidal time dependence about S

```

1  def ancRot(a):          #Angle a=alpha
2      #Rotational matrix:
3      RotMat=np.array([[np.cos(a), -np.sin(a)],
4                       [np.sin(a), np.cos(a)]] )
5      #Coordinates of the deflected anchor:
6      [xR, yR]=RotMat@[x, y]
7      [xGr, yGr]=RotMat@[xG, yG]
8      return xR, yR, yGr, yGr
9  anc,=plt.plot(xR,yR, 'k-o')
10
11 def anim(i):
12     a=np.pi/4*np.sin(i*0.1)
13     xR, yR, xGr, yGr = ancRot(a)
14     anc.set_data(xR, yR)
15
16 AnchorAnim=animation.FuncAnimation(figA, anim,
17     frames=range(180), interval=100, repeat=False)

```

Table 6.17 Coordinates of the center of gravity, at rest and rotated

```

1  #Center of gravity Cg
2  m=np.array([1,1,1,0,1]) #Masses of L-M-S-M-R
3  xG=np.dot(xA,m)/np.sum(m) #Coordinates of c of g
4  yG=np.dot(yA,m)/np.sum(m)
5  [xGr, yGr]=RotMat@[xG, yG] #Coordinates of rotated anchor

```

Table 6.18 Program for drawing a figure like Fig. 6.14a

```

1 FigStd('x',-1.0,3.0,1,'y',-3.0,1.0,1.0)
2 plt.plot(xA,yA,'k-o',label="α=0")      #At rest
3 plt.plot(xR,yR,'k:o',fillstyle='none',
4         lw=2,label="α="+str(round(a,2))) #Rotated by α
5 plt.plot(xG,yG,'ko',ms=4,
6         label='c of grav. ')          #Center of gravity
7 plt.plot(xGr,yGr,'ko',ms=4)
8 plt.legend()
9 plt.axis('scaled')

```

Table 6.19 Moment of inertia

1	<i>#Momentum of inertia, around S</i>		
2	IS=np.sum((x**2+y**2)*m)	IS	19.25
3			
4	<i>#Rotated coord. of center of gravity:</i>	xG	0.0
5	[xGr,yGr]=RotMat@ [xG,yG]	yG	-1.875
6			
7	<i>#Moment. of inertia, around c of grav</i>		
8	IarCg=np.dot((x-xG)**2,m) \	IarCg	5.19
9	+np.dot((y-yG)**2,m)		
10			
11	<i>#Rotation of center of gravity</i>		
12	ICg=(xG**2+yG**2)*np.sum(m)	ICg	14.06
13			
14	Itot=IarCg+ICg	Itot	19.25

We confirm again that (b) is smaller than (a) and that (b) + (c) equals (a).

Table 6.20 displays a program for plotting the anchor held at its center of gravity, as in Fig. 6.14b.

6.6 Wavefronts, Sound Barriers, and Mach Cone (T-T)

We draw the wave crests of sound waves emitted by a source moving at a certain speed and direction in the xy -plane and demonstrate the breaking of the sound barrier for supersonic speed. Polar coordinates are used for the calculation, and Cartesian coordinates for the scatter diagrams.

Table 6.20 Program for plotting the anchor gripped at its center of gravity (Fig. 6.14b)

```

1  x_Cg=xA+(xGr-xG)
2  y_Cg=yA+(yGr-yG)
3
4  FigStd('x',-1.0,3.0,1,'y',-3.0,1.0,1.0)
5  plt.plot(xA,yA,'k-o') #Anchor at rest
6  plt.plot(xG,yG,'kx',ms=8,label='c of g') #Cent. of gravity
7  plt.plot(x_Cg,y_Cg,'k:o',fillstyle="none") #Rotated anchor
8  plt.plot([0,xGr],[0,yGr],'k--',label="α="+str(a)) #Rotated rod
9
9  plt.plot(xGr,yGr,'kx',ms=8)
10 plt.plot()
11 plt.legend()
12 plt.axis('scaled')
    
```

6.6.1 Emitting Sound Waves

In Fig. 6.18, crests of sound waves in a plane emitted from a moving source are shown, in **b**, for supersonic speed. One wave crest is emitted in every period of the sound signal.

Sound barrier and Mach cone

► **Mag** Are the motions linear or rotational?

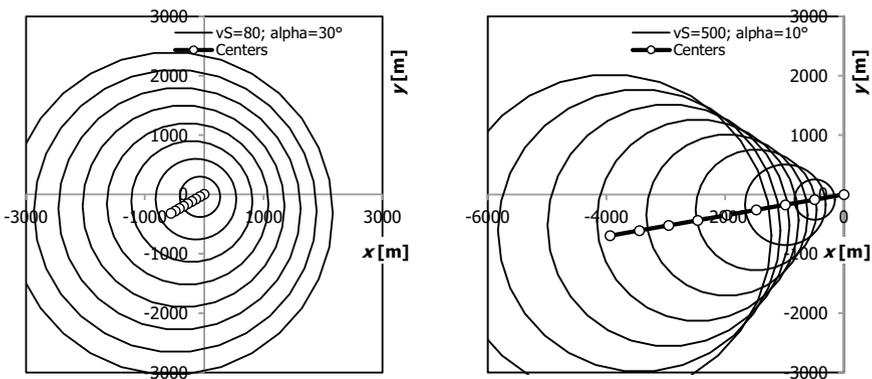


Fig. 6.18 **a** (left) Wavefronts of acoustic waves emitted by a moving sound source that has reached the position (0, 0) at $t = 0$; airspeed v_S (here, below the velocity of sound) and angle α of flight direction against the horizontal axis. **b** (right) As in **a**, but with $\alpha = 10^\circ$ and at supersonic speed

- ▶ **Alac** The motion of the airplane is along a straight line.
- ▶ **Mag** Ok, so it's translational. But what about the sound?
- ▶ **Tim** Sound is a longitudinal wave, so it's a linear motion.
- ▶ **Alac** Ok. But how do the circles arise?
- ▶ **Mag** As wave propagation is isotropic, the wave crests are circles.
- ▶ **Tim** And the circles are described with polar coordinates.
- ▶ **Alac** I understand, polar coordinates but no circular motion.
- ▶ **Tim** Ok, that's clear. The airplane, modeled as a point in the plane, moves linearly in two dimensions. Sound is a wave and propagates isotropically in air.
- ▶ **Mag** Another point: What happens if the aircraft flies at exactly the speed of sound?
- ▶ **Alac** Then a sound barrier builds up, and there is a loud bang.
- ▶ **Mag** Simulate this situation! The best way to do so in a spreadsheet is to install a slider and increase the speed v_S , of the sound source slowly, starting from zero up to the speed of sound!
- ▶ **Tim** What does “breaking the sound barrier ” mean?
- ▶ **Mag** When an airplane speeds up to the speed of sound, all sound waves arrive at a particular place at the same time and enforce each other to become the “sound wall”.

Coordinates of the circular wave crests

The flying object (the “source”) is traveling at speed v_S relative to the air and at an angle α to the horizontal axis (the x -axis), emitting sound waves in every period that propagate in the air at the speed of sound c . In our representation, Fig. 6.18, the source t is located at $t = 0$ at the site $(0, 0)$, and circular wave crests are calculated for every second, going back in time (negative time).

The center of the circle is given by the position of the flying object at the (negative) time of emission:

$$\begin{aligned}x_S(t) &= (v_S \cdot \cos(\alpha)) \cdot t \\y_S(t) &= (v_S \cdot \sin(\alpha)) \cdot t\end{aligned}\tag{6.20}$$

The expressions in parentheses decompose the distance $v_Q \cdot t$ traveled into horizontal (x) and vertical (y) components. $x_S(t)$ and $y_S(t)$ are smaller than 0 because $t < 0$ for our settings.

Starting from the trajectory of the object, waves are spreading with velocity c and have covered a distance r up to time t :

$$r = -c \cdot t \text{ for } t \leq 0 \quad (6.21)$$

so that (in our drawing plane) a circular wavefront arises. In three-dimensional reality, the wavefronts are, of course, spherical surfaces. For our two-dimensional representation, the quantities r become the radii of the circles in Fig. 6.18.

6.6.2 Data Structure and Nomenclature

c	speed of sound
v_S	speed of sound source
α	angular deviation of the linear track of the source from the x -axis
t	array of instants of time, negative; the current time is 0
r	array of radii of wave crests emitted at times t
x_S, y_S	arrays of coordinates of the source at times t ; the position at $t = 0$ is (0, 0)
phi	list of the polar angles for drawing the wave crests
x, y	2D matrices containing the coordinates of a set of wave crests, shape $size(phi) \cdot size(t)$.

6.6.3 Spreadsheet Solution

Calculating circles

In the spreadsheet of Fig. 6.19 (S), we produce eight circles to represent the crests of waves that have been sent out at instants $t = -1, -2, \dots, -8$ s. Two examples are shown in Fig. 6.18.

The definition of the coordinates of a circle is best done in polar coordinates. In Fig. 6.19 (S), the angle is defined in [A13:A43] from $\varphi = 0$ to 2π in 31 steps of $d\varphi = 0.209 = 2\pi/30$, set in A10. The three parameters of the task (sound speed c , speed v_S of source, flight angle α) are defined in B1:B4 and get named in A1, A2, A4.

The worksheet in Fig. 6.19 (S) has the typical Γ structure indicated by the bold Γ -shaped line. The calculation range below Γ spans B13:R43. The column-specific parameter set, coded as row vectors r, x_S, y_S , for the eight functions is in B7:I9 above Γ , controlled by the time t (see the formulas reported in J7:J9). The independent variable φ , the polar angle of the circles, is in A13:A43, to the left of Γ .

	A	B	C	D	E	F	G	H	I	J	K	L	R	S	
1	c_	340	m/s	Sound speed											
2	vS	500	m/s	Speed of source											
3		10	°	Direction angle											
4	alpha	0.175	rad	=B3/180*PI() vS=500; alpha=10°											
5															
6	t	-1	-2	-3	-4	-5	-6	-7	-8						
7	r_	340	680	1020	1360	1700	2040	2380	2720	=-c_*t					
8	xS	-492.4	-985	-1477	-1970	-2462	-2954	-3447	-3939	=vS*COS(alpha)*t					
9	yS	-86.82	-174	-260.5	-347.3	-434.1	-520.9	-607.8	-694.6	=vS*SIN(alpha)*t					
10	0.209	=2*PI()/30													
11	=A13+\$A\$10 =r_*COS(phi)+xS														
12	phi								={=r_*SIN(phi)+yS}						
13	0.000	-152	-305	-457	-610	-762	-914	-1067	-1219				-87	-174	-695
14	0.209	-160	-320	-480	-639	-799	-959	-1119	-1279				-16	-32	-129
43	6.283	-152	-305	-457	-610	-762	-914	-1067	-1219				-87	-174	-695

Fig. 6.19 (S) Coordinates (x, y) of circles with their centers shifted in the xy-plane (below Γ); Γ starts at B13 and is extended to the right and downwards; the x-coordinates in B13:I43 are generated with the formula reproduced in B11. The corresponding y values are in columns K:R. The angular coordinates φ for all curves are in A13:A43 (independent variable left of Γ) . The time in row 6 controls the radius (row 7) and the coordinates of the center (in rows 8 and 9)

This table can be enlarged row by row beyond row 43, for example, to allow for a finer angular scale, because it is downwards open. It cannot be broadened column by column, because, starting with column K, another calculation range follows, in which the y-coordinates of the circles are calculated.

- ▶ Remember: If you have written the formula in a cell correctly with relative and absolute references or with variable names, you can drag it into a larger cell range without changing it.

Questions

- concerning Fig. 6.19 (S):
- What is the meaning of the formula in cell B4, reported in D4?³¹
- What is the formula in cell D7?³²
- What is the formula in cell D9?³³
- Wouldn't it be nicer to have the wave crests cover the whole cone in Fig. 6.18b? How would you do that?³⁴

³¹ Transformation of the angle from degrees to radians: 360° = 2π, 180° = π.

³² [D7] = [= -c_*t], see J7!

³³ [D9] = [=vS*sin(alpha)*t], see J9!

³⁴ Specify the number of time instants to go from -1 to -12 in 12 steps.

Below Γ , eight circles around the origin with the radii r from line 7 are calculated from the column vector polar angle ϕ and the row vectors r , x_S and y_S .

The x values in B13:I43 are calculated with the simple formula $[=r_*\cos(\text{phi}) + x_S]$; the current values are taken from the same row or the same column as the current cell.

Matrix formula

The formulas in range B13:I43 of Fig. 6.19 (S) refer, for each cell, to the same column (when named row vectors are addressed) and the same row (when the named column vectors are addressed). The same formula network can be generated in any range with a matrix formula. An example is given in K13:I43 ($=\{[=r_*\cos(\text{phi}) + y_S]\}$). To do so, activate the range, enter the formula, and complete with $\Psi Ctrl + Shift + Enter!$

► **Task sliders** Install two sliders to adjust the source's direction and speed and observe how the diagram reacts to changes in both parameters and a change in sound velocity.

Discussion EXCEL

► **Tim** The wave crests in my diagram do not change when I change the speed of the aircraft.

► **Mag** You have copied the *numbers* from Fig. 6.19 (S) into range B7:I9. However, these cells have to contain *formulas*, not just numbers. Then, the values in these cells change as the parameters of the problem are changed. The formulas for column I can be found in column J. If you drag cells I7:I9 (with the formulas in J7:J9) to the left all the way to column B, the formula network for the parameters is complete.

► **Tim** What exactly is our task?

► **Mag** The polar angle is in column A, independent of the time. Your task is to enter a formula into cell B13 that creates the x coordinates of all eight circles at time t by dragging to the right and down to cell I43. To get the y coordinates, you have to apply a matrix formula, such as in K11 in Fig. 6.19.

At time $t = 0$, the aircraft shall be at the origin $(0, 0)$ of the coordinate system. The time in row 6 of Fig. 6.19 (S) is counted backward. So, the coordinates $(x_S; y_S)$ in rows 8 and 9 indicate where the flying object was 1, 2, etc., seconds ago. They are calculated, as discussed above, from the speed v_S of the source, the angle α , and the time t . The respective location of the flying object at this past instant is also the center of the circles representing the crests of the emitted sound waves.

Table 6.21 Coordinates of wave crests

1	<code>c=340</code>	<code>#[m/s] speed of sound</code>
2	<code>vS=500</code>	<code>#[m/s] speed of source</code>
3	<code>alpha=10</code>	<code>#[Degree]</code>
4	<code>alpha*=np.pi/180</code>	<code>#[rad]</code>
5		
6	<code>t=np.linspace(-1,-8,8,endpoint=True)</code>	
7	<code>r=-c*t</code>	<code>#Radius</code>
8	<code>xS=vS*np.cos(alpha)*t</code>	<code>#Center</code>
9	<code>yS=vS*np.sin(alpha)*t</code>	
10		
11	<code>ph=np.array([np.linspace(0,2*np.pi,31)])</code>	
12	<code>phi=ph.transpose(1,0)</code>	
13		
14	<code>x=r*np.cos(phi)+xS</code>	<code>#Coord. of circle</code>
15	<code>y=r*np.sin(phi)+yS</code>	

6.6.4 Python

The Python program for calculating the coordinates of the wave crests is shown in Table 6.21. After specifying the parameters speed of sound c , speed of source v_S , and deviation of the track of the source from the horizontal by an angle α , we specify the instants t of time at which the source is supposed to emit a signal. From t , we get the radii r of the circular sound crests and the Cartesian coordinates of their centers x_S and y_S , i.e., the current positions of the sender, all as arrays broadcast from t .

In ph , we define, as an array, the 31 angles with which circles are to be drawn as regular polygons. This array is transposed into a column vector phi . With these constructs and the two instructions

```
x = r*np.cos(phi) + xS
y = r*np.sin(phi) + yS
```

we reproduce the spreadsheet data structure of Fig. 6.19 (S) with the four row vectors t , r , x_S , y_S , and one column vector phi . The values of the different arrays are reported in Table 6.22. They coincide with those of the spreadsheet calculation in Fig. 6.19 (S).

Tangent to a circle

With Fig. 6.20, we complete Fig. 6.18b with tangents to the circles that represent the Mach cone. The construction of the tangent to a circle is illustrated in Fig. 6.20a and realized in Table 6.23.

► **Task** Verify the instructions in Table 6.23 with the help of Fig. 6.20a!

Table 6.22 Structure of the arrays in Table 6.21

alpha	0.175
t	[-1 -2 -3 -4 -5 -6 -7 -8]
r	[340 680 1020 ... 2040 2380 2720]
xS	[-492 -985 -1477 ... -2954 -3447 -3939]
yS	[-87 -174 -260 ... -521 -608 -695]
x[0]	[-152 -305 -457 ... -914 -1067 -1219]

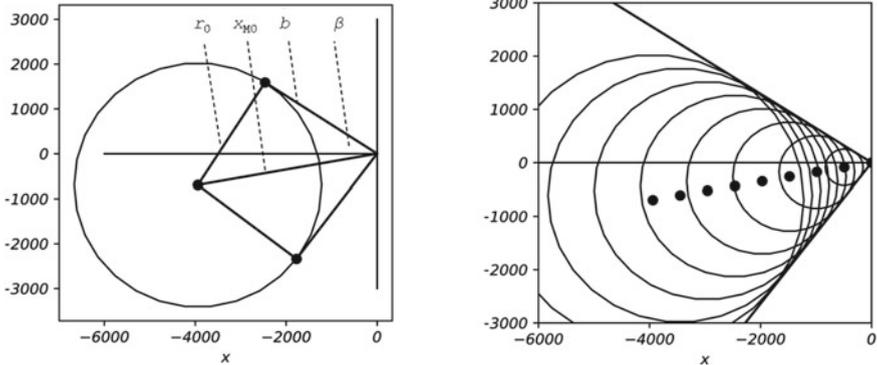


Fig. 6.20 **a** (left) Construction of a tangent to a circle. **b** (right) Mach cone with the sound barrier drawn as a tangent to the circles

Table 6.23 Tangent to a circle, drawn as straight lines from (0, 0) to (x_T, y_T) and from (0, 0) to (x_{Tu}, y_{Tu})

1	#Tangent, angle beta	6	$x_T = -np.\cos(-\alpha + \beta) * b$
2	$x_{M0} = vS * t[-1]$	7	$y_T = np.\sin(-\alpha + \beta) * b$
3	$r0 = r[-1]$	8	$x_{Tu} = -np.\cos(-\alpha - \beta) * b$
4	$b = np.\sqrt{x_{M0}^2 - r0^2}$	9	$y_{Tu} = np.\sin(-\alpha - \beta) * b$
5	$\beta = np.\arcsin(r0 / -x_{M0})$		

6.7 Questions and Tasks

1. What are the polar coordinates for the Cartesian coordinates (0, 5) and (1, 1)?
2. What are the Cartesian coordinates for the polar coordinates $r = 2$, and $\phi = 45^\circ$ or $\phi = 135^\circ$?
3. The spreadsheet formula = COS(90) returns -0.44807362. Is EXCEL thus wrong? However, np.cos(90) similarly returns -0.4480736161291701. Why?
4. Given the vector (3, 4), you are to attach this vector and its x- and y-components as arrows to point (1, 1). What does the data series in a spreadsheet look like? What are the three lists in Python for the coordinates?

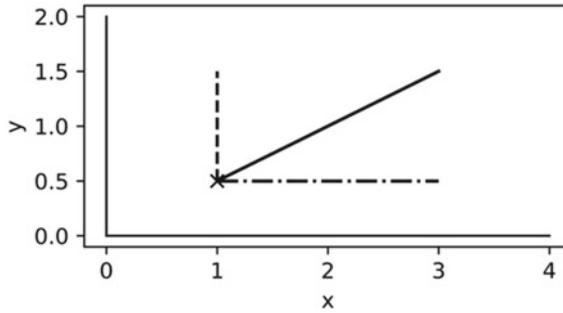


Fig. 6.21 Vector (2, 1) attached to (1, 0.5)

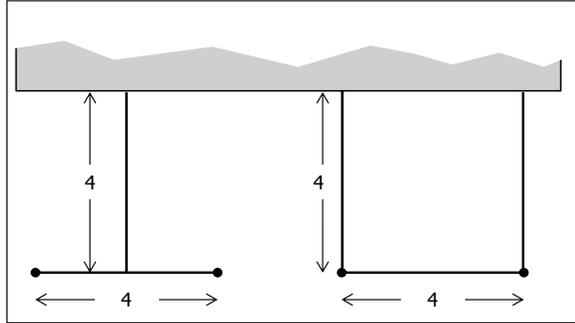
Table 6.24 Program snippet for attaching arrows at a point in the plane

```

1 FigStd('x',0,4,1,'y',0,2,0.5)
2 plt.plot(xO[0],xO[1],'kx')
3 plt.plot([xO[0],Arrow[0]],[xO[1],Arrow[1]],'k-')
4 plt.plot( ... )
5 plt.plot([...,'k-.' )
6 plt.axis('...')
```

5. In Fig. 6.21, the vector $v_A = (2, 1)$ and its components are attached to the point $x_O = (1, 0.5)$. Specify the variables x_O , v_A , and $Arrow$ and complete the program snippet in Table 6.24!
6. You are to draw the path (x, y) of a point in the plane, which moves with speed v in the xy -plane at an angle $\alpha = 30^\circ$ to the x axis. What are the formulas in the parameter representation $(x, y) = f(v, \alpha)$?
7. Describe the rotation of a point on a circle around the origin with rotation time T , both in polar and in Cartesian coordinates!
8. Which formulas apply to the Cartesian coordinates of a circle with diameter d , moving with velocity v along the x -axis?
9. A point moves with constant velocity v along the y axis of a laboratory system. What are its polar and Cartesian coordinates, in the laboratory system, and in a system moving relative to the laboratory system, with constant angular velocity ω_D , around an axis through the origin of the laboratory system?
10. Calculate the moments of inertia of the two dumbbells in Fig. 6.22! The two points represent masses of equal size. The connections between the masses and to the suspension points are supposed to be massless.

Fig. 6.22 A dumbbell, (left) suspended on the (massless) middle stem fixed to it, (right) suspended on two threads at its ends



- 11. Explain Steiner's theorem on the basis of Fig. 6.22a, b, together with Fig. 6.14!
- 12. How do you calculate the center of gravity of an arrangement of point masses m_i at (x_i, y_i) ?

Integration of Newton's Equation of Motion

7

The Newtonian equation for one-dimensional motions of point masses is numerically integrated by estimating the mean acceleration $\langle a \rangle$ in the time intervals between the supporting points of the time trajectory. Four methods are used: *Euler*, *half step*, “*Progress with look-ahead*” and *Runge-Kutta of the 4th order*. Three examples from an adventurous life are simulated realistically: Stratosphere jumping, bungee jumping, and car in a racing start. Different friction models (dry, viscous, Newtonian) are applied.

7.1 Introduction: Approximated Mean Value Instead of Exact Integration

Solutions of Exercises 7.2 (Python), 7.3 (Excel), 7.4 (Python), and 7.6 (Excel) can be found at the internet address: go.sn.pub/THdoLU.

7.1.1 Newton's Equation of Motion

We investigate the motion of a point mass on a straight line. According to Newton's laws, the acceleration at a certain point in time is determined by the forces acting on the body at that point in time. These forces F are generally dependent on the location x , e.g., due to a spring, or on the velocity \dot{x} , e.g., due to friction, or on both, so that, generally, we have $F = F(x, \dot{x})$.

Newton's equation of motion for one dimension is

$$\ddot{x}(t) = \frac{F(x, \dot{x})}{m} \quad (7.1)$$

where $\ddot{x} = a$ is the acceleration, $\dot{x} = v$ the velocity, x the location, and m the mass of the body. If all forces within a period of time are known, as well as the location and velocity of the point mass at the beginning of the period, the differential equation of motion, Eq. 7.1, can be solved. Then, the values of x and \dot{x} at the end of the time period can be calculated.

The forces in some of the exercises in this chapter depend only on velocity or only on location, making the calculations simpler and easier to follow.

We learn how Newton's equation of motion can be solved numerically in a sufficiently precise way over a period of time. To introduce the different methods, we work on tasks that can be analytically solved so that we can check the results of our calculations: Vibrations of a mass-spring system without friction (Exercise 7.2) and falling through air with friction (v^2 -proportional, Exercise 7.3).

Mean value instead of integral

The numerical solution we are striving for is based on difference equations in which time progresses by finite amounts Δt . If the location and the velocity of the body at the beginning t_0 of a time interval are known, the values at the end $t_0 + \Delta t$ of the interval follow as

$$v(t_0 + \Delta t) = v(t_0) + \int_{t_0}^{t_0 + \Delta t} a(t) dt = v(t_0) + \langle a \rangle \cdot \Delta t \quad (7.2)$$

$$x(t_0 + \Delta t) = x(t_0) + \int_{t_0}^{t_0 + \Delta t} v(t) dt = x(t_0) + \langle v \rangle \cdot \Delta t \quad (7.3)$$

Equations 7.2 and 7.3 are exact. The expressions with the mean values $\langle a \rangle$ and $\langle v \rangle$ correspond exactly to the integral, because the mean value is simply defined like that.

Our task is to estimate the *mean acceleration* $\langle a \rangle$ and *the mean velocity* $\langle v \rangle$ in the time interval under consideration. The numerical estimation is an approximation for which we present four different methods in the next section.

Ψ *Approximated mean instead of exact integral.*

Friction

To consider friction, we use the general formula:

$$a_{Frict} = -\text{sgn}(v) \cdot a_f \cdot |v|^n$$

The acceleration due to friction is proportional to a power n of the speed (the absolute value of the velocity). The first term $-\text{sgn}(v)$ guarantees that friction always tries to decrease the speed. The exponent n depends on the type of friction:

- $n = 0$ for internal friction, applied for the losses in a bungee rope (Exercise 7.6)
- $n = 1$ for viscous friction, applied for the damping of a harmonic oscillator (Exercise 7.2)

- $n = 2$ for Newtonian friction, active when dragging a body rapidly through a viscous medium, in our exercises, air (Exercises 7.3, 7.4, and 7.5).

Observe the influence of the different types of friction on the time trajectory!

7.1.2 Four Methods for Estimating the Average Acceleration in a Time Segment

In the simple *Euler method*, the mean acceleration in the time segment $[t_n, t_{n+1}]$ is approximated by the acceleration at the beginning of the time segment:

$$v(t_{n+1}) = v(t_n) + a(t_n) \cdot \Delta t \quad (7.4)$$

$$x(t_{n+1}) = x(t_n) + v(t_n) \cdot \Delta t \quad (7.5)$$

This procedure can be improved if the values from Eq. 7.4 and Eq. 7.5 are taken only as a “preview” for the velocity $v_p(t_n + \Delta t)$ and the location $x_p(t_n + \Delta t)$, from which the acceleration $a_p(t_n + \Delta t)$ at the end of the considered time period is calculated. For this “*Progress with look-ahead*”, the quantities at the beginning of the next interval are estimated as:

$$v(t_{n+1}) = v(t_n) + \frac{a(t_n) + a_p(t_n + \Delta t)}{2} \cdot \Delta t \quad (7.6)$$

$$x(t_{n+1}) = x(t_n) + \frac{v(t_n) + v_p(t_n + \Delta t)}{2} \cdot \Delta t \quad (7.7)$$

The *half-step procedure* has a similar structure; the values at the beginning of an interval are used to estimate the values in the middle of the interval, which then represent the entire interval.

A further improvement can be achieved with the fourth-order *Runge-Kutta method*, in which three projections are made in an interval, the first two into the middle and the third to the end of the interval. Then, a weighted average of four values is taken as representative of the whole interval.

All methods are trained

We shall apply all four methods in Exercise 7.3 (Falling from a great height; the force depends only on the speed) and compare the results with each other. There are analytical solutions for this task, so that we can check the precision of our numerical methods.

All methods provide sufficiently accurate solutions if the distance between the supporting points, which mark the time segments’ boundaries, is made sufficiently small. The bigger the effort with which the mean acceleration in a time segment is

calculated, the longer the time segments can be. Consequently, it must be checked, for each method, whether the selected length of the time segment is short enough.

As standard procedure in EXCEL, we shall use *Progress with look-ahead*, because it proves to be sufficiently efficient and can be implemented clearly in a spreadsheet. However, experienced readers may also use the *half-step procedure*, which provides the same accuracy and is also explained in detail in Exercise 7.3.

In Python, we shall use a function that implements the 4th-order Runge–Kutta method.

Adventurous life

In the course of this chapter, we will deal with three cases from an adventurous life: stratospheric jumping, bungee jumping, and a car in a racing start. In the follow-up volume *Physics with Excel and Python, Using the Same Data Structure. Applications*, the procedures from this chapter come into full fruition, with the treatment of motions in the plane, all kinds of oscillation, field lines, and wave functions of the Schrödinger equation.

7.1.3 Tactical Approaches in Python and Excel

In Python, the steps from t to $t + dt$ are done in a progress loop that calls a progress function, invariably called *progr* in all exercises. It does not have a proper code, but is assigned to one of the existing functions for *Euler*, *look-ahead*, or *Runge-Kutta*. Within the progress functions, the acceleration is invariably called *acc*, again, without proper code, but with an assignment to a function that is specific for the physical problem under consideration (*accSpring*, *accFall*, *accJump*, *accPwr*, *accBungee*). This approach is typical of Software solutions in which functions are embedded within a larger body.

The EXCEL approach is less general and must be adapted individually to each problem. The preview calculations for the current time interval are done in a row with specific formulas for acceleration. The values at the beginning of a new time interval in a new row have to be calculated from values in the preceding row.

Animations

The motions calculated in this chapter can easily be animated with the methods presented in Sect. 6.2.5.

7.2 Harmonic Oscillation with “Progress with Look-Ahead” and “Runge–Kutta”

We integrate Newton's equation of motion for a mass-spring system by calculating the acceleration in a time interval, (a) as an average of two values,

one at the beginning of the interval and the other calculated for the end of the interval with the values at the beginning, and (b) as an average over four values with the Runge-Kutta method.

7.2.1 Equation of Motion

The equation of motion for a mass fixed to a linear-elastic spring is as follows:

$$a = \ddot{x} = \frac{D}{m} - d \cdot v = -f \cdot x - d \cdot v \quad (7.8)$$

Here, x is the deflection out of the rest position, and D and m are the spring constant and the mass. Damping is set proportional to the velocity v and opposite to it: $-d \cdot v$.

Questions

What is the physical unit of the “spring constant” f in Eq. 7.8 in SI units?¹

What is the physical unit of the damping constant d in Eq. 7.8 in SI units?²

What are the initial conditions $x(0)$ and $v(0)$ in Fig. 7.1a?³

The solution for vanishing friction ($d = 0$) is a stationary oscillation with constant amplitude. We use this knowledge to check whether our numerical solutions are good enough. An example is shown in Fig. 7.1 for an oscillation with $f = 0.10$ calculated with *LookAhead*.

The zoom in Fig. 7.1b shows that the amplitude for $n = 100$ has increased over the initial value, indicating that 100 points for a time span of 80 are not good enough, but that $n = 400$ does the job. The position of the fourth maximum is at $t = 79.5$, corresponding reasonably well to the theoretical value of $4\pi/\sqrt{0.1} = 79.48$. We may therefore rely on *LookAhead* with this segmentation of time and can play around with the parameters.

In Fig. 7.2, we have doubled the spring constant and introduced damping, causing a strong decay of the amplitude. In the zoom, we see furthermore that the period of the damped oscillation is bigger than for the undamped oscillation.

► **Task** Vary the constant $f = D/m$ and observe whether the period duration behaves as predicted by the formula $T = (2\pi)/\sqrt{f}$ (for vanishing friction)!

¹ $[f] = [a/x] = (m/s^2)/m = 1/s^2$.

² $[d] = [a/v] = (m/s^2)/(m/s) = 1/s$.

³ $x(0) = 1$ (maximum deflection), $v(0) = 0$ (slope of $x(t)$).

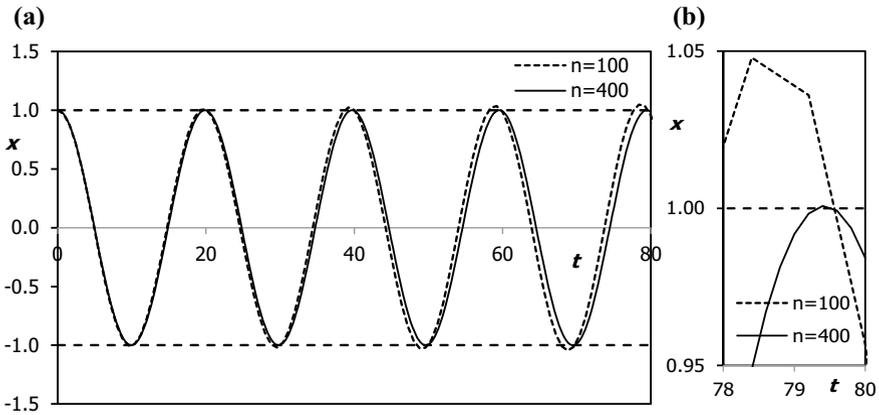


Fig. 7.1 **a** (left) Oscillation calculated with *LookAhead*. **b** (right) Zoom of **a**

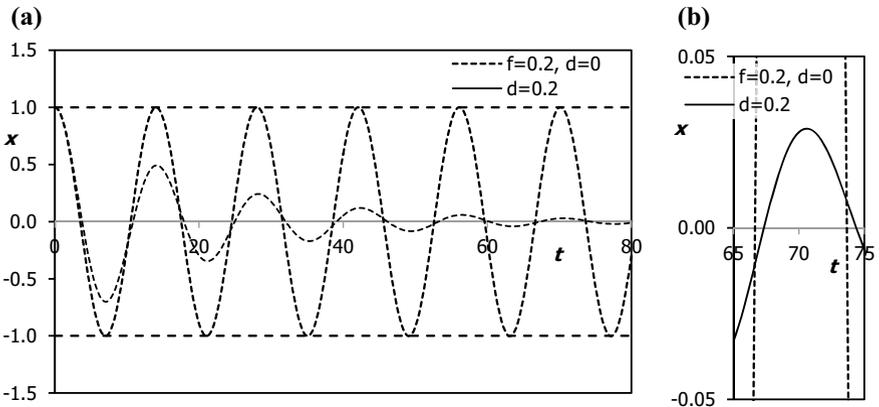


Fig. 7.2 Oscillation of a harmonic oscillator. **a** (left) Undamped and damped. **b** (right) Zoom of damped oscillation

In the following sections, we present *LookAhead* in a spreadsheet and *LookAhead*, *Euler*, and *Runge-Kutta* in Python. Varying the number n of points as in Fig. 7.1, reliable results are obtained with:

- $n = 20,000$ for *Euler*,
- $n = 400$ for *Progress with Look-ahead*,
- $n = 60$ for *Runge-Kutta*.

There is a reduction of about 50 going from *Euler* to *LookAhead* and about 8 going further to *Runge-Kutta*. Therefore, we choose *LookAhead* as standard in spreadsheet calculations with only four additional columns (see Fig. 7.3), thus keeping the programming effort to be repeated in every new spreadsheet small. We choose *Runge-Kutta* as standard in Python because the 12 necessary statements (Table 7.2) have to be implemented only once in a function that can be used in all further exercises.

	A	B	C	D	E	F	G	H
1	n	400			f	0.10	spring constant	
2	dt	0.20 =80/n			d	0.00	damping constant	
3		n=400 ="n"&n						
4	$=A6+dt$ $=B6+(C6+F6)/2*dt$ $=C6+(D6+G6)/2*dt$ $=x+v*dt$ $=v+aA*dt$ $=f*xD-d*vD$							
5	t	x	v	aA	xD	vD	aD	
6	0.00	1.00	0.00	-0.10	1.00	-0.02	-0.10	
7	0.20	1.00	-0.02	-0.10	0.99	-0.04	-0.10	
406	80.00	0.98	-0.06	-0.10	0.97	-0.08	-0.10	

Fig. 7.3 (S) An oscillation of a mass-spring system is calculated with LookAhead. The time runs vertically down, the look-ahead to the end of the current time segment is performed horizontally to the right. The mass is assumed to be $m = 1$

7.2.2 Data Structure and Nomenclature

- n number of points for a time span of 80 s
- f, d spring and damping constants
- dt = $80/n$, length of the time interval
- t array of time points, $(n + 1)$ elements, dt apart
- x deflection of the mass at t (array)
- v velocity of the mass at t (array)
- a acceleration at t (array)
- x_A, v_A deflection and velocity at a specific time t_A
- x_D, v_D deflection and velocity predicted for time $t + dt$
- x_B, x_C deflection predicted at time $t + dt/2$ (Runge-Kutta)
- v_B, v_C velocity predicted for time $t + dt/2$ (Runge-Kutta)

All calculations are performed for $t = 0$ to $t = 80$ s.

7.2.3 Spreadsheet Calculation

The basic spreadsheet layout for integrating Newton’s equation of motion with *LookAhead* is shown in Fig. 7.3 (S), with the resulting deflection displayed in Fig. 7.1.

The time t in column A runs from top to bottom, and thus so does the deflection $x(t)$ of the mass in column B and its velocity $v(t)$ in column C. For a step from t_n to t_{n+1} :

- the acceleration a at the beginning of the time segment beginning with t_n is calculated in each row from the values of x and v at that time (column D),
- with these values, the deflection x_D and the velocity v_D at the end $t_n + dt$ of the time segment are predicted (columns E and F),
- with these new values, the acceleration a_D at the end of the time interval is estimated in column G.

All calculations reported for columns D to G use formulas with named variables referring to the same row. The new values for x and v at the next instant t_{n+1} of time are calculated in the following row with information from the previous row; see the formulas in B4 and C4 valid for B7 and C7, e.g., $B7 = [B6 + (C6 + F6)/2*dt]$.

The estimation of the mean value at the time $t_{n+1} = t_n + dt$ can be improved by predicting further values for speed and acceleration in the current time segment $[t_n, t_n + dt)$ in further columns, especially with the more efficient fourth-order *Runge-Kutta* method. However, due to the simpler table layout, we use *LookAhead* as our standard in spreadsheet calculations.

7.2.4 Python

Progress functions

In the second cell of Table 7.1, we have implemented the *LookAhead* method as a function, similar to the spreadsheet calculation of Fig. 7.2 (S). In the first cell of Table 7.1, the *Euler* method is implemented. In both functions, the acceleration is not calculated explicitly, but is rather outsourced to another function *acc* that must be specified in the main program.

The Runge-Kutta method of the 4th order is implemented in a function reported in Table 7.2, with two jumps into the center of the interval (x_B, v_B and x_C, v_C) and one jump to the end of the interval (x_D, v_D). The values (x_R, v_R) for the next time instants to be returned are calculated as a weighted average over the velocities or the accelerations.

Ψ Half, half, whole; the halves count twice.

The main program is shown in Table 7.3. In the first 4 lines, the parameters of the exercise are defined, followed by the definition of the function *accSpring* for the acceleration which makes use of the global parameters f and d . The arrays t , x , v are defined with their length corresponding to the specified number n of time instants, with the first entries containing the initial conditions set in lines 12–14.

Table 7.1 Functions performing the *Euler* and the *lookAhead* methods; the current values of $x(t)$ and $v(t)$ are passed as x_A and v_A

1	def Euler(xA, vA) :	1	def lookAhead(xA, vA) :
2	aA=acc(xA, vA)	2	aA=acc(xA, vA)
3	xR=xA+vA*dt	3	xD=xA+vA*dt
4	vR=vA+aA*dt	4	vD=vA+aA*dt
5	return xR, vR	5	aD=acc(xD, vD)
		6	xR=xA + (vA+vD) / 2*dt
		7	vR=vA + (aA+aD) / 2*dt
		8	return xR, vR

Table 7.2 Function implementing the Runge-Kutta method

```

1  def RungeKutta (xA, vA) :
2      aA=acc (xA, vA)
3      xB=xA+vA*dt/2 #Half step into center of interval
4      vB=vA+aA*dt/2
5      aB=acc (xB, vB) #Accel. at center of interval
6      xC=xA+vB*dt/2 #Second half step
7      vC=vA+aB*dt/2 #
8      aC=acc (xC, vC) #Accel. at center of interval
9      xD=xA+vC*dt #Full step
10     vD=vA+aC*dt #
11     aD=acc (xD, vD) #Accel. at end of interval
12     xR=xA+ (vA+2*vB+2*vC+vD) *dt/6
13     vR=vA+ (aA+2*aB+2*aC+aD) *dt/6
14     return xR, vR #Position, velocity at end of interval

```

Table 7.3 Calculating the motion of an oscillator

```

1  n=30 #Number of time instants
2  dt=80/n
3  f=0.10 #Spring constant
4  d=0.0 #Damping constant
5
6  def accSpring(x, v) :
7      return -f*x-d*v
8
9  t=np.zeros (n)
10 x=np.zeros (n)
11 v=np.zeros (n)
12 t[0]=0 #Initial conditions
13 x[0]=1
14 v[0]=0
15 acc=accSpring #acc = Name in progress function
16
17 for i in range(n-1): #Progress loop
18     t[i+1]=t[i]+dt
19     #progr=Euler; lbl1="Euler, n="+str(n)
20     #progr=lookAhead; lbl1="lookAhead, n="+str(n)
21     progr=RungeKutta; lbl1="RungeKutta, n="+str(n)
22     x[i+1], v[i+1]=progr(x[i], v[i])

```

Progress loop

The integration is performed in a progress loop (line 17–22) progressing by time steps dt . It recurs to the progress function *progr* that has no code of its own but is linked to one of the progress functions *Euler*, *lookAhead*, or *RungeKutta*. To keep

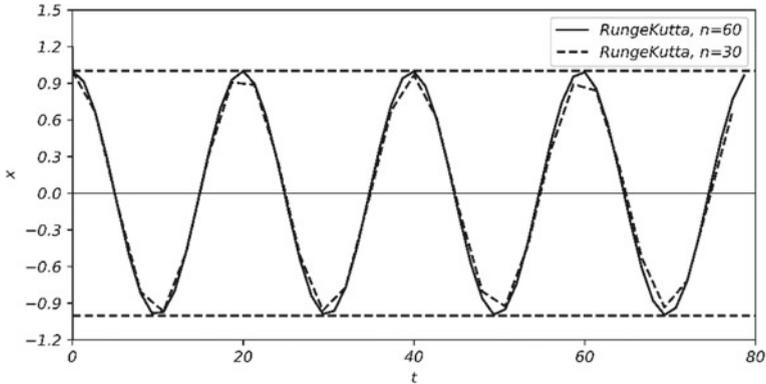


Fig. 7.4 Oscillation of a mass-spring system calculated with 4th order Runge–Kutta (with Table 7.3)

track of the program flow it is necessary to create the labels for the corresponding figures in the same program cell.

Within the progress functions, in Table 7.1, the acceleration is invariably taken from a function *acc* that, again, has no code of its own but must be linked to a function that is specific to the physical problem under consideration, here, in line 15, to *accSpring*. This increases the versatility of programming in that different approaches can be tried out.

To get stable results, we have to choose $n = 20,000$ for *Euler*, 400 for *LookAhead*, and 60 for 4th order *Runge-Kutta* ($n = 30$ is not good enough). The results of *Runge-Kutta* calculations are shown in Fig. 7.4.

7.3 Falling from a (Not Too) Great Height

Two forces act on a body falling from a great height: constant gravity and friction proportional to the square of the velocity. For these settings, the acceleration depends only on the velocity of the body, so that the location does not have to be calculated synchronously but can be determined subsequently by integration over the velocity. Initial acceleration and stationary velocity are compared with analytical solutions.

7.3.1 Limiting Cases, Analytically Solved

We drop a body from a great height and let two forces act on it: the gravity $-m \cdot g$, assumed to be independent of the height, and a friction force $k \cdot v^2$, proportional to the square of the speed (Newtonian friction) and opposite to its direction. With these assumptions, the equation of motion results as

$$a = -g - \frac{k}{m} v^2 \cdot \text{sgn}(v) = -g - \frac{k}{m} v \cdot |v| \tag{7.9}$$

where sgn (signum) is a function that calculates the sign of the argument. The force $m \cdot a$ depends only on the speed, because we assume, simplifying, that gravity and friction forces do not depend on the height. So, we don't have to consider the location in the calculation and may calculate it afterward by integrating the velocity trajectory. For a fall through air, $k = \rho A/2$ is valid with the density ρ of the air and the cross-sectional area A of the body.

In Fig. 7.5a, velocities $v(t)$ obtained with *Runge-Kutta of the 4th-order* (“RK4”) for two interval lengths $dt = 2$ and $dt = 4$ and with *LookAhead* for $dt = 0.5$ coincide at their points of calculation. Halving the interval length does not change the RK4 solution, so $dt = 4$ is good enough. For *LookAhead*, an interval length eight times smaller is necessary to obtain the same result.

In Fig. 7.5b, $v(t)$ trajectories are shown for three different initial velocities: zero, upwards, and downwards. The final speed is independent of the initial velocity.

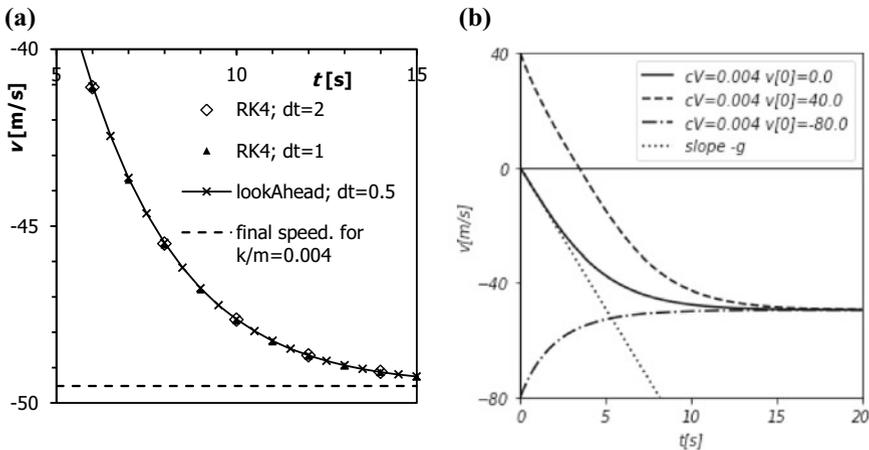


Fig. 7.5 a (left) Velocity of the falling object calculated in a spreadsheet with Runge-Kutta of the 4th order (RK4) and lookAhead. b (right) Velocity of the falling object as a function of time for different initial velocities, calculated with RK4 in Python

If, at the beginning of the fall, the velocity is 0, $v(0) = 0$, the initial gradient of the velocity curve is $-g$; the initial acceleration is the acceleration due to gravity close to the surface of the earth.

After a certain time, the body falls at a constant speed, because the frictional force equals gravity. This final velocity results from the condition $a = 0$ (no further acceleration):

$$v_{Fin} = -\sqrt{\frac{m \cdot g}{k}} \quad (7.10)$$

7.3.2 Data Structure and Nomenclature

g acceleration due to gravity
 $c_V = k/m$, (see Eq. 7.9) pre-factor of friction
 t array of time instants dt apart
 v_A velocity at t

Accelerations

a_A at the beginning of the calculation interval
 a_B, a_C in the center
 a_D at the end.

7.3.3 Spreadsheet

Runge-Kutta of the 4th order

The *Runge-Kutta* method implemented in a function requires 10 additional statements (1 for A, 3×3 for B, C, D, see Table 7.1). A corresponding spreadsheet layout would require 10 additional columns. As our problem is independent of the location, we need only $1 + 3 \times 2 = 7$ columns. This is realized in Fig. 7.6 (S).

The values in any row within the range 6–26 of columns C to I are valid for the interval $[t_n, t_{n+1})$ and are obtained as follows:

- (1) The acceleration a_A at time t_n is calculated from the velocity at that time (column C).
- (2) The velocity v_B is calculated with the values v_A and a_A for the middle of the interval, i.e., for $t_n + dt/2$, and from that, the corresponding acceleration a_B (first half-step, column E) is obtained.

	A	B	C	D	E	F	G	H	I	J
1	g	9.81 m/s ²	cV=k/m:=0.004							
2	cV	0.004 1/m								
3	dt	2.00	RK4; dt=2 = "RK4; dt="&dt							
4	=A6+dt	$=B6+(C6+2*E6+2*G6+I6)*dt/6$ $=-g-cV*vA^2*SIGN(vA)$ $=vA+aA*dt/2$ $=-g-cV*vB^2*SIGN(vB)$ $=vA+aB*dt/2$ $=-g-cV*vC^2*SIGN(vC)$ $=vA+aC*dt$ $=-g-cV*vD*ABS(vD)$								
5	t	vA	aA	vB	aB	vC	aC	vD	aD	
6	0.0	0.00	-9.81	-9.8	-9.43	-9.4	-9.45	-18.9	-8.38	
7	2.0	-18.65	-8.42	-27.1	-6.88	-25.5	-7.20	-33.1	-5.44	
26	40.0	-49.52	0.00	-49.5	0.00	-49.5	0.00	-49.5	0.00	

Fig. 7.6 (S) Calculation with 4th order *Runge-Kutta*; v_B , a_B , v_C and a_C are predicted values in the middle of the time interval, v_D and a_D are predicted values at the end of the time interval

- (3) The velocity v_C and, from that, the acceleration a_C for $t_n + dt/2$ are calculated a second time (second half step, columns F and G), now with the values a_B and v_B .
- (4) The velocity v_D at the end of the interval, i.e., for $t_n + dt$, is calculated with the values a_C and v_C and, from that, the acceleration a_D (whole step, columns H and I)

The new value of the velocity at t_{n+1} is calculated in the next row (e.g., in B7 with the formula in B5) from the velocity at t_n and a weighted average of the four accelerations calculated in the previous row (here row 6) for the interval $[t_n, t_{n+1}]$:

$$v_{n+1} = v_n + \langle a \rangle \cdot dt \text{ with } \langle a \rangle = (1a_A + 2a_B + 2a_C + 1a_D)/6 \tag{7.11}$$

The accelerations from the half steps count twice. We remember the procedure with a broom rule:

- Ψ *Half, half, whole, the halves count twice.* (4th order Runge-Kutta). “Halves” means half-steps.

Questions

concerning Fig. 7.6 (S):

Which accelerations from steps (1)–(4) are used in the formulas for velocities reported in F4, H4, and B4?⁴

Why is the sum in B7 divided by 6 to get the average, although only four accelerations enter the formula?⁵

⁴ a_B in the center of the time interval after the first jump; a_C in the center after the second; a_A , $2a_B$, $2a_C$, a_D for $t = 2$.

⁵ Ψ *Half, half, whole; the halves count twice.* The sum of the weights is 6.

Fig. 7.5a shows two solutions obtained with Fig. 7.6 (S) for $dt = 4$ and $dt = 2$.

► **Task** Determine the height fallen until you reach 95% of the final speed! You have to integrate the speed over time!

To get the three functions $v(t)$ displayed in Fig. 10.5b, we have to run the calculation three times with different initial values written into B6 and store the results $v_A(t)$ in extra columns (COPY, PASTE SPECIAL/CONTENTS).

7.3.4 Python

Table 7.4 reports the preparatory instructions for the progress loop. A special feature of the code is the stacking of x and v into a 2D matrix to store the fall trajectories for three initial conditions.

In lines 1–4 of Table 7.4, the fall parameters and the function for the acceleration are defined. In lines 8–10, the elementary arrays t , x , v are specified. In order to calculate the motion for three different initial velocities, position x and velocity v are replicated in `np.stack` constructions so that they can be addressed by the same index as the curve itself. Labels and line styles for the three curves in a diagram are also organized as lists with three entries in lines 16 to 19 in Table 7.5.

The progress of motion for the three initial velocities 0, 40, and -80 m/s is calculated in Table 7.5 in a nested loop. In the first loop over k , line 17, labels are created for the three initial velocities and in line 19, line styles before the progress is calculated.

The program for plotting the three time-curves is given in Table 7.6. The line styles and the labels have been specified in Table 7.5 so that curves, styles, and labels can all be addressed with the same index.

Table 7.4 Parameters for the fall and function for calculating the acceleration during the fall

```

1  g=9.81                #[m/s²] Gravitational acceleration
2  cV=0.004              #[1/m] Friction coefficient
3  def accFall(x,v):
4      return -g-cV*v**2*np.sign(v)
5  acc=accFall           #Link to acc in progress function
6
7  dt=0.05               #[s] Time increment
8  t=np.arange(0,20+dt,dt)
   #t sets the structure also for x and v, and determines the
   length of the progress loop.
9  x=np.zeros(len(t))
10 v=np.zeros(len(t))
11 xM=np.stack((x,x,x)) #For 3 initial conditions
12 vM=np.stack((v,v,v))

```

Table 7.5 Progress loop for the fall; “Runge–Kutta” is the function defined in Table 7.2

```

13  vM[0,0]=0
14  vM[1,0]=40
15  vM[2,0]=-80
16  lbl1=[0,1,2]
17  for k in range(3):           #Labels for three curves
18      lbl1[k]='cV='+str(cV)+' v[0]=''+str(vM[k,0])
19  linStyle=['k-', 'k--', 'k-.']
20  for k in range(3):           #Over all initial conditions
21      for i in range(len(t)-1): #Progress loop over t
22          xM[k,i+1],vM[k,i+1]= RungeKutta(xM[k,i],vM[k,i])

```

Table 7.6 Plotting three time-curves

```

23  FigStd('t[s]',0,20,5,'v[m/s]',-80,40,40)
24  for k in range(3): #Over initial conditions
25      plt.plot(t,vM[k],linStyle[k],label=lbl1[k])
26  plt.plot([0,20],[0,-20*g], 'k:', label = "slope -g")
    #Straight line with slope -g
27  plt.legend()

```

Questions

What is the shape of x_M and v_M in Table 7.4 and what are the first and last two elements of t ?⁶

What does “linStyle[k]” in line 25 of Table 7.6 do?⁷

How can the two k -loops in Table 7.5 be merged into one?⁸

7.4 Stratospheric Jump

When considering jumps from great altitudes, it must be taken into account that the coefficient of friction changes with height according to changes in air density. The friction force is, therefore, a function of both location and velocity. We model the air density in a simplified way with the barometric formula for a constant temperature (15 °C). In this model, the maximum speed for a jump from 39 km altitude is reached after a 50 s fall.

⁶ The shape of x_M and v_M is (401, 3), $\text{len}(t) = 401$, $t = [0, 0.05, \dots, 19.95, 20.00]$.

⁷ $\text{linStyle}[k]$ selects the k -th element of the list linStyle defined in line 19.

⁸ Insert line 18 between lines 20 and 21. The indentation with respect to line 20 is already correct.

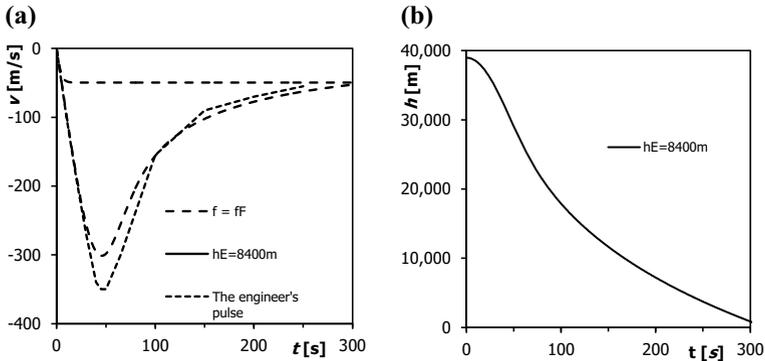


Fig. 7.7 **a** (left) Velocity for a fall from 38,969 m, compared with the data from Baumgartner's jump ("The engineer's pulse", Fig. 7.8a); the dashed line "f=fF" is valid for constant friction. **b** (right) Altitude as a function of time for the velocity calculated in **a**

t [s]	0	10	20	30	40	45	50	65	100	150	200	250
v [m/s]	0	-100	-190	-270	-340	-350	-350	-300	-155	-90	-70	-55

	A	B	C	D	E	F	G	H	I	J
3	dt	0.4 s			fF	0.004 1/m				
4	g	9.81 m/s ²			hE	8400 m		hE=8400m		
5	=A7*dt	=B7+(D7+H7)/2*dt	=fF*EXP(-h/hE)	=D7+(E7+I7)/2*dt	=-g+f*v^2	=h+v*dt	=fF*EXP(-hD/hE)	=v+a*dt	=-g+fD*vD^2	
6	t	h	f	v	a	hD	fD	vD	aD	
7	0.0	38969	0.0000	0.00	-9.81	38969	0.0000	-3.92	-9.81	
8	0.4	38968	0.0000	-3.92	-9.81	38967	0.0000	-7.85	-9.81	
807	320.0	-150	0.0041	-49.45	0.15	-170	0.0041	-49.39	0.15	

Fig. 7.8 **a** (top) Measured data of the jump of October 14, 2012 (The engineer's pulse, Oct.15, 2012, Mechanical analysis of Baumgartner's dive (Part B)). **b** (bottom, S) Spreadsheet layout for a jump from a high altitude, when the friction coefficient (*f* and *f_D* in columns C and G) is height-dependent

On October 14, 2012, the Austrian adventurer Felix Baumgartner ascended into the stratosphere in a helium balloon. He jumped out at a height of about 39 km and fell about 34 km in free flight, reaching supersonic speed before opening his parachute. Some measurement data are reported in Fig. 7.8a and plotted in Fig. 7.7a as a dashed line. We are going to simulate this jump with a simple model for height-dependent friction.

Height-dependent friction force

The coefficient of friction depends on the altitude, because the density of air, and, with it, the coefficient of friction, decreases with increasing altitude. We apply a height-dependent friction coefficient that directly corresponds to the barometric formula:

$$f(h) = f_F \cdot \exp\left(-\frac{h}{h_e}\right) \tag{7.12}$$

with $f_F = 0.004$ 1/m valid for $h = 0$ (the same as in Exercise 7.3). With $h_e = 8400$ m (valid for an air temperature of 15 °C), we get the best coincidence with the empirical data for the velocity as a function of altitude (see Fig. 7.7a).

Our simple model can well reproduce the essential characteristics of the time trajectory of the velocity. In the beginning, the fall is free, i.e., without friction, showing up as a linear increase in speed. After about 50 s, the maximum fall speed is reached; in the real experience, it exceeded the speed of sound. After that, the speed decreases sharply because of increasing friction due to increasing air density and approaches the stationary speed we calculated for $f = f_H = c_0$ in Exercise 7.3.

For a more realistic simulation, we have to consider that the atmosphere consists of different air layers with different temperatures. But we are content to find the main characteristics of the jump when applying just the barometric formula with constant temperature.

7.4.1 Data Structure and Nomenclature

t	array of time points, dt apart, independent variable
h_E	characteristic height of the barometric formula
f_F	friction at $h = 0$
h	height for t
f	friction coefficient at h
v	velocity at t
a	acceleration at t
h_D, f_D, v_D, a_D	values predicted for the end of the interval.

7.4.2 Spreadsheet Calculation

The spreadsheet layout for the integration of the equation of motion is presented in Fig. 7.8b. The integration method is *LookAhead*. The main change from Exercise 7.3.2 is that the height is now calculated simultaneously with the velocity. This is necessary because the coefficient of friction is height-dependent.

The height is calculated in column B from the height and the speed in the preceding rows; from that, in column C, the coefficient of friction f , which is then used to calculate the acceleration a in column E. Analogously, the same applies to the “looked-ahead” values in columns F to I.

Table 7.7 Parameters for the stratospheric jump, together with a function for the acceleration

```

1  n=801          #Number of calculation points
2  dt=320/(n-1)  #Time span 320 s
3  g=9.81        #[m/s2], Gravitational acceleration
4  hE=8400       #Characteristic barometric height
5  fF=0.004      #[1/m] Damping constant
6
7  def accJump(xi,vi):      #x represents height
8      f=fF*np.exp(-xi/hE) #Barometric formula
9      ac=-g+f*vi**2
10     return ac

```

Questions

Which thermodynamic quantities determine the characteristic height h_E ?⁹

The formulas for a and a_D in Fig. 7.8b (S) are valid only when the velocity is always negative, so that the body is always falling. Sloppy programming! How do the formulas have to be changed when the body can also rise?¹⁰

Interpret the formula in B8:B807 in the form $h(t+dt)=!$ ¹¹

The calculated velocity is shown in Fig. 7.7a as a function of time and compared with the empirical data of the stratospheric jump by Baumgartner.

7.4.3 Python

The parameters for the Python program are specified in Table 7.7, together with a function for the acceleration, *accJump*.

The main program is reported in Table 7.8 with the specification of the arrays t , h , and v and their initial values for $t = 0$. The for-loop is identical to all our programs for integrating the equation of motion. We have to assign the name *acc*, called in our standard function *RungeKutta*, to the acceleration function specific to the current problem.

7.5 A Car Drives with Variable Power

A driver wants to accelerate his vehicle at full throttle, but making sure that the wheels are not spinning (a “racing start”). We consider two approximations for this behavior, constant power and speed-proportional power. We

⁹ The height h_E arises from the Boltzmann distribution of air molecules in the atmosphere: air density $\rho(h) = \rho(0) \cdot \exp(-mgh/k_B T)$; $h_E = k_B T/mg$, where m is the average mass of the air molecules.

¹⁰ Instead of $[= \dots + v^2]$ one sets $[= \dots + \text{abs}(v)^2 * \text{SIGN}(v)]$ or $[= \dots - v * \text{abs}(v)]$.

¹¹ Formula in B8:B807: $h(t + dt) = h(t) + (v(t) + v_D(t))/2 \cdot dt$.

Table 7.8 Progress loop for the stratospheric jump

```

11 t=np.empty(n)      #n determines the length of t, h, and v
12 h=np.empty(n)
13 v=np.empty(n)
14 t[0]=0
15 h[0]=38969        #[m] Jump height
16 v[0]=0
17
18 acc = accJump      #acc() accessed within RungeKutta
19 for i in range(n-1):
20     t[i+1]=t[i]+dt
21     h[i+1],v[i+1]=RungeKutta(h[i],v[i])

```

answer the following two questions: How does the final speed depend on the frictional resistance? How do longer-term fluctuations in power become noticeable? The programming challenge is to realize case distinctions.

7.5.1 Various Types of Power

Analytical solution for vanishing sliding friction

In this section, we characterize the racing start of a car (“full throttle”) by a constant power P during the whole process (not by constant acceleration, as is usually assumed in textbook exercises). Without friction, the work W is converted into kinetic energy:

$$W = P \cdot t = \frac{m}{2} v^2 \rightarrow v = \sqrt{\frac{2P}{m}} \cdot \sqrt{t} \quad (7.13)$$

The resulting velocity v and the acceleration $a = dv/dt$ are shown in Fig. 7.9a for $P = 100$ kW and a car with mass $m = 1500$ kg. The velocity increases with the square root of time, and the acceleration becomes infinitely large at $t = 0$.

Non-zero motion friction and limited acceleration

In the real case, two effects have to be considered: (1) motion-inhibiting friction (rolling friction, driving resistance) and (2) the condition that the driving force must never be greater than the static friction force between the tire and the road. The second effect is taken care of by limiting the acceleration to a maximum value a_{Max} . We generally set the frictional force that inhibits motion proportional to v^{n_F} so that the exponent n_F can later be selected as a parameter, e.g., $n_F = 2$ for friction caused by an air stream.

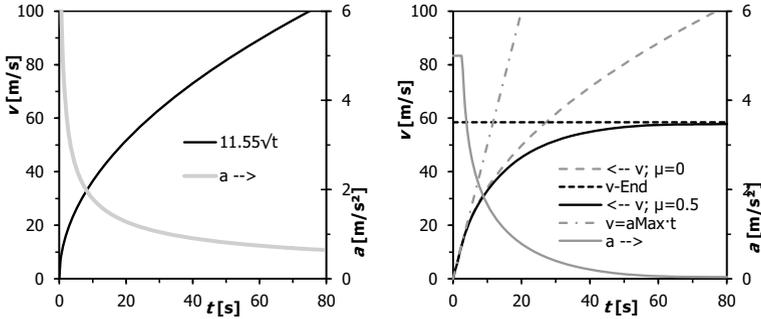


Fig. 7.9 **a** (left) Speed and acceleration for a constant power of $P = 100$ kW, without friction losses, calculated according to 7.13. **b** (right) Speed and power if driving friction is taken into account and acceleration is limited to a_{max} ; numerically calculated

The power needed to increase the kinetic energy and to overcome friction is

$$P = \frac{d}{dt} \left(\frac{m}{2} v^2 \right) + v \cdot F_R \tag{7.14}$$

$$F_R = \mu \cdot v^{n_F} \rightarrow P = m \cdot v \cdot \frac{dv}{dt} + \mu \cdot v^{n_F+1} \tag{7.15}$$

The acceleration

$$a(t) = \frac{dv}{dt} = \frac{P}{m \cdot v(t)} - \frac{\mu}{m} \cdot v(t)^{n_F} \tag{7.16}$$

does not depend on the location. It can be calculated from the power and the current velocity alone.

Question

The velocity in this task is always greater than or equal to zero. How do Eqs. 7.15 and 7.16 have to be changed if negative velocities are allowed?¹²

For stationary motion ($v = \text{const.}$), we get

$$a(t) = \frac{dv}{dt} = 0 \rightarrow v_s(t) = \sqrt[n_F+1]{P/\mu} \tag{7.17}$$

The stationary velocity v_s can easily be calculated analytically with Eq. 7.17, which we use to check the numerical calculation. This check also works the other

¹² $v(t) \rightarrow \text{Abs}(v(t)); v(t)^{n_F} \rightarrow \text{Abs}(v(t))^{n_F} \cdot \text{sgn}(v(t))$

way around, because, if numerically and analytically calculated values do not match, this may also indicate that our mathematical derivations are erroneous. It is, therefore, doubly good to compare the results of the two methods; we check whether our logical reasoning and our numerical calculation are consistent.

The calculated curves for the acceleration and the speed under friction can be seen in Fig. 7.9b. The acceleration at the beginning of the journey is at its pre-set maximum value 5 m/s^2 (for the acceleration, the right ordinate is valid) and goes practically to zero within 80 s. The speed is limited; the numerically calculated curve converges towards the stationary value obtained from Eq. 7.17. We can, therefore, assume that we did not make any gross programming mistakes. Without friction ($\mu = 0$), the velocity would continue to increase as in Fig. 7.9a.

Fluctuating power

What is the impact of power fluctuations on the speed? The answer is found in Fig. 7.10a, similar to Fig. 7.9b, yet with the power always fluctuating by 10% after a time span $\Delta t = 3.5 \text{ s}$. Although the acceleration fluctuates on this scale, the speed hardly fluctuates. Speed is the integral of acceleration and, as such, averages over fluctuations. The fluctuations of the acceleration influence the speed more strongly when they remain constant over a longer duration.

► **Task** Change the time period Δt during which the power remains constant and observe the time course of a and v !

Speed-proportional power

For internal combustion engines, the power depends on the number of revolutions of the engine, and thus on the speed. If a function is known for this dependence, it can be included in the formula for the power. We investigate the simple case in which

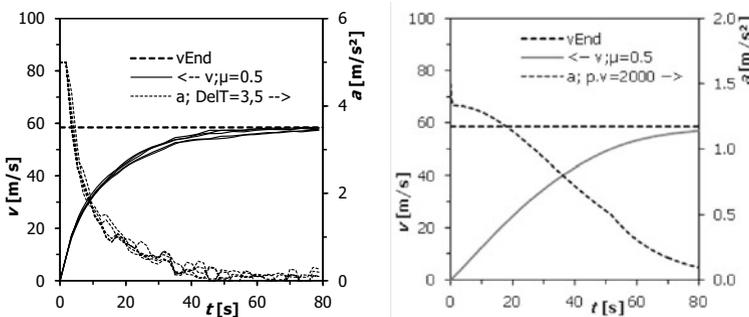


Fig. 7.10 **a** (left) Five different starting processes with a fluctuation of the power by about 10%; the power remains constant within $\Delta t = 3.5 \text{ s} = 10 \times dt$. **b** (right) A starting process for speed-proportional power for the parameter values $P_{min} = 1000$, $P_{max} = 100,000$ and $p_v = 2000$

the power P is proportional to the speed v :

$$P = p_v \cdot v \quad (7.18)$$

- ▶ **Tim** If the car is at a standstill, the power is zero, i.e., the car does not start at all.
- ▶ **Alac** That's why you put your foot down when you're idling and then let the clutch come.
- ▶ **Tim** How are we supposed to include that in our calculation model?
- ▶ **Mag** With a case distinction: If the power, according to Eq. 7.18, is too small, we use a constant value P_{\min} ; in real life, this would be achieved by letting the clutch slide.
- ▶ **Alac** That's for low speed. On the other end, at high revs, the engine runs out of breath.
- ▶ **Mag** Then we apply a constant power when exceeding a certain speed.
- ▶ **Tim** Is that really the way to do it?
- ▶ **Mag** Anyway, this is our model assumption. If this were a research project, we would have to compare the curves calculated according to the model with measured curves, and probably also consider that the gear would be changed at a certain speed.

In our model, we distinguish three cases:

- $v \cdot p_v \leq P_{\min}$ $P = P_{\min}$, clutch slides
- $v \cdot p_v \geq P_{\max}$ $P = P_{\max}$, engine at power limit
- $P_{\min} < v \cdot p_v \leq P_{\max}$ $P = v \cdot p_v$, speed - proportional power

The time curves velocity $v(t)$ and acceleration $a(t)$ are shown in Fig. 7.10b. The differences from Fig. 7.9b (and Fig. 7.10a) are clearly visible. At the beginning, with sliding clutch, the acceleration remains approximately constant, and then decreases linearly with time.

Question

What is the physical unit of p_v ?¹³

¹³ $[p_v] = [P/v] = \text{Ws/m} = \text{J/m}$.

7.5.2 Data Structure and Nomenclature

- m mass of the car
- μ friction coefficient for air resistance
- n_F power in friction law
- P default constant power
- P_{\min} power when the clutch slides
- P_{\max} engine at the power limit
- p_V coefficient for speed-proportional power
- a_{\max} maximum acceleration to avoid sliding
- t sequence of times, dt apart, independent variable
- P_{var} varying power as a function of t
- a acceleration at t
- v velocity at t
- v_S stationary velocity
- x distance travelled since $t = 0$

7.5.3 Excel

Constant power

For the numerical calculation, we use our standard method, “Progress with look-ahead” (see Fig. 7.11 (S)). The condition “driving force < static friction force” is taken into account by limiting the acceleration using a MIN function, $a = \text{Min}(a(t); a_{\max})$, where, for $a(t)$, Eq. 7.16 is to be used. The corresponding spreadsheet formula, valid for C6:C235, is reported in C4.

We do not set the velocity at time $t = 0$ in B6 to zero, but rather to 0.00001, so that, in C6, there is no division by 0. The value calculated in column C is

	A	B	C	D	E	F	G	H
1	dt	m	μ	n	P	aMax		
2	0.35	1500	0.5	2	100000	5.00		
3	<-- v; $\mu=0.5$		a -->					
4	$=A6+dt$ $=B6+(C6+E6)/2*dt$ $=\text{MIN}(P/m/v-\mu/m*v^n;aMax)$ $=v+a*dt$ $=v+a*dt$ $=\text{MIN}(P/m/vD-\mu/m*vD^n;aMax)$							
5	t	v	a	vD	aD		<-- v; $\mu=0$	
6	0.00	0.00001	5.00	1.75	5.00		0.00001	
7	0.35	1.75	5.00	3.50	5.00		1.75	
235	80.15	58.25	0.01	58.26	0.02		102.09	

Fig. 7.11 (S) Acceleration and velocity are calculated using the method “Progress with look-ahead”. The parameters of the problem are defined in row 2 with their names in row 1. The values in column G were copied from column B when μ was set to 0

thereby not affected, because, from the MIN condition, the maximum permissible acceleration a_{Max} results anyway.

Remember: In the calculation of the velocity v in a row (time t), only values from the previous row (time $t-dt$) are used. Therefore, the formula in B7 (reported in B4) gets input only from B6, C6, and E6. Formulas should only be entered into the cells in bold, here in A7 and B7. The rest of the respective columns is obtained by copying (“dragging down”). The initial values in A6 and B6 are entered as numbers. The formulas reported in C4:E4 are valid in their entire columns, e.g., C6:C235, because they do not refer to cell addresses, but rather to names for variables (cells and column ranges).

Questions

concerning Fig. 7.11 (S):

When is the acceleration in column C of constant?¹⁴

Examine and describe the expressions for a and a_D !¹⁵

Another technical note for the spreadsheet calculation: $10/5/2 = 1$. So, the formula is divided consecutively: $10/5 = 2$, then $2/2 = 1$. According to the rules of fractional calculation, the formula could be interpreted as $10/(5/2) = 4$, which is not so in EXCEL.

► **Task** Determine time and distance as a function of the power needed to accelerate to 95% of the final speed! To calculate the distance, you must integrate the velocity.

Temporally fluctuating power with the Mod function

How do the curves $a(t)$ and $v(t)$ change when the power does not remain constant, but rather fluctuates over time?

When the power fluctuates over time, we cannot treat it as a global constant P , but must insert an extra column in the spreadsheet, P_{var} , into which the power can be entered at any time. To calculate the acceleration, we must access this variable, and not the global parameter P .

In Fig. 7.12 (S), a calculation is presented in which the power fluctuates over a longer period of time, i.e., remains constant over a time interval Δt (DelT). The two variables r_T and P_{var} have been inserted into columns B and C for this purpose. The formulas for v , a , v_p and a_p are exactly the same as in Fig. 7.11 (S), but with the time-dependent power P_{var} instead of the constant power P .

¹⁴ As long as the acceleration calculated in column C according to Eq. 7.16 is greater than the value a_{Max} given in F2.

¹⁵ Think about it and read on!

	A	B	C	D	E	F	G	H	I	J	K
1	dt	DelT	P	DelP	m	μ	n	aMax			
2	0.35	3.4	100000	10000	1500	0.5	2	5.00			<-- v;μ=0.5
3											a; DelT=3,5 -->
4	=A6+dt	=ABS(MOD(t;DelT))	=IF(rT <=0.2; NORM.INV(RAND()); P; DelP); C6)	=D6+(E6+G6)/2*dt	=MIN(Pvar/m/v-μ/m*v^*n; aMax)	=v+a*dt	=MIN(Pvar/m/vD-μ/m*vD^*n; aMax)				
5	t	rT	Pvar	v	a	vD	aD				a; DelT=3,5
6	0.00	0.00	115530	0.00001	5.00	1.75	5.00		0.00	0.00	5.00
7	0.35	0.35	115530	1.75	5.00	3.50	5.00		1.75	8.75	5.00
15	3.15	3.15	115530	15.72	4.82	17.40	4.32		15.75	40.17	1.23
16	3.50	0.10	125449	17.32	4.73	18.97	4.29		17.50	42.20	0.98
235	80.15	1.95	104604	58.13	0.07	58.16	0.07		71.75	56.94	-0.11

Fig. 7.12 (S) The calculation of Fig. 7.11 (S) is extended with columns B and C, in which a time-dependent power is calculated, remaining constant over the time period *DelT*. Columns I, J and K are written by SUB *Vari* in Fig. 7.13 (P) with a selection of the data from columns A, B and C

In our model, we require that the power remain constant over a period of time *DelT*. This condition can be fulfilled with the MOD function (performing a modulo operation), whose action is visible in column B of Fig. 7.12 (S). The time *t* is divided by *DelT* and the remainder is returned; e.g., in row 16 $t/DelT = 3.5/3.4 = 0.1$, remainder (*modulus*) is 0.1, and 0.1 is returned.

The expression in C4 of Fig. 7.12 (S), valid for C7, outputs a new value for *P_{var}* only at certain times, otherwise, the value from the previous time is carried over. It is structured as follows:

```
IF (rT <= 0.2) THEN (Pvar new) ELSE (old value from the previous cell C6)
with Pvar new = NORM.INV(RAND (); P; DelP), setting a new power,
fluctuating around the mean value P with a standard deviation DelP.
```

In columns I, J, and K, time *t_R*, velocity *v_R*, and acceleration *a_R* are entered successively for five start processes using a macro (protocol procedure SUB *Vari* in Fig. 7.13 (P)): The curves are graphically represented in Fig. 7.10a. In every run,

1	Sub Vari()	For r = 6 To 235 Step 5	11
2	'The course of velocities is	Cells(r2, 9) = Cells(r, 1) 't.R	12
3	'calculated several times and	Cells(r2, 10) = Cells(r, 4) 'v.R	13
4	'stored in columns 9-11.	Cells(r2, 11) = Cells(r, 5) 'a.R	14
5	r2 = 6	r2 = r2 + 1	15
6	Cells(3, 10) = "a; DelT=" & Round(Cells(2, 2), 2)	Next r	16
7	For rep = 1 To 5	r2 = r2 + 1	17
8	Application.Calculation = xlCalculationManual	Application.Calculation = xlCalculationAutomatic	18
9		Next rep	19
10		End Sub	20

Fig. 7.13 (P) Rep-log procedure, copies five start operations (FOR REP = 1 TO 5) continuously into columns 9 to 11 (= I, J, K) of Fig. 7.12 (S). Only the data for every fifth time are transferred (line 11, ... STEP 5)

	A	B	C	D	E	F	G	H	I
1	dt	m	μ	nF	P	aMax		Pmin	1000
2	0.35	1500	0.5	2	100000	1.50		Pmax	100000
3								p.v	2000
4	=A6+dt =IF(v*p.v>Pmin;IF(v*p.v<Pmax;v*p.v;Pmax);Pmin) =C6+(D6+F6)/2*dt =MIN(Pv/m/v- μ /m*v^nF;aMax) =v+a*dt =MIN(Pv/m/vD- μ /m*vD^nF;aMax)								
5	t	Pv	v	a	vD	aD			
6	0.00	1000	0.00001	1.50	0.53	1.27			
7	0.35	1000	0.48	1.38	0.97	0.69			
235	80.15	100000	56.87	0.09	56.91	0.09			

Fig. 7.14 (S) Speed-proportional power $P_v = v \cdot p_v$ if it is bigger than P_{\min} and smaller than P_{\max}

only every fifth data point (“Step 5”) is transmitted.

Questions

What would the curves of v in Fig. 7.10a look like if, in Fig. 7.13 (P), line 17 were missing?¹⁶

How does $\text{del}P$ appear as a standard deviation in the fluctuating power in Fig. 7.12 (S)?¹⁷

Speed-proportional power

To get a power that is proportional to the speed of the car, Fig. 7.14 (S), we replace the formula in C4 of Fig. 7.11 (S), valid for C6:C235, with a nested if-loop with the outputs $v \cdot p_v, P_{\max}, P_{\min}$:

$$[= \text{IF}(v * p_v > P_{\min};$$

$$(\text{IF}(v * p_v < P_{\max}; v * p_v;$$

$$P_{\max}));$$

$$P_{\min}]$$

The power is $v \cdot p_v$, i.e., proportional to the speed, if it is bigger than P_{\min} and smaller than P_{\max} .

¹⁶ There would be no empty line separating the data sets, so that the last point of a curve would be connected to the first point of the following curve.

¹⁷ See formula in C4 where a formula for normal noise is applied.

Table 7.9 Parameter specification for driving with constant power

```

1 dt=0.35
2 m=1500           #[kg] Mass of the car
3 mu=0.5          #Friction coefficient
4 nF=2            #Power coefficient in friction law
5 P=100000        #[W] Power
6 aMax=5.0        #[m/s2] max. acceleration
7
8 def accPwr(x,v): #v is always positive
9     a=min(P/m/v-mu/m*v**nF,aMax)
10    return a
11
12 t=np.arange(0,80.15+dt,dt)
13 x=np.zeros(len(t))
14 v=np.zeros(len(t))
15 a=np.zeros(len(t))

```

Table 7.10 Progress loop with *RungeKutta* for constant power

```

16 v[0]=0.00001
17 acc=accPwr      #acc is name for acceler. within RungeKutta
18 for i in range(len(t)-1):
19     a[i]=accPwr(x[i],v[i])
20     x[i+1],v[i+1]=RungeKutta(x[i],v[i])

```

7.5.4 Python

Constant power

The setting of the parameters for driving with constant power is shown in Table 7.9. The size of the arrays is specified with `np.arange(0, 80.15 + dt, dt)` to yield exactly the same range as in the EXCEL spreadsheet of Fig. 7.11 (S). The acceleration is executed in a function `accPwr` that is assigned the additional name `acc` (line 17 in Table 7.10) expected in *RungeKutta*.

Progress of motion is achieved in Table 7.10 with the usual progress loop.

Question

Why do we calculate `a[i]` separately in line 19 of Table 7.10, although it is already calculated within *Runge-Kutta*?¹⁸

¹⁸ We want to plot $a(t)$, e.g., in Figs. 7.9 and 7.10.

Fluctuating power

To calculate the car's velocity for fluctuating power, the additional parameters $DelT$, $DelP$, and Pc are specified in Table 7.11. To store the results of five different runs, matrices v_M and a_M are built by stacking five times the arrays v and a , respectively (lines 6 and 7). Their shape is 5 rows \times 231 columns. The usual progress-loop contains an inner loop ($k =$) running over the 5 rows and calling *RungeKutta* successively with the individual $v_M[k, i]$ so that, in the end, five time series of velocity and acceleration will have been calculated.

The contents of a_M and v_M are completely calculated in Table 7.11 and plotted with the program in Table 7.12.

Table 7.11 Calculation of five time series of v and a for motion with fluctuating power

```

1 import numpy.random as npr
2 DelT=3.5 #Fluctuation period
3 DelP=10000 #Fluctuation amplitude
4 Pc=100000 #Mean power
5 P=Pc
6 vM=np.stack((v,v,v,v,v))
7 aM=np.stack((a,a,a,a,a))
8 for i in range(len(t)-1):
9     for k in range(5):
10         aM[k,i]=accPwr(x[i],vM[k,i])*10
11         PBef=P #Setting the (new?) power
12         rem=np.mod(t[i],DelT) #0 ≤ rem ≤ DelT
13         P=Pc+DelP*npr.randn() if rem<=DelT else PBef
14         x[i+1],vM[k,i+1]=RungeKutta(x[i],vM[k,i])
np.shape(vM) (5, 231)
np.shape(vM[0]) (231,)
```

Table 7.12 Plotting five time series of velocity and acceleration for motion with fluctuating power

```

1 FigStd('t',0,80,20,'v',0,60,20)
2 print('np.shape(t) ',np.shape(t))
3 print('np.shape(vM[0]) ',np.shape(vM[0]))
4 for k in range(5):
5     plt.plot(t,vM[k],'k-') #Full black
6     plt.plot(t,aM[k],'k:') #Dotted black
np.shape(t) (231,)
np.shape(vM[0]) (231,)
```

Table 7.13 Definitions for speed-proportional power

```

1 Pv=np.zeros(len(t))
2 Pmin=1000          #Min. power
3 Pmax=100000       #Max. power
4 pv=2000           #Pn=v*pv, proportionality factor
5 aMax=1.5          #Max. acceleration
6
7 def accPwrV(x,v): #Pn is specified in progress loop
8     a=min(Pn/m/v-mu/m*v**nF,aMax)
9     return a

```

Table 7.14 Loop for progress with *RungeKutta* for speed-proportional power

```

1 v[0]=0.00001
2 acc=accPwrV        #acc is name in RungeKutta
3 for i in range(len(t)-1):
4     if (Pmin<v[i]*pv<Pmax): Pn=v[i]*pv
5     elif (v[i]*pv<Pmin): Pn=Pmin
6     else: Pn=Pmax
7     Pv[i]=Pn
8     a[i]=accPwrV(x[i],v[i])
9     x[i+1],v[i+1]=RungeKutta(x[i],v[i])

```

Questions

How do you change the arguments in line 1 of Table 7.12 to get the same axis titles as in Fig. 7.10?¹⁹

How does the changing power P enter $accPwr(\dots)$?²⁰

Speed-proportional power

The additional parameters for driving with speed-proportional power are given in Table 7.13, together with a function $accPwrV(x, v)$ for calculating the acceleration with the input variables x and v , as is expected for the function acc applied in *RungeKutta*. The function $accPwrV$ makes use of global parameters. To keep control of the situation, the global parameters and the function should be defined in the same program cell. This is actually the case in Tab. 7.13, except for P_v , the speed-dependent power that depends on the speed calculated in the progress loop.

The progress loop for speed-proportional power in Table 7.14 contains a nested `if ... elif ... else` query to assign the correct value of the power to the variable P_n accessed as a global parameter in $accPwrV$.

¹⁹ `FigStd('t [s]', 0, 80, 20, 'v [m/s]', 0, 60, 20)`.

²⁰ The power P is a global variable changed within the progress loop.

7.6 Bungee Jump

A bungee jumper falls in free fall until the rope becomes tight; only gravity acts. When the rope is stretched, two additional forces come into play: the back-driving rope force and the friction force due to the inner friction of the rope. We neglect air friction so that the force depends only on the location, and not on the speed. The form of motion is, at times, a free fall or a damped vibration.

7.6.1 Simulation of the Motion

In a bungee jump, a person hangs on an elastic rope and lets her/himself fall from a great height into the depths. As an example, we consider a bungee jumper (mass $m = 60$ kg, size 1.65 m) jumping from a height of 50 m. The bungee rope is $l = 25$ m long in relaxed condition and has an elastic constant (corresponding to a spring constant) of $k = 100$ N/m. The zero of the z -axis is at the point where the rope is fixed. The jumpers move in free fall until the elastic rope is fully expanded but not yet stretched. For $l = 25$ m, this is at $z = -25$ m.

No friction

We first consider the situation without friction in which the jumper starts at height $z = -25$ m (fully elongated rope, not yet stretched; see Fig. 7.15a). The jumper oscillates like a mass-spring system with a period of $T = 2\pi/\sqrt{k/m} = 4.87$ s or $4 \cdot T = 19.47$ s.

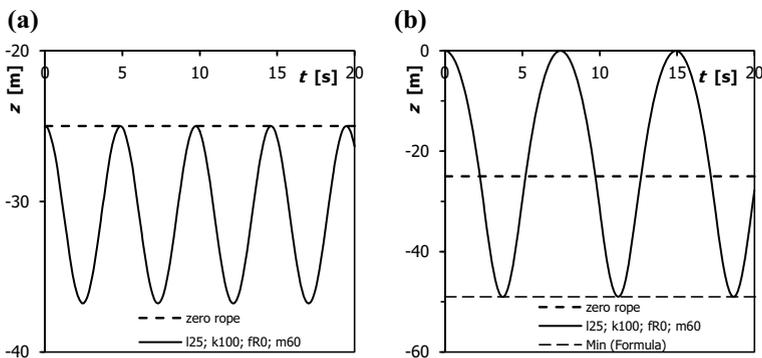


Fig. 7.15 **a** (left) Oscillation when the jumper starts with a fully elongated rope. **b** (right) Sequence of free fall and oscillation when the jumper starts at the fixing point of the rope

When the jumpers start at the fixing point of the rope at the height of $z = 0$ m, Fig. 7.15b, they:

- fall freely until the rope is fully elongated ($z = -25$ m),
- complete a half-period oscillation driven back by the elasticity of the rope to the zero-rope position ($z = -25$ m),
- and overshoot up to the original height $z = 0$ in free fall (starting upwards because of the initial upward velocity), where the height is a parabolic function of time.

This sequence of motions repeats periodically.

Friction force of the rope

We assume a *friction work* W_F proportional to the elongation or relaxation path of the rope, because, during elongation and relaxation, rope components are shifted against each other, thus dissipating energy. Consequently, the frictional force $f_R = dW_F/dh$ is constant. For the example of Fig. 7.16, we have chosen $f_R = 220$ N.

The motion with unexpanded rope (above $z = -l$) is still a free fall, whereas, with expanded rope, a dampened oscillation is observed.

The horizontal lines in Fig. 7.16 are obtained from analytical calculations of:

- the maximum height fallen,
- the top speed of the jumper,
- the rest position at the stretched rope.

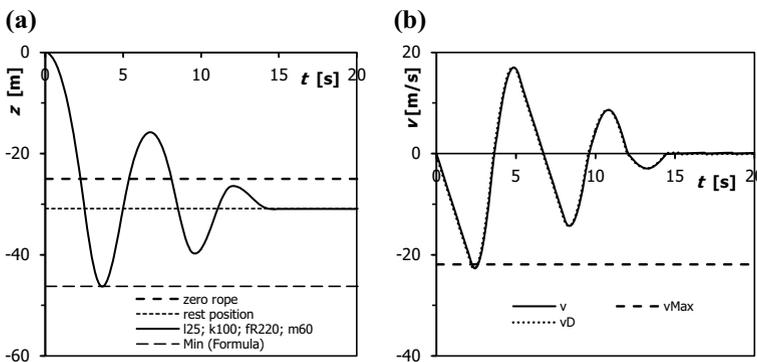


Fig. 7.16 a (left) Height as a function of time during a bungee jump, “zero rope” = length of unexpanded rope. b (right) Velocity as a function of time; v_D (dotted line) = look-ahead to end of the time segment, calculated with the acceleration at the beginning. Parameters specified in Fig. 7.18 and Table 7.15.

Question

What do you think: is rope friction effective only for elongation or also for relaxation of the rope?²¹

Reverse problem: The properties of the rope are deduced from the observed motion

In this exercise, we have specified values for the spring constant and the friction force of the rope and therewith calculated the time curves for the position and velocity of the jumper.

More interesting from a physical point of view is the reversal of the problem, in which the properties of the rope are deduced from an actually possible observation. A possible approach is to observe the oscillation period and measure the time it takes for the jumper to come to rest, and then adjust the parameters of the simulation model so that the simulation fits the measured results. We could then possibly see whether our assumptions about the rope's elastic and frictional properties and our neglect of air friction are justified.

7.6.2 Analytical Calculations

We calculate the minimum and the rest position, as well as the maximum speed by setting up the equation of motion with the starting point at $z = 0$:

$$m \cdot \ddot{z} = -m \cdot g \text{ for } z > -l \quad (7.19)$$

$$m \cdot \ddot{z} = -m \cdot g - (z - l) \cdot k - f_R \text{ for } z \leq -l \quad (7.20)$$

The parameter k is the spring constant coming into play when the jumper's position is lower than the length of the unexpanded rope.

First minimum by energy balance

The lowest point of the jump can be calculated from the energy balance. The kinetic energy disappears at the lowest point, being a reversal point; the gravitational energy has passed into the elastic energy of the rope and the friction energy. With $h =$ height fallen, the following quadratic equation applies:

$$m \cdot g \cdot h = \frac{(h - l)^2 k}{2} + f_R(h - l) \quad (7.21)$$

²¹ Elongation: gravitational energy is converted into kinetic, elastic, and friction energy.

Relaxation (rope becomes shorter): elastic and kinetic energy is converted into gravitational and frictional energy.

$$h^2 + h \cdot \left(-2l + \frac{2f_R}{k} - \frac{2mg}{k}\right) + \left(l^2 - f_R \frac{2l}{k}\right) = 0 \tag{7.22}$$

It is solved in Fig. 7.17 (S), Eq. A, for the parameters of Fig. 7.18.

Maximum speed of the Bungee jumper

The maximum speed of the jump is obtained by considering the dependence of the kinetic energy on the height of the fall:

$$E_{kin} = \frac{m}{2}v^2 = mgh - \frac{k}{2}(h - l)^2 - f_R(h - l) \tag{7.23}$$

The gravitational energy goes into the kinetic energy of the jumper, the elastic energy and the friction work of the rope.

The derivative of the kinetic energy with respect to the height gives the position of the maximum velocity:

$$\frac{dE_{kin}}{dh} = mg - k(h - l) + f_R = 0 \tag{7.24}$$

$$h = \frac{mg + kl + f_R}{k} \tag{7.25}$$

The maximum speed is calculated from the kinetic energy T_{max} at this altitude:

$$v_{max} = \sqrt{\frac{2T_{max}}{m}} \tag{7.26}$$

The value of v_{max} is calculated in Fig. 7.17 (S), Eq. (B), for the parameters of Fig. 7.16, specified in Fig. 7.18 (S).

Eq. (A)	Terms of the quadratic equation	p	-57.37 = $-2 \cdot l + 2 \cdot f_R / k - 2 \cdot m \cdot g / k$
		q	515.0 = $l^2 - f_R \cdot 2 \cdot l / k$
	Deepest point	h+	-11.14 = $p / 2 + \text{SQRT}((p / 2)^2 - q)$
		h-	-46.23 = $p / 2 - \text{SQRT}((p / 2)^2 - q)$
Eq. (B)	Hight for max. speed	hVm	33.09 = $(m \cdot g + k \cdot l + f_R) / k$
	Max. kinetic energy	Tmax	14426 = $m \cdot g \cdot hVm - k / 2 \cdot (hVm - l)^2 - f_R \cdot (hVm - l)$
	Max. speed	vMax	21.93 = $\text{SQRT}(2 \cdot Tmax / m)$
Eq. (C)	Rest position	hRest	30.89 = $m \cdot g / k + l$

Fig. 7.17 (S) Analytical calculation of the lowest point, the maximum speed, and the rest position of the bungee jump; p is the first parenthesis of Eq. 7.22, q the second. Eq. (A) = Eq. 7.22, Eq. (B) = Eq. 7.25 and Eq. 7.26, Eq. (C) = Eq. 7.28

Final position of the bungee jumper

When the jumper has come to rest, there is a balance between the weight force and the elastic rope force:

$$m \cdot g = k \cdot (h - l) \quad (7.27)$$

$$h = \frac{m \cdot g}{k} + l \quad (7.28)$$

calculated in Eq. (C) of Fig. 7.17 for the parameters of Fig. 7.18 (S).

► **Task** Determine the time until the jumper is at rest as a function of jump height!

► **Task** Vary k and F_R and observe how the number of oscillations and the time until rest change!

7.6.3 Data Structure and Nomenclature

g	gravitational constant 9.81 m/s ²
l	length of the rope
k	elastic constant of the rope
f_R	friction force of the rope, constant, energy = $f_R \cdot (h - l)$
m	mass of the jumper
t	sequence of times, dt apart
z	position of the jumper at t , rope fixed at $z = 0$
z_{Min}	minimum position of the jumper
z_{End}	final rest position of the jumper
v	velocity of the jumper as a function of t
v_{Max}	maximum speed
T_{Max}	maximum kinetic energy.

7.6.4 Excel

The parameters m , l , k , f_R , and g ($= 9.81 \text{ m/s}^2$) are specified in named cells in A1:F2 of Fig. 7.18 (S). H3 displays the formula (concatenation of text and variables) for obtaining the legend in H2.

The equation of motion of the bungee jump is integrated with our standard method, “Progress with look-ahead”. The result of the calculation is shown in Fig. 7.16.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	dt	g	l	k	m	fR			i				
2	0.05	9.81	25	100	60	220			I25; k100; fR220; m60				
3	s	m/s ²	m	N/m	kg	N			=C1&I&H1&D1&k&H1&"fR"&fR&H1&"m"&m				
4													
5		a=IF(z<-l;-g-SIGN(v)*fR/m-k/m*(z+l)-g)											
6		aD=IF(zD<-l;-g-k/m*(zD+l)-SIGN(vD)*fR/m;-g)											
7	=A9*dt	=B9+(D9+G9)/2*dt	=C9+(B9+E9)/2*dt	=v+a*dt	=z+(v+vD)/2*dt					=MIN(v)	=MIN(z)	=INDEX(z;401)	
8	t	v	z	a	vD	zD	aD			vMax	zMin	zEnd	
9	0.00	0.00	0.00	-9.81	-0.49	-0.01	-9.81			Simu	-22.70	-46.28	-30.95
10	0.05	-0.49	-0.01	-9.81	-0.98	-0.05	-9.81			Formula	-21.93	-46.23	-30.89
409	20.0	0.15	-30.9	-3.56	-0.03	-30.9	3.77						

Fig. 7.18 (S) Calculation model for a bungee jump; l = length of the rope; k = spring constant; m = mass of the jumper; f_R = friction constant of the rope; acceleration is inhibited by friction only when the rope is stretched; the formulas $a = IF(\dots)$ in row 5 and $a_D = IF(\dots)$ in row 6 are valid for the column vectors a and a_D , respectively. Bewegung wird gehemmt

Questions

Analyze the formula “ $a = \dots$ ” in B5 of Fig. 7.18 (S) applied in column D. What is the logical structure of the formula? When does only gravity act?²²
 Which forms of the time trajectory do you observe for an unstretched and a stretched rope?²³

The spreadsheet set-up presented here applies the same friction force for relaxing and stretching rope. Friction losses are likely to occur only when the rope is longer than in the unstretched state (discuss!); this is taken into account through an IF query.

The calculated height is shown in Fig. 7.16a as a function of time. For checking purposes, the prominent points of the jump calculated analytically, namely, maximum depth (= minimum height) and rest position, are entered into the diagram as horizontal lines. The agreement with the simulated curve is quite good, which indicates that neither our analytical calculations nor our simulation within the framework of our model contain gross errors;-) or, less likely, both errors cancel each other out.

In Fig. 7.16b, the velocity is plotted as a function of time, and also, as a horizontal line, the maximum velocity analytically calculated according to Eq. 7.26. The dashed curve represents the velocity foreseen at the end of the interval calculated with the acceleration at the beginning of the interval. The full curve shows the velocity at the beginning of the next interval, calculated from the mean of the two accelerations, the one at the beginning of the interval and the estimated one

²² IF(LOGICAL_TEST, [VALUE_IF_TRUE], [VALUE_IF_FALSE]). If the amount of the deflection is smaller than the rope length, then only the acceleration due to gravity acts; neither the elastic rope force nor the friction force are active. In the IF function, this is the [VALUE_IF_FALSE] case, i.e., when the condition ($z < -l$) is not fulfilled.

²³ Unstretched: free fall, parabola; stretched: damped oscillation around the rest position.

Table 7.15 Parameters and function for calculating the acceleration during a bungee jump

1	<code>g=9.81</code>	<code>#[m/s²]</code>
2	<code>l=25</code>	<code>#[m] Length of rope</code>
3	<code>k=100</code>	<code>#[N/m] Spring constant</code>
4	<code>m=60</code>	<code>#[kg] Mass of jumper</code>
5	<code>Fr=220</code>	<code>#[N] Frictional force</code>
6		
7	<code>def accBungee(z,v):</code>	<code>#v positive, negative, or 0</code>
8	<code> if z<-l:</code>	<code>#Tensioned rope</code>
9	<code> a=-g-k/m*(z+l)-np.sign(v)*Fr/m</code>	<code>#np.sign(0)==0</code>
10	<code> else: a=-g</code>	
11	<code> return a</code>	
12		
13	<code>lbl_1=("l"+str(l)+"; k"+str(k)+"; fR"+str(Fr)+"; m"+str(m))</code>	
	<code> lbl_1</code>	<code> l25; k100; fR220; m60</code>

Table 7.16 Progress of the bungee jump

14	<code>dt=0.251</code>	<code>#[s] Time increment</code>
15	<code>t=np.arange(0,100+dt,dt)</code>	
16	<code>v=np.zeros(len(t))</code>	<code>#Velocity</code>
17	<code>z=np.zeros(len(t))</code>	<code>#Height</code>
18	<code>acc=accBungee</code>	<code>#acc(...) is accessed in RungeKutta</code>
19	<code>v[0]=0</code>	<code>#Initial conditions</code>
20	<code>z[0]=0</code>	
21	<code>for i in range(len(t)-1):</code>	
22	<code> z[i+1],v[i+1]=RungeKutta(z[i],v[i])</code>	

at the end of the interval. The difference between the two curves illustrates the difference between the *Euler* and the *lookAhead* methods.

7.6.5 Python

The parameters of the bungee jump and the function *accBungee* for calculating the acceleration of the jumper are given in Table 7.15. The progress of the motion of the jumper is calculated in Table 7.16.

7.7 Questions and Tasks

From integration to numerical averaging

1. How is the velocity $v(t)$ related to the acceleration $a(t)$?

2. How is the mean value of a continuous function $f(t)$ within the range t_1 to t_2 defined?
3. Interpret the formula $v(t_n + dt) = v(t_n) + [a(t_n) + a(t_n + dt)]/2 \cdot dt$ with respect to the previous two questions!
4. What does the broom rule (*Ψ half, half, whole, the halves count twice*) tell us about the 4th order Runge-Kutta method?

Half-step procedure in a spreadsheet

In Fig. 7.19 (S), you can see a spreadsheet structure for the numerical integration of Newton’s equation of motion for a fall with friction using the half-step method. The spreadsheet calculation is structured in the same way as for our "Progress with Forecast" procedure. However, the predicted acceleration a_P is calculated for the middle of the interval and is considered representative of the entire time interval.

5. What are the formulas for x_P , v_P , and a_P ?
6. What are the formulas for x and v at the beginning of the next time interval (in C7 and D7)?

Function for “Progress with look-ahead” or “half step”

In Table 7.17, you can see the code of a Python function for calculating the progress from t to $t + dt$. However, in lines 4 to 8, the formulas have been replaced by 1.

7. Insert the appropriate formulas if “Progress with look-ahead” is to be used!

	B	C	D	E	F	G	H	I	J
1	g	9.81 m/s ²		<i>gravitational acceleration</i>					
2	cV	1.5 1/m		<i>friction coefficient</i>					
3	dt	0.005 s		<i>time increment</i>					
4	$=B6+dt$ $=C6+G6*dt$ $=D6+H6*dt$ $=-g-cV*v^2*SIGN(v)$ $=x+v*dt/2+a/8*dt^2$ $=v+a*dt/2$ $=-g-cV*vP^2*SIGN(vP)$								
5	t	x	v	a	xP	vP	aP		
6	0.000	4.000	1.000	-11.31	4.00	0.97	-11.23		
7	0.005	4.005	0.944	-11.15	4.01	0.89	-10.99		
406	2.000	-0.361	-2.557	0.00	-0.37	-2.56	0.00		

Fig. 7.19 (S) Falling with friction, half-step procedure: The velocity v_P and the acceleration a_P in the middle of the interval are calculated. Formula in C4 is valid for C7

Table 7.17 Function for numerical integration of the Newtonian equation of motion

1	dt=0.1			
2	def ForcA(x, v):		6	aA=1
3	a=acc(x, v)		7	xN=1
4	xA=1		8	vN=1
5	vA=1		9	return(xN, vN)

8. Insert the appropriate formulas if the half-step method is to be used!

Power and work

9. How are work W and power P related to the force F , the body's displacement x , and the velocity v ?

Bungee jumping

10. What are the formulas for: a) the elastic energy of a spring with constant k , b) the gravitational energy of a mass m near the earth's surface, and c) the kinetic energy of a mass m ?

Consider the statement: "The frictional energy when stretching or relaxing a rope is proportional to the change in length (absolute value) of the rope."

11. How is this approach justified?
 12. Which function results for the frictional force?

Mathematical pendulum

The oscillation equation for a mass-spring system is

$$\frac{\partial^2 x(t)}{\partial t^2} = -\frac{k}{m} \cdot x(t) \quad (7.29)$$

The oscillation equation for a mathematical pendulum is

$$\frac{\partial^2 \phi(t)}{\partial t^2} = -\frac{g}{l} \cdot \sin(\phi) \quad (7.30)$$

13. Why does no mass appear in Eq. 7.30 as does in Eq. 7.29?
 14. Why does $\sin(\phi)$ turn up on the right side of Eq. 7.30, and not simply ϕ , as one might expect in analogy to Eq. 7.29?

Random Numbers and Statistical Reasoning

8

We perform statistical experiments by applying functions that generate uniformly distributed, normally distributed, and \cos^2 distributed random numbers. We set up frequency distributions of a set of random numbers and compare them quantitatively with model distributions by means of the Chi^2 test, the interpretation of which will be explored. (Random-number generators are used in later chapters to simulate measurement inaccuracies and noise, e.g., in Chap. 9 (Evaluation of measurements) and Chap. 10 (Trend lines).) Required spreadsheet functions are: `RAND()`, `FREQUENCY()` as a matrix function, and `CHISQ.TEST()`, as well as the logical functions `AND` and `OR`, together with their counterparts in the Python libraries `numpy.random` and `scipy.stats`.

8.1 Introduction: Statistical Experiments Instead of Theoretical Derivations

Solutions of Exercises 8.2 (Excel), 8.3 (Python), 8.5 (Excel), and 8.8 (Python) can be found at the internet address: go.sn.pub/zHV7Ko.

Russian proverb: *Доверяй, но проверяй!* *Trust but verify!*

German variant: *Vertrauen ist gut, Kontrolle ist besser.* *Trust is good, control is even better.*

Ψ *If in doubt, count!*

Ψ *Mostly, not always.* “Fundamental rule of statistical reasoning”.

Random-number generators

In Python, two libraries have to be imported to make statistical functions available:

`numpy.random (npr)` for the functions `rand` (random numbers between 0 and 1) and `randn` (normally distributed random numbers).

`scipy.stats (sct)` for the functions `norm(0, 1)` (normal distribution) and `chisquare` (Chi² test).

The spreadsheet function `RAND` and the *Python* function `npr.rand` return random numbers x that are equally distributed between 0 and 1 ($0 \leq x < 1$). Other distributions are obtained when these random numbers are redistributed with suitable functions, namely, with the inverse of the distribution function of the desired probability density. For the Gaussian, exponential and Cauchy-Lorentz distributions, these inverse functions are available as `NORM.INV`, `LN` and `TAN` in EXCEL, and `sct.norm(0, 1).ppf`, `np.log`, and `np.tan` in Python.

The distribution function can be approximated with a polyline in a *user-defined function*. This is important for cases in which no predefined function for the inverse of a cumulated distribution function exists. As an example, we calculate the diffraction image of photons that have passed through a double-slit where they are distributed along a line perpendicular to the slits in \cos^2 -shaped maxima.

What is to be learned?

After having worked through this chapter, you should be able to handle the following safely:

- generating a set of random numbers that obey a given distribution model,
- determining the frequencies of occurrence of a data set, and displaying them graphically,
- quantitatively comparing observed frequency distributions with model distributions. To this end, we practice the cautious use of the Chi² test with the correct degree of freedom *dof*.

Statistical experiments instead of theoretical derivations

- ▶ **Alac** Numbers, numbers, numbers. They're quite tiring. Is it worth the effort?
- ▶ **Mag** Yes. This is the basis of the evaluation of experiments.
- ▶ **Alac** Evaluation of Lab-course experiments? I'm happy when I have the annoying protocols behind me. I just want to memorize a few answers from the introduction to the exercises to get through the exam talks scot-free.
- ▶ **Mag** We only practice what is absolutely necessary—but understanding that well is quite important, not only for studying physics, but for all empirical sciences and political statements.
- ▶ **Alac** Understand it well? Do we have to reproduce mathematical proofs?

► **Mag** No. We perform experiments to grasp the essence of certain theorems of statistics.

► **Tim** “Statistical experiments”, that sounds interesting. Nevertheless, probability theory is not my thing. Nobody in my study group understands the Chi^2 test yet.

► **Mag** I can understand that. Appearances are often deceptive, and no statement is a hundred percent certain. Nevertheless, you have to be able to move on this unsteady ground. In this chapter, we will start with gait exercises.

We use two types of statistical test in which a random experiment is repeated many times:

- multiple tests for equal distribution and
- multiple tests for error probability.

In Exercise 8.2, multiple tests for equal distribution will be used to check whether the results of Chi^2 tests are equally distributed, as it should be when the model distribution corresponds to the population from which the samples are taken.

Multiple tests for error probability are used in Chap. 9, “Evaluation of measurements”, and Chap. 10, “Fitting of trend curves to measurement points”, to check whether the experimental confidence intervals of a measurement result match the assumed confidence levels.

Statistical functions in Python and Excel

```
import numpy.random as npr
import scipy.stats as sct.
```

Functions concerning the normal distribution:

```
sct.norm(xm, xd).pdf(x)    NORM.DIST(x;xm;xd; FALSE)
-∞ < x < ∞, probability density function
sct.norm(xm, xd).cdf(x)    NORM.DIST(x;xm;xd; TRUE)
-∞ < x < ∞, cumulative density function (distribution function)
sct.norm(xm, xd).ppf(p)    NORM.INV(p; xm; xd)
0 ≤ p ≤ 1, percent point function, inverse of cdf.
```

Functions generating random numbers:

```
npr.rand(n)                RAND() (matrix function)
random numbers equi-distributed between [0 and 1)
npr.randn(n)              NORM.INV(RAND();0;1) (matrix function)
normally distributed
RANDBETWEEN (no matrix function)
between two given integers,
npr.choice(2, 100, p = [0.2, 0.8])
```

chooses numbers from a list with specified probabilities, here, 100 numbers 0 or 1

```
npr.choice(10, 10, replace = False)
```

without replacement; every number occurs only once.

Nomenclature

<i>pdf</i>	probability density function
<i>cdf</i>	cumulative density function (distribution function)
<i>ppf</i>	percent point function, inverse of <i>cdf</i>
<i>dof</i>	degrees of freedom, by default number of measurements minus 1
<i>ddof</i>	delta degrees of freedom, 0 by default, greater than 0 if parameters of the model distribution other than the sample mean are estimated from the empirical distribution of the frequencies of occurrence.

8.2 Equi-Distributed Random Numbers, Frequencies of Occurrence, Chi² Test

We generate numbers randomly between 0 and 1, determine their frequency of occurrence in intervals of width 0.1, and check, with the *Chi² test*, whether they are equally distributed.

8.2.1 A Spreadsheet Experiment with Random Numbers

In this exercise, we shall perform experiments with random numbers. They will be explained by means of the spreadsheet in Fig. 8.1, where the data structure is clearly laid out, and repeated in Sect. 8.2.3 in Python.

- (1) We generate 1000 random numbers named *Rnd* in column A, equally distributed between 0 and 1, with the spreadsheet function `RAND()`.
- (2) We determine the empirical frequency distribution *FrqObs* of these numbers in intervals with 11 specified boundaries $I_b = 0.0, 0.1, \dots, 0.9, 1.0$ in column C. The 10 intervals between 0 and 1 all have the same width of 0.1. Principally, however, the intervals may be of different widths.
- (3) We perform a Chi^2 test to check how close the observed frequencies *FrqObs* are to *FrqXpt*, the frequencies expected for an equi-distribution. This is done in Range H6:I9. In H9, the Chi^2 test for the equi-distribution. It yields the same value as performed with the function `ChiSq.Test` on *FrqObs* and *FrqXpt*. In H8, this is performed via the value of *ChiSqr* in I6 and the function `CHISQ.DIST.RT`.

	A	B	C	D	E	F	G	H	I	J	K	
2	=RAND()											
3	Rnd	{=FREQUENCY(Rnd;Ib)}										
4	0.38	Ib	FrqObs	FrqXpt	CSq	=(FrqObs-FrqXpt)^2/FrqXpt						
5	0.57	0	0			dof	ChiSq					
6	0.80	0.1	111	100	1.21	9	10.10	=SUM(CSq)			90	
7	0.34	0.2	92	100	0.64						90	
8	0.72	0.3	90	100	1						90	
9	0.35	0.4	85	100	2.25	0.34 =CHISQ.DIST.RT(ChiSq;dof)					90	
10	0.95	0.5	98	100	0.04	0.34 =CHISQ.TEST(FrqObs;FrqXpt)					90	
11	0.44	0.6	117	100	2.89	0.30 =CHISQ.TEST(FrqObs;Frqstep)					110	
12	0.04	0.7	102	100	0.04						110	
13	0.54	0.8	97	100	0.09						110	
14	0.05	0.9	95	100	0.25						110	
15	0.04	1	113	100	1.69						110	
1003	0.69	0										

Fig. 8.1 (S) In A4:A1003, named *Rnd*, 1000 random numbers are generated. In D4:D15, the spreadsheet function FREQUENCY is used to count how many of these random numbers fall into intervals whose limits I_b are specified in C4:C14. Column E shows the frequencies expected for equal distribution. In H8, the Chi² test with *FrqXpt* is performed. In H11, the Chi² test is performed against a step function *FrqStep* in column K, *dof* (in H6) = degrees of freedom

Empirical frequency distribution

In EXCEL, empirical frequencies of occurrence are determined with the matrix function FREQUENCY(DATA_ARRAY; BINS_ARRAY). It outputs 12 frequencies (in D4:D15 in Fig. 8.1 (S)) for the 11 boundaries I_b (BINS_ARRAY), because it also determines the frequencies of the data below the lowest and above the highest boundaries.

The first value in BINS_ARRAY gives the number of occurrences in the examined data set that lie below the first interval boundary, that is, from $-\infty$ on. The last value specifies the number of occurrences that lie above the last interval limit, i.e., up to ∞ .

► Ψ Always one more in Excel! O.k., but of what and than what?¹

The spreadsheet function FREQUENCY is a *matrix function*. Before it is entered, a spreadsheet range must be activated that can capture all output data, and the operation must be concluded with a *magic chord*:-), Ψ (Ctl + Shift) + Enter.

Read the box to get an answer to the important question: “What is a matrix function?”

What is a matrix function in Excel?

¹ For the output of the matrix function *Frequency*, a range has to be activated comprising one cell more than the number of interval boundaries.

FREQUENCY is a matrix function. In terms of spreadsheet calculation, this means that a cell range must be selected that is large enough to contain the amount of the returned data. In the spreadsheet of Fig. 8.1 (S), the area D4:D15 was marked, then the function was entered (see D2), and the process was completed with the “magic chord” for matrix functions: (CTL + SHIFT) + ENTER.

Questions

What is the formula in A4:A1003?²

What set of numbers is sorted into the intervals in Fig. 8.1 (S).³

The Chi² test as a judge

If the random numbers are evenly distributed between 0 and 1, we may expect that, below 0 and above 1, there will be no values, and that in each of the ten intervals of width 0.1, 100 values will occur. This is specified in the variables *FreqXpt*. A Chi² test can be used to check how well the observed distribution, here, *FrqObs*, and the expected one, here, *FrqXpt*, match.

To this end, we calculate the value of χ^2 (Chi², “Chi square”) defined as

$$\chi^2 = \sum_{i=1}^N \frac{(O_i - E_i)^2}{E_i} \quad (8.1)$$

with O_i and E_i being the observed and expected frequencies, respectively, and N the number of intervals considered for the comparison. In our implementations, $O_i = \text{FreqObs} = [\text{D5:D14}]$, excluding the intervals with zero occurrence, and $E_i = \text{FreqExp} = [\text{E5:E14}]$ (only non-zero numbers), with the same size as *FreqObs*. χ^2 is termed *ChiSqr* and calculated in I6 as the sum over the individual terms *CSq* in column F.

The *Chi² test* is listed in the literature under various names, e.g., χ^2 -Test, Chitest. We call it the *Chi² test*, because it is based on a distribution of the quantity $\chi^2 = \text{chi}^2$ (“chi squared”), defined in Eq. 8.1.

We perform such a Chi² test in two ways:

- by introducing the calculated value of *ChiSqr* into the distribution function CHISQ.DIST.RT (in H8), and
- by applying a special function for that test, CHISQ.TEST (in H9).

² = RAND().

³ The 1000 numbers in A4:A1003.

The values of such tests are between 0 and 1, giving the probability that χ^2 of the *current sample* fits better with the model distribution of the *whole population* than *any other sample*. If the Chi² test results in values significantly smaller than 0.01, it is generally concluded that observation and expectation do not match, i.e., the sample is not from the assumed model population. The values in H8 and H9 are 0.34, so that there is no reason to doubt that the random numbers are equally distributed.

The result 0.34 means that, for 34% of other samples of the same population, χ^2 will be greater, i.e., the frequency distribution of these samples fits worse with the theoretical one. A more precise explanation is given in Exercise 8.8.

In H8, the Chi² test is performed by explicitly entering the value of *ChiSqr* and the degrees of freedom *dof* into the inverse function of the integral distribution function of Chi². In EXCEL, we use CHISQ.DIST.RT(CHISQR,DOF). The suffix.RT means that a right-tailed distribution function is to be used. The left-tailed distribution function is addressed with CHISQ.DIST. The meaning of these terms will become clearer in Exercise 8.8.

DOF is the degree of freedom. For our test, *dof* = 9, corresponding to the 10 intervals in which the frequencies are compared minus 1; minus 1 because the frequency in the tenth interval is no longer independent of the other frequencies, but is rather determined by the total number of events.

Questions

FrqObs in Fig. 8.1 (S) is defined for 12 intervals. Why is the degree of freedom for the Chi² test not 11, but rather 9?⁴

The spreadsheet function ChiSq.Test

EXCEL provides the spreadsheet function CHISQ.TEST(ACTUAL_RANGE, EXPECTED_RANGE) for the Chi² test. The test is employed in H9 of Fig. 8.1 (S) and results in the same value $p = 0.34$ as the first test. This function calculates the value of *ChiSqr* internally and assumes a degree of freedom one less than the number of intervals. It must not be applied when parameters of the expected distribution are calculated from the data set.

To repeat: The spreadsheet function CHISQ.TEST may only be used if the parameters of the model distributions are defined in advance, as is the case in our example, and not estimated from the sample.

Does a step distribution also fit?

In addition to comparing our observed frequencies with an equi-distribution, we also compare them with the step distribution *FrqStep* in column K of Fig. 8.1 (S).

⁴ The comparison is only between numbers in the 10 intervals with nonzero value, D5:D14 and E5:E14; *dof* = 10 - 1 = 9.

► **Tim** Looking at Fig. 8.1 (S), we see that the Chi² test provides a larger value ($p = 0.34$ in H9) for the correct distribution than for the incorrect step distribution ($p = 0.30$ in H11).

► **Alac** Well, that’s to be expected. A good reliability check must deliver a lower probability value for a wrong model distribution than for the right one!

► **Mag** Let’s wait and see! The correct distribution does not always fit best.

Repetition of the statistical experiment

We are going to generate the random numbers anew. To this end, we place the cursor into an empty cell and “delete” its content. EXCEL reacts as if the calculation had been changed, generates all 1000 random numbers anew, and recalculates the distribution *FreqObs*. The Chi² test sometimes gives higher values for the comparison with *FrqStep* than for the one with *FrqXpt*.

Within the VBA rep-log procedure in Fig. 8.2b, we copy 1000 results of Chi² tests into columns 13 and 14. Furthermore, we count how often the step function fits better with the observed data than the equi-distribution. This holds true in about 10% of the tests.

The frequency distribution of the results of the two times 1000 Chi² tests is shown in Fig. 8.2a. The values for the comparison with the equi-distribution scatter around 100, i.e., they seem to be equi-distributed, whereas the test against the step distribution delivers mostly values below 0.1. These results are also reflected in the average values of the Chi² tests: 0.5 for the equi-distribution and 0.13 for the step distribution.

Repetitions of statistical experiments are generally not possible in real life, in which there is mostly only one sample and one result of a Chi² test. This is, however, possible in our exercises, because we invent our population ourselves with the

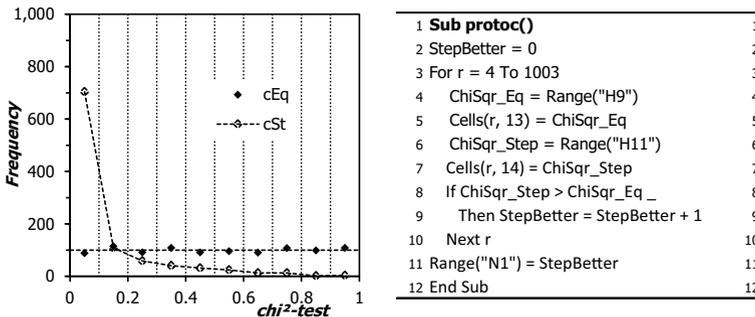


Fig. 8.2 a (left) Frequency distribution of the results of Chi2 tests against an equi-distribution (*cEq*) and a step distribution (*cSt*). b (P, right) The VBA procedure for 1000 repetitions of the two Chi2 tests in Fig. 8.1 (S)

Table 8.1 Interval boundaries and theoretical frequencies of occurrence

```

1 db=0.1
2 Ib=np.arange(0,1+db,db)
3 FrqXpt=np.ones(10)*100 #Equidistribution, expected freqs.
4 FrqStep=np.ones(10) #Step distribution
5 FrqStep[5:]*=110 #Entries 5 to 9
6 FrqStep[:5]*=90 #Entries 0 to 4

```

function `RAND()`, meaning we have a didactical method at hand for demonstrating the peculiarities of statistics.

8.2.2 Data Structure and Nomenclature

Rnd 1000 numbers supposedly randomly distributed between 0 and 1
I_b 11 boundaries for 10 intervals (“bins”) between 0 and 1
FrqObs frequencies of occurrence for *Rnd*
FrqXpc frequencies expected for an equi-distribution
FrqStep frequencies expected for a step distribution
dof Degrees Of Freedom, number of intervals minus 1
ddof Delta Degrees Of Freedom, to be deduced from *dof* if parameters of the distribution are estimated from the sample.

8.2.3 Python

Standard histogram in Python

The interval boundaries *I_b* and the expected frequencies of occurrence for an equi-distribution (*FrqXpc*) and a step function (*FrqStep*) are specified in Table 8.1. We know that there are 10 intervals with numbers greater than zero, so that we specify *FrqXpc* and *FrqStep* with size 10.

Questions

What is the value of *FrqStep[5]* in Table 8.1?⁵

In Table 8.2, 1000 Chi² tests are performed in a for loop on *Rnd*, an array with 1000 random numbers created with the Python function `npr.rand(1000)`. The frequency distribution is obtained with `np.histogram(Data;boundaries)`. This function returns an array of shape (2, 10) with the frequencies reported in row number 0 and the boundaries repeated

⁵ *FrqStep[5]* is 110. *FrqStep[:5]* relates to the first 5 elements, *FrqStep[0]* to *FrqStep[4]*.

Table 8.2 Performing 1000 Chi^2 tests of 1000 random numbers *Rnd* versus equi-distribution *FrqXpc* and versus step function *FrqStep*

```

7 import scipy.stats as sct
8 import numpy.random as npr
9 N=1000
10 cEq=np.zeros(N) #Chi2 values for equi
11 cSt=np.zeros(N) #Chi2 values for step
12 StepBetter=0
13 for i in range(N): #N Chi2 tests
14     Rnd=npr.rand(1000) #Sample
15     FrqObs=np.histogram(Rnd,Ib)[0] #Freq. of occurrence
16     ChiSqr_Eq=sct.chisquare(FrqObs[0],FrqXpt,ddof=0)[1]
17     ChiSqr_Step=sct.chisquare(
18     FrqObs[0],FrqStep,ddof=0)[1]
19     cEq[i]=ChiSqr_Eq
20     cSt[i]=ChiSqr_Step
21     if ChiSqr_Step > ChiSqr_Eq: StepBetter+=1
22 print(„StepBetter „, StepBetter)
StepBetter 57 #Better Chi2 test for step distribution

```

in row number 1. We therefore specify the returned variable with index 0: `FrqObs = np.histogram (Rnd,Ib) [0]` to get the observed number of frequencies.

Contrary to the EXCEL function FREQUENCY, only the 10 frequencies in the intervals between the lowest and highest boundaries are returned, and not the frequencies of occurrence below the lowest and above the highest interval boundaries. The size of `FrqObs[0]` is, therefore, one less than the number of the specified boundaries (`Ib`, reproduced in `FrqObs[1]`). In a diagram, we would plot them over the 10 centers of the intervals with, e.g., `plt.plot(Ic, FrqObs)`.

The Chi^2 test is performed with the function `sct.chisquare (FrqObs,FrqExp,ddof)`. The parameter `ddof` specifies a reduction in *dof*, the degrees of freedom. If no parameter of the theoretical distribution is calculated from the observed sample, `ddof = 0`. This is the case for our model distributions because we estimate only the mean from the sample. So, `dof` is set internally to the number of sorting intervals (“bins”) minus 1.

The tests are performed within a for-loop with $N = 1000$ iterations and their results are stored in arrays `cEq` and `cSt` (lines 20, 21). Plots of `cEq` and `cSt` over the center of the intervals look like Fig. 8.2a. We get qualitatively the same results: The chi^2 values of the “Test of the 1000” versus the equi-distribution (*FrqEq*) are equally distributed between 0 and 1, while those vs. the step distribution (*FrqStep*) increase strongly for Chi^2 test approaching zero.

► *Ψ Always one more in Excel! O.k., but from what and than what?*⁶

⁶ For the output of the matrix function FREQUENCY, a range has to be activated comprising one cell more than the number of interval boundaries. Frequencies of occurrence are returned for values below the lowest interval boundary and for values above the highest boundary.

► *Ψ Always one less in Python! O.k., but from what and than what?*⁷

All-including intervals

The Python function `np.histogram` can cover, similarly to the EXCEL function `FREQUENCY`, the whole range of real numbers from minus infinite to infinite by extending the definition of the interval boundaries with `[-np.inf, ...,]`. This is recommended, because it allows for checking whether the sum of all frequencies is equal to the total number of data points considered.

Questions

Regarding Table 8.2: In what percentage of cases does the Chi² test in our Python program give a higher value for *Step* than for *Eq*?⁸

How does the code have to be changed if 10,000 random numbers are to be considered?⁹

Degree of freedom

What is the degree of freedom for the Chi² tests in lines 16 and 17 of Table 8.2? From Fig. 8.1 (S) and Fig. 8.2a (S), we deduce that the degree of freedom is 9. Is it correct to set `ddof = 0`? We may not be sure what the description means and check this question with the program in Table 8.3, a modified version of parts of Tables 8.1 and 8.2. In lines 8 and 9, we sum up the results of Chi² tests of 10,000 repetitions of the statistical experiment, in `ChiS0` with `ddof = 0` and `ChiS1` with `ddof = 1`. The average values in the lower cell show that, with `ddof = 0`, we are closer to the theoretical average 0.5 of Chi² tests than with `ddof = 1`. So, `ddof = 0` seems to be true, so that we may conclude that the default degree of freedom in `scipy.stats.chisquare` is 9, number of intervals minus 1, `dof = n - 1`, as for the EXCEL function `CHISQ.TEST`.

Consider also the improvement of the code in Table 8.3 over that in Tables 8.1 and 8.2: The number of repetitions can be changed by changing *N* alone.

⁷ Numpy's `np.histogram` outputs a number of frequencies that is one less than the number of boundaries; frequencies of occurrence are returned only for internal intervals.

⁸ Bottom cell of Table 8.2: *StepBetter* = 57, of 1000 runs, makes about 6%. *StepBetter* is updated in line 20 within the for-loop.

⁹ Line 9: `N = 10,000`; line 14: `npr.rand(10,000)`, line 16 `FrqXpc*10`; line 17: `FrqStep*10`. It would be better to use the variable *N* instead of specific numbers.

Table 8.3 Checking the degree of freedom with $N = 10,000$

1	N=10000	
2	FrqXpt=np.ones(10)/10 #Freq. in each of ten bins = FrqXpt*N	
3	ChiS0=0	
4	ChiS1=0	
5	for i in range(N):	
6	Rnd=npr.rand(N)	
7	FrqObs=np.histogram(Rnd, Ib) [0]	
8	ChiS0+=sct.chisquare(FrqObs, FrqXpt*N, ddof=0) [1]	
9	ChiS1+=sct.chisquare(FrqObs, FrqXpt*N, ddof=1) [1]	
10		
11	print("ChiS0 ", np.round(ChiS0/N,3)) #Average value of	
12	print("ChiS1 ", np.round(ChiS1/N,3)) # Chi ² tests	
ChiS0	0.501	#0.50 expected for correct dof
ChiS1	0.429	

8.3 Points Randomly Distributed in a Unit Square

We create coordinates of points randomly distributed in a unit square, use the logical functions AND, OR, NOT to separate points in sub-regions of the unit square, and illustrate the broom rule: Ψ *Chance is blind and checkered.*

8.3.1 Creation and Distribution of the Points

Chance is blind and checkered

Figure 8.3a displays a sample of 2000 points (x, y) , randomly distributed in the unit square. The cross $+$ indicates the center of the distribution obtained with the mean (x_m, y_m) of the coordinates. The vertical bars (“devi bars”) are at $x = x_m \pm x_{sd}$, with x_{sd} being the standard deviation of the sample. The region within the vertical bars is called V . Figure 8.3b displays the same points, with the addition of horizontal bars at $y = y_m \pm y_{sd}$ (y devi bars), and marks points in the inner rectangle, called VH , with open diamonds. The region within the horizontal bars is called H .

Questions

concerning Fig. 8.3:

Do the points give the impression of being equally distributed?

Do the white spots and the point clusters disturb the impression of equal distribution?

What fraction of points in Fig. 8.3a lies within the vertical lines? More or less than 50%?

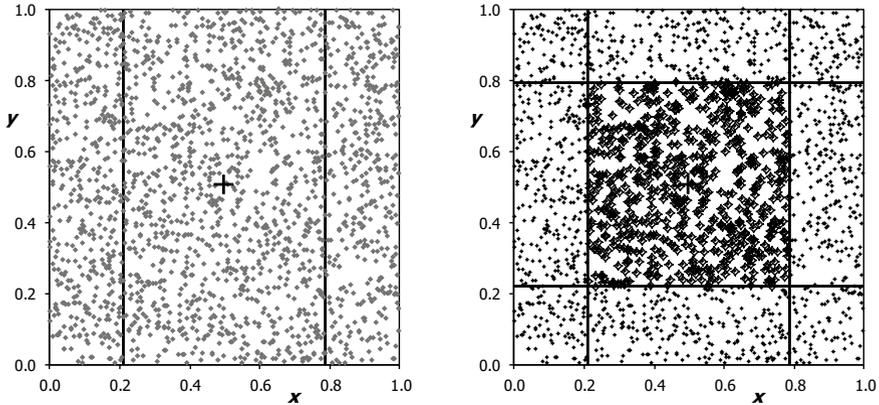


Fig. 8.3 **a** (left) 2000 points (x, y) are randomly distributed in the unit square; the empirically determined center is marked by “+”. The vertical lines mark the standard deviation from the center in the x -direction. **b** (right) As in **a**, but with the addition of horizontal lines marking the standard deviation from the center in the y -direction and the points in the inner rectangle marked with open diamonds

How many points lie within the inner rectangle vH in Fig. 8.3b? Is it more or less than 25%?¹⁰

Calculate a theoretical estimate for the number of points in the rectangle with $x_{Sd} = 1/\sqrt{12}$!¹¹

How do you generate the sets of numbers x and y anew, in Python and in EXCEL?¹²

► **Mag** Take a look at Fig. 8.3, where the distribution of points is intended to be random. You need to place another point, but in such a way that the distribution looks genuinely random. Where would you put it?

► **Alac** In no case would I place it where there are already many points heaped up, but rather into one of the white areas, where there aren’t any points yet.

► **Mag** You must not do that under any circumstances. Chance is blind and does not see where it has already struck before.

¹⁰ Solution in Fig. 8.5 (S), cell P3, $672/2000 = 0.336$.

¹¹ $\frac{2}{\sqrt{12}} \cdot \frac{2}{\sqrt{12}} = \frac{1}{3}$ (see also Eq. 8.6).

¹² EXCEL: Place the cursor in an empty cell and press “Del”, i.e., “delete” the contents of an already empty cell. Then, all formulas in the spreadsheet are recalculated, including the spreadsheet function RAND(). -Python: The numpy function `np.rand(...)` has to be executed in a loop or in a user-defined function.

- **Tim** Now I'm confused. So that I don't make any mistakes, I'll close my eyes before I set my point.
- **Alac** But then you're much more likely to hit a spot outside the predefined rectangle.
- **Tim** So, I'll keep putting down points until you tell me that I've landed a hit within the rectangle.
- **Mag** This is, in fact, a way of randomly placing points within the target rectangle.
- **Tim** But using this blind method, even more clusters can arise.
- **Mag** That's all right. Ψ : *Chance is blind and checkered.*

Statistical characteristics of a sample

The average value or *mean* x_m of a sample is defined, with the number n of elements in the sample, as

$$x_m = \frac{\sum_i x_i}{n} \quad (8.2)$$

The *variance* σ^2 (or var_X in the notation that we will later use) of a sample is calculated with the mean as

$$\sigma^2 = \text{var}_x = \frac{\sum_i (x_i - x_m)^2}{(n - 1)} \quad (8.3)$$

and therefrom the *standard deviation* as the square root:

$$x_{sd} = \sigma = \sqrt{\frac{\sum_i (x_i - x_m)^2}{(n - 1)}} \quad (8.4)$$

The number of elements in the denominator of the variance is reduced by 1, ($n - 1$), because one parameter, the mean, has been estimated from the sample and not been fixed a priori.

In mathematical literature, the standard deviation is usually called σ . We use the term x_{sd} (or y_{sd}) instead, because it has the same physical unit as the sample members x_i (y_i).

In EXCEL, the corresponding functions are AVERAGE for the mean, VAR.S for the variance of a *sample* with ($n-1$) in the denominator, and, correspondingly, STDEV.S for the standard deviation. In PYTHON, the functions are np.mean and

`np.std(x, ddof = 1)`, `ddof = 1` being necessary to get the correct denominator ($n - 1$). When the standard deviation of an *entire population* is to be determined, we have to use `STDEV` in `EXCEL` and `ddof = 0` in the `Python` function.

According to statistical theory, for a uniform distribution between 0 and 1, we get

$$\text{Mean } x_m = \frac{1}{2} \tag{8.5}$$

$$\text{Standard deviation } x_{Sd} = \frac{1}{\sqrt{12}} \approx 0.289 \tag{8.6}$$

8.3.2 Data Structure and Nomenclature

x, y	random coordinates of points in the unit square
x_m, y_m	mean of x and y
x_{Sd}, y_{Sd}	standard deviation of x and y
in_X	true if $x_m - x_{Sd} \leq x < x_m + x_{Sd}$
in_Y	true if $y_m - y_{Sd} \leq y < y_m + y_{Sd}$
in_{XY}	true if $in_X == \text{True}$ and $in_Y == \text{True}$
V	inner vertical stripe of points with $in_X == \text{True}$
H	inner horizontal stripe of points with $in_Y == \text{True}$
VH	inner rectangle, points with $in_{XY} == \text{True}$
in_V, in_H, in_{VH}	number of points in V, H, VH
x_l, y_l	coordinates of points in VH
L, R, B, T	left, right, bottom, top stripe defined by $x \leq x_m - x_{Sd}, x \geq x_m + x_{Sd}, \dots$

8.3.3 Excel

With the spreadsheet function `RAND()`, 2000 random numbers are generated each in columns A and B of Fig. 8.4 (s), designated x and y , and displayed as 2000 points (x, y) in the plane (Fig. 8.3a).

The mean values x_m and y_m of 2000 between 0 and 1 equally distributed random numbers are (not surprisingly) about 0.5. The standard deviations are found to be $x_{Sd} = 0.29$ and $y_{Sd} = 0.29$, corresponding within 2% to the theoretical value of 0.289 (Eq. 8.6). So, there is a fraction $2 \times 0.29 = 0.58$ to be expected within the vertical bars or within the horizontal bars, and a fraction $0.58^2 \approx 0.33$ ($\frac{2}{\sqrt{12}} \cdot \frac{2}{\sqrt{12}} = \frac{1}{3}$) within the inner rectangle of Fig. 8.3b.

	A	B	C	D	E	F	G	H	I	J	K	L
1	=RAND() =RAND()								=AND(xm-xSd<=x;x<xm+xSd) =AND(ym-ySd<=y;y<ym+ySd) =AND(G4;H4) =IF(I4;x;NA()) =IF(I4;y;NA()) =COUNT(YI)			
2	x	y	xm	0.51	=AVERAGE(x)	inX	inY	inXY	xI	yI		
3	0.24	0.57	xSd	0.29	=STDEV.S(x)	TRUE	TRUE	TRUE	0.24	0.57		672
4	0.38	0.55	ym	0.49	=AVERAGE(y)	TRUE	TRUE	TRUE	0.38	0.55		0
5	0.89	0.22	ySd	0.29	=STDEV.S(y)	FALSE	TRUE	FALSE	#N/A	#N/A		=SUM(inY)
2002	0.12	0.02				FALSE	FALSE	FALSE	#N/A	#N/A		

Fig. 8.4 (S) In columns A and B, each 2000 random numbers between 0 and 1 are generated, and in C3:F3, their mean value (x_m , x_m) and standard deviations x_{Sd} and y_{Sd} are determined. The columns G to I check whether the points are within certain boundaries. L4 contains the formula SUM(INY). “#N/A” means “Not available”. The name of this worksheet is “calc” and is addressed as such in VBA procedures

	M	N	O	P	Q	R	S	T	U
1	=COUNTIF(inX;TRUE) =COUNTIF(inY;TRUE) =COUNTIF(inXY;TRUE) =inVH/2000 =COUNTIF(x;"<="&xm-xSd) =COUNTIF(x;">"&xm+xSd) =COUNTIF(y;"<="&(ym-ySd)) =COUNTIF(y;">"&(ym+ySd))								
2	inV	inH	inVH	fvH	nL	nR	nB	nT	
3	1157	1179	672	0.336	409	434	408	413	

Fig. 8.5 (S) COUNTIF is applied, counting how often the data x and y in Fig. 8.4 (S) are less away from the center than the standard deviation, in M:O with the logical data in x_{in} , y_{in} , and xy_{in} . In Q:T, the data x and y themselves are taken for the logical query if the data are further away. We get $inV + nL + nR = 2000$

Check with And

In column G, for each individual x -coordinate, we check whether it deviates less than the standard deviation downwards or upwards from the mean value x_m . In column H, the same is done for the y -component. For these checks, the logical function AND(LOGICAL1, [LOGICAL2], ...) is used. The corresponding formulas are.

$$inX = \text{AND}(x_m - x_{Sd} \leq x; x < x_m + x_{Sd})$$

$$inY = \text{AND}(y_m - y_{Sd} \leq y; y < y_m + y_{Sd})$$

In column I, we mark the points within the *devi* limits of both x_m and y_m . We cannot use the statement AND(x_{in} ; y_{in}), because it returns TRUE if *all* elements of the column vectors x_{in} and y_{in} are true. So, we have to write, e.g., I4 = AND(G4;H4), and so on, to check *individually* whether the contents of both cells are true.

Logical functions

In columns J and K, we extract the coordinates of the points within the inner rectangle with a statement like K4 = [=If(I4; x; NA())], returning the value of x if I4 = True and #N/A else. The column ranges in K and J can be entered as data series into a figure with the cells with #N/A being ignored.

A check $COUNT(y_I)$ (in L3) yields 672, although the queried range y_I comprises 2000 entries, because only the cells with numbers, and not those with #N/A, are counted. The function $SUM(inY)$ returns 0 because only numbers are summed up and logical values are ignored.

Logical Functions

We have used two logical functions so far:

`IF(LOGICAL_TEST, [VALUE_IF_TRUE], [VALUE_IF_FALSE]);`

in `[VALUE_IF_TRUE]` and `[VALUE_IF_FALSE]` may appear numbers, or functions whose result is then processed.

`AND(LOGICAL1, [LOGICAL2], ...)`

expects logical values as input and outputs TRUE or FALSE.

Please refer to EXCEL help to inform yourself about the other functions in the category LOGIC: OR; FALSE; TRUE; NOT; IFERROR.

Count the Trues

In Fig. 8.5 (S), the logical calculations of Fig. 8.4 (S) are continued. The spreadsheet function `COUNTIF(RANGE, CRITERIA)` counts the non-empty cells of a range whose contents match the search criteria. In M3:O3, the ranges in_X , in_Y , and in_{XY} are queried as to how often their cells contain TRUE. In P3, the relative frequency f_{VH} of points within the inner rectangle in Fig. 8.3b is reported ($f_{VH} = 0.336$).

In the subsequent columns Q to T, the number of points in the lower n_L , right n_R , bottom n_B , and top n_T stripe is calculated. Here, logical expressions formulated as a character string are used as criteria. In the formula $S3 = [=COUNTIF(y\&" < = "&y_m - y_d)]$, the criterion is `" < = "&y_m - y_d`. The expression is of the form Ψ "Text"&Variable, composed of the comparison symbol in quotation marks "<" and the arithmetic expression $y_m - y_d$.

In Fig. 8.6 (S), the results of Fig. 8.5 (S) are evaluated to estimate the relative amount of points within the inner rectangle with a probabilistic calculation. In columns V and W, the number of points outside the x and y devi boundaries is obtained through addition; in X and Y, it is converted into the numbers inside the devi bars (the same as in Fig. 8.5 (S)) from which the relative amount is obtained in Z and AA. These relative amounts are interpreted as probabilities.

Fig. 8.6 (S) Continuation of Fig. 8.5 (S)

	V	W	X	Y	Z	AA	AB
1	$=n_L+n_R$	$=n_B+n_T$	$=2000-U^3$	$=2000-V^3$	$=W^3/2000$	$=X^3/2000$	$=pV*pH$
2			$=inV$	$=inH$	pV	pH	pVH
3	843	821	1157	1179	0.58	0.59	0.341

Product of probabilities

As the x and y coordinates are independent of each other, the probability of finding a point in the inner rectangle may be calculated as the product of the two probabilities mentioned above:

$$p_{VH} = p_V \cdot p_H$$

We get $p_{VH} = 0.341$ (AB3). This is to be compared with the relative frequency of occurrence $f_{VH} = 0.336$ in Fig. 8.5 (S) and with the theoretical value of $1/3 = 0.333$ (from Eq. 8.6).

Keep in mind the differences:

- The theoretical probability is *logically derived* from the theoretical values of the mean and the standard deviation of an equi-distribution independent of the current experiment with 2000 points.
- The quantity p_{VH} is *estimated* from the relative occurrence of points in V and H in our current experiment.
- The relative frequency of occurrence in VH is *empirically determined* in our current experiment.

Due to these methodological differences, the three values are not necessarily the same; but, as they evaluate the same set of points, their values should be close together.

8.3.4 Python

In Python, we have to import the `numpy.random` library as `npr` (see Table 8.4) for our exercise. The generation of the 2000 points is straightforward with `npr.random(2000)`.

The logical queries as to whether the points are in the areas V or H are performed in three different ways, all resulting in a Boolean array with length 2000:

- in line 6, “`inX = np.logical_and((xm-xSd) < x, x < (xm + xSd))`”, identical to the spreadsheet formulas in Fig. 8.4 (S), with the result being shown in the bottom cell of Table 8.4,
- in line 7, “`inX2 =` ” as list comprehension with identical results to those in line 6, as checked with the instruction “`inX == inX2`”, resulting in a Boolean array of `Trues`,
- in line 8, “`inY =` ” as a different type of list comprehension.

The query as to whether a point is in the inner rectangle is done by means of the list comprehension in line 9.

Table 8.4 Python: 2000 points (x, y) randomly distributed in a unit square, the points (x, y) are depicted in Fig. 8.3a, as are the points (x_I, y_I) in Fig. 8.3b

```

1 import numpy.random as npr
2 x=npr.random(2000)
3 xm,xSd = np.mean(x),np.std(x)
4 y=npr.random(2000)
5 ym,ySd = np.mean(y),np.std(y)
6 inX=np.logical_and(xm-xSd<x,x<xm+xSd) #Within std error?
7 inX2=[(xm-xSd<x) & (x<xm+xSd)]
8 inY=[(ym-ySd < yi < ym+ySd) for yi in y]
9 inXY=[inX & inY]
   #Arrays of coordinates of points within error range:
10 xI=np.extract(inXY,x)
11 yI=np.extract(inXY,y)

```

```

inX  [ True  True  True ... False False  True]
len(inX)      2000
inX==inX2
      [[ True  True  True ...  True  True  True]]
len(yI)      649

```

The coordinates of the points in the inner rectangle are obtained, in lines 10 and 11, with the function `np.extract(condition, array)`. The condition here is that `inXY == True`. The length of x_I , i.e., the number of points in the inner rectangle, is 649, to be compared to $672 = \text{COUNT}(Y_I)$ in the EXCEL sheet of Fig. 8.4 (S). The theoretical expectation for a binomial question (point in VH or not in VH) is $x_m = n \cdot p = 667$ and $x_{sd} = \sqrt{npq} = 21$, so that the above values are within the standard error range. The meaning of these statements will be discussed in Chap. 9.

Logical operations on lists and arrays in Python

The operators `&` (and), `|` (or), and `~` (not) operate item-wise on lists of Booleans and create new lists of Booleans. In Table 8.4, however, the `numpy` functions `np.logical_and`, `np.logical_or`, and `np.logical_not` are applied; they do the same, except that they create arrays. The difference becomes apparent when we try to combine two of them, e.g., with `&`. This is not possible for lists; in trying to do so, we get the error message "Unsupported operand type(s) for `&`: 'list' and 'list'". But it works well with arrays, e.g.,

```
inXY=[inX & inY]
```

yielding a Boolean array with "True" if the point lies within the inner rectangle in Fig. 8.3b. With the instruction.

```
xI = np.extract(inXY,x)
```

Table 8.5 Size of the Boolean arrays denoting the various partial areas in Fig. 8.3b

12	<code>inV=sum(inX)</code>	<code>inV</code>	1124
13	<code>inH=sum(inY)</code>	<code>inH</code>	1153
14	<code>inVH=sum(inXY)</code>	<code>inVH</code>	649
		<code>fVH</code>	0.32
15	<code>inL=sum(xi<xm-xSd for xi in x)</code>	<code>inL</code>	437
16	<code>inR=sum(xm+xSd<xi for xi in x)</code>	<code>inR</code>	439
17	<code>inB=sum(yi<ym-ySd for yi in y)</code>	<code>inB</code>	433
18	<code>inT=sum(ym+ySd<yi for yi in y)</code>	<code>inT</code>	414

a new array is created containing the x coordinates of the points in the inner rectangle.

The points in the inner square are plotted with an open diamond (see Fig. 8.3b) with the instruction

```
plt.plot(xI,yI,'kd', fillstyle='none').
```

In Table 8.5, we have calculated the numbers of points within the various partial areas V , H , VH , L , R , B , T of the unit square, defined in Sect. 8.3.2.

The values TRUE and FALSE in a Boolean array are coded as 1 and 0 in a binary list. So, the number of TRUES is equal to the sum over the list. This is not possible in Excel; = SUM(INY) in L4 of Fig. 8.4 (S) yields 0.

Questions

How are in_V and in_H related to in_L , in_R , in_B , in_T ?¹³

How do you estimate the probability of finding points in VH from in_V and in_H ? Compare the constructions in lines 6 to 9 in Table 8.4!¹⁴

8.3.5 Why Calculate Twice?

► **Alac** In Sect. 8.3.3, we calculate the probability of finding a point in the inner rectangle one time too many, once as f_{VH} in O3 of Fig. 8.5 (S) and a second time as $p_V \cdot p_H$ in Fig. 8.6 (S). We could have saved work.

► **Tim** There is a difference: f_{VH} estimates the probability from the relative frequency in VH , whereas p_{VH} relies on the product rule of probability.

► **Mag** Tim is right. Anyhow, it is always good to follow two calculation paths that are expected to deliver the same result. In this way, we can check whether

¹³ $in_V = 2000 - in_L - in_R$; $in_H = 2000 - in_T - in_B$.

¹⁴ $p_{VH} = p_V \cdot p_H = in_V/2000 \cdot in_H/2000 = 0.324$.

our thoughts have been logical and whether we have correctly transferred our thoughts into the calculation. This is especially advisable when using functions and formulas for the first time.

► **Practical advice** Before you seriously use functions for the first time, check their operation with examples for which you can do the arithmetic mentally!

Product rule of probability

As can be seen in Fig. 8.5 (S) (and Table 8.5), only about $p_H = 59\%$ ($inH/2000 = 0.59$) of the points lie within the two horizontal lines in Fig. 8.3b. Within the vertical lines, the proportion is also 0.59 (0.56). Within the rectangle (p_{VH}), it is less. According to the rules of probability, it should be the product of the probabilities within the vertical and horizontal lines, $p_{VH} = p_V \cdot p_H$, provided the two events are independent of each other. This is indeed the case, because the x - and y -coordinates have been created independent of each other. We estimate the probabilities with the point frequencies $(0.59 \times 0.59) = 0.348$. If there is a difference between this probability and that obtained with the number of points in the inner rectangle, it is due to the fact that the probabilities are not exact, but simply estimated from the number of points in the considered areas.

8.4 Set Operations in Numpy

Python provides a data format for sets, and the operations that will be performed on them. With such sets, we illustrate basic rules for the probabilities of unions and intersections, as well as Bayes' rule on conditional probabilities and apply logical queries.

8.4.1 Sets

In the program underlying Fig. 8.7, 400 points are created in the unit square, with their x - and y -coordinates being two arrays of independent, equally distributed random numbers. In Fig. 8.7a, only the points in three subsets are displayed: the top stripe T with $y > 0.8$ (marked with +), the right stripe R for $x > 0.6$ (marked with x), and the points in the right upper rectangle RT (additionally marked with open squares □).

The points in the ranges R and T are defined in Python as sets with their coordinates obtained through logical queries $x > 0.6$ and $y > 0.8$ (see Table 8.7). The number of points in the sets is obtained with the function `len`:

$$\text{len}(R) \rightarrow 158, \quad \text{len}(T) \rightarrow 85.$$

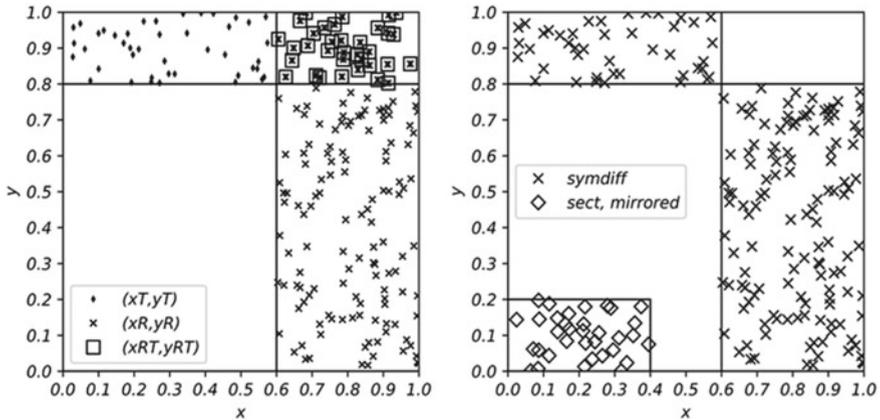


Fig. 8.7 Points in T , R , and RT . **a** (left) Plotted with their coordinates, e.g., `plt.plot(xT, yT, ...)`; **b** (right) Plotted as two sets, `symdiff` and `sect = RT` mirrored at $(0.5, 0.5)$ (see Table 8.8)

The probability that a new point will be in one of the sets may be estimated as:

$$p(R) = 158/400 = 0.395, \quad p(T) = 85/400 = 0.213$$

Now, we perform set operations on R and T (Table 8.8):

- *Union*, defined in Python as `R|T`; comprising the points in R **or** T
- *Section*, defined in Python as `R&T`, comprising the points in R **and** T , marked above in Fig. 8.7a with open squares.

For the situation in Fig. 8.7, we get:

$$\begin{aligned} \text{len}(R|T) &= 211, \quad p(R|T) = 211/400 = 0.528 \\ \text{len}(R\&T) &= 32, \quad p(R\&T) = 32/400 = 0.080 \end{aligned}$$

Applying the addition rule for the union yields:

$$p(R|T) = p(R) + p(T) - p(R\&T) \quad (8.7)$$

which holds exactly true for our special case:

$$211/400 == 158/400 + 85/400 - 32/400$$

The multiplication rule for independent sets reads:

$$p(R\&T) = p(R) \cdot p(T) \quad (8.8)$$

The precondition for the multiplication rule is that the two events are independent of each other. R and T are indeed independent sets, because x and y are generated independently, and the membership in R and T depends exclusively on x or y . As an illustrative counter-example, consider RT where x and y are not independent of each other.

Checking the multiplication rule for our special case gives us

$$\begin{aligned} p(R\&T) &= 32/400 = 0.080 \\ p(R) \cdot p(T) &= 158/400 \times 85/400 = 0.084 \end{aligned}$$

There is no exact coincidence between these probabilities, because they are based on an estimate using a sample size of 400. Increasing the number of points lets the two probabilities come closer together.

Other set operations are:

- *Difference*, defined as $R-T$, points in R but not in T :
- $\text{len}(R-T) = \text{len}(R) - \text{len}(R\&T)$
- *Symmetric difference*, defined in Python as $R\hat{T}$, points either in R or in T , indicated in Fig. 8.7b with x :
- $\text{len}(R\hat{T}) = \text{len}(R|T) + \text{len}(T|R) - \text{len}(R\&T)$.

For our specific case, we get: $\text{len}(R-T) = 126$, $\text{len}(R\hat{T}) = 179$.

Conditional probabilities, Bayes' rule

What is the probability $p(T|R)$ of finding a point in set T when we know that it is in set R ? Graphically, this is the area of RT divided by the area of T , formulated with probabilities as

$$p(T|R) = \frac{p(R\&T)}{p(R)} \tag{8.9}$$

For our specific case, we get

$$\begin{aligned} p(T|R) &= (32/400)/(158/400) = 0.202 \\ p(R|T) &= (32/400)/(85/400) = 0.376 \end{aligned}$$

The two conditional probabilities are interconnected by Bayes' rule:

$$p(R|T) = p(T|R) * \frac{p(R)}{p(T)} \tag{8.10}$$

Questions

What is the theoretical probability of finding a point in the upper-right rectangle RT of Fig. 8.7?¹⁵

Check the rules for difference and symmetric difference for the numbers of our specific case!¹⁶

Check Bayes' rule for the numbers of our specific case!¹⁷

8.4.2 Data Structure and Nomenclature

x, y	coordinates, random between 0 and 1
R	set of points in the right stripe
x_{Rl}	left boundary of R
in_R	Boolean array, TRUE if the point is in the right stripe
x_R, y_R	coordinates of points in R
R_{tup}	array of tuples (x_R, y_R)
T	set of points in the top stripe
y_{Tp}	bottom boundary of T
RT	section of R and T
x_T, y_T	coordinates of points in T .

8.4.3 Python

In Table 8.6, random coordinates (x, y) of $N = 400$ points are generated from which those points are selected that lie in a vertical stripe R right of x_{Rl} . To achieve this, first, a logical array in_R is determined containing `True` when the condition $x_{Rl} < x$ is fulfilled. Then, the coordinates x_R and y_R are extracted from the `True` positions.

In Table 8.7, two sets R and T are built based on the coordinate lists x_R, y_R and x_T, y_T . We first zip the coordinates into tuples (result in Table 8.10) and then define R and T as sets with these tuples as elements. In our specific case, an array with 158 (out of 400) entries results. The same is done for a horizontal stripe T above y_{Tp} . In the following, the set operations *union*, *section*, *difference*, and *symmetric difference* are performed. The size of the resulting sets is reported in the right cell of Table 8.7.

¹⁵ The probability of finding a point in RT is its area $0.2 \times 0.4 = 0.08$ in the unit square.

¹⁶ $\text{len}(R-T) = \text{len}(R) - \text{len}(R\&T)$; $126 = 158 - 32$.

$\text{len}(R\hat{T}) = \text{len}(R|T) - \text{len}(R\&T)$; $179 = 211 - 32$.

¹⁷ $32/85 = 32/158 * 158 / 85$.

Table 8.6 Coordinates of the three sections *R*, *T*, and *RT* in Fig. 8.7

```

1  N=400
2  x= npr.random(N)
3  y= npr.random(N)
4
5  xRt = 0.6                #Right border
6  inR=[xRt<x]             #Right stripe
7  xR=np.extract(inR,x)
8  yR=np.extract(inR,y)
9
10 yTp = 0.8                #Top border
11 inT=[yTp<y]             #Top stripe
12 xT=np.extract(inT,x)
13 yT=np.extract(inT,y)
14
15 inRT=np.logical_and(inR,inT) #Top-right rectangle
16 xRT=np.extract(inRT,x)
17 yRT=np.extract(inRT,y)

```

Table 8.7 Defining the sets *R* and *T* and combined sets

18 Rtup=zip(xR,yR) #Right str.		
19 R=set(Rtup)	len(R)	158
20		
21 Ttup=zip(xT,yT) #Top stripe		
22 T=set(Ttup)	len(T)	85
23		
24 union = R T	len(R T)	211
25 sect = R&T	len(R&T)	32
26 differ = R-T	len(R-T)	126
27 symdiff = R^T	len(R^T)	179
28 un_m_sc = union-sect	len(un_m_sc)	179

Questions

In Table 8.6, two variables with nearly identical names occur, *xRt* (a scalar) and *xRT* (an array). Why is there no name conflict?¹⁸

Table 8.8 reports the instructions for plotting Fig. 8.7b. Bayes’ rule is numerically checked in Table 8.9.

¹⁸ Python is case-sensitive; *xRt* and *xRT* are two different names. Nevertheless, this is bad naming.

Table 8.8 Plotting the sets in Fig. 8.7b

```

1 FigStd('x',0,1,0.1,'y',0,1,0.1)
2 x4,y4=zip(*syndiff)
   #For effect of zip(* see:
   https://stackoverflow.com/questions/29139350/difference-
   between-ziplist-and-ziplist
3 plt.plot(x4,y4,'kx',label="syndiff")
4 x5,y5=zip(*sect)
5 x6,y6=np.array(x5),np.array(y5)
6 plt.plot(1-x6,1-y6,'kD',fillstyle="none",
           label="sect, mirrored")

```

Table 8.9 Bayes' rule

1	$T_given_R=len(sect)/len(R)$	0.203
2	$R_given_T=len(sect)/len(T)$	0.376
3	$Bayes=T_given_R*len(R)/len(T)$	0.376

Table 8.10 Data structure of R_{tup} and R defined in Table 8.7

```

xR [ 0.86 0.78 0.86 ... 0.99 0.73 0.97]
yR [ 0.79 0.24 0.14 ... 0.09 0.43 0.48]
*zip(xR,yR) (0.86, 0.79) (0.78, 0.24) ...
Rtup <zip object at 0x000001E3C57F7B48>
R {(0.85, 0.17), (0.9, 0.09), ... }

```

Some Python constructs

The function `np.extract (Boolean_array, Value_array)`, applied in Table 8.6, extracts those values of the `Value_array` for which the corresponding position in a `Boolean_array` contains `True`.

Table 8.10 elucidates the data structure of a `zip` object and a set. The `zip()` function pairs together the items of iterators passed as arguments. An iterator is an object that contains a countable number of values; in our case, we pair the two arrays x_R and y_R . The set R contains the same tuples as R_{tup} , but in a different order because the order does not play any role in sets.

8.5 Normally Distributed Random Numbers

The functions `NORM.INV(RAND();0;1)` (EXCEL), as well as `sct.norm.ppf(npr.random(...))` and `npr.randomn(...)`

(Python), generate numbers that are standard-normally distributed. We check this statement by comparing a frequency distribution of a set of such numbers with the theoretical distribution using the Chi² test. The expected frequency of occurrence in an interval is calculated in two ways: *Exactly*, with the values of the cumulative distribution function (*cdf*) at the interval boundaries, and *approximately*, with the probability density (*pdf*) in the middle of the interval. The inaccuracy of the approximation will only become visible with more than 10,000 random numbers.

8.5.1 Normal Distribution, Probability Density and Distribution Function

This section presents the mathematical background of the formulas applied later in this exercise.

Normal distribution

The following nominations and specifications hold:

x_m, x_{sd}	mean value and standard deviation
$-\infty < x < \infty$	argument range
$0 < p < 1$	probability, value range

EXCEL provides two spreadsheet functions, one with a Boolean parameter CUMULATIVE? for calculations with normal distributions, corresponding to three Python functions available from the `scipy` library:

- probability density function *pdf*, $x \rightarrow p$

NORM.DIST (X, X_M, X_{SD}, CUM), *cum = False* or 0
`sct.norm(x_m, x_sd).pdf(x)`

- cumulated density function *cdf* (distribution function), $x \rightarrow p$

NORM.DIST (X, X_M, X_{SD}, CUM), *cum = True* or 1,
`sct.norm(x_m, x_sd).cdf(x)`

- the inverse of the distribution function, $p \rightarrow x$

NORM.INV(P, X_M, X_{SD}),
`sct.norm(x_m, x_sd).ppf(p)`

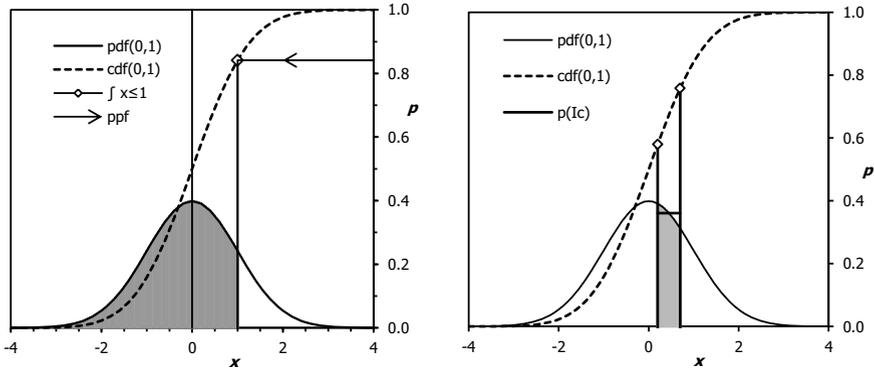


Fig. 8.8 **a** (left) Solid line (bell curve): $pdf(0, 1)$, probability density of the standard normal distribution (i.e., $x_m = 0$ and $x_d = 1$); dashed line: $cdf(0, 1)$, distribution function of the normal distribution; the content of the grey area under the probability density is indicated by the diamond at the end of the vertical bar ending in a diamond on the distribution function cdf . **b** (right) The grey area below the probability density pdf is equal to the difference of the two y -values of the diamonds on the distribution function cdf . The horizontal bar indicates the value of the probability density in the middle of the considered interval

The extension ppf stands for “percent point function”. However, most people in statistics use “quantile function”.

In mathematical textbooks, the normal distribution is often referred to as $N(\mu, \sigma^2)$, where μ is the mean value and σ^2 is the variance. However, since it is the standard deviation that has to be entered as a parameter into the spreadsheet and Python functions, we shall use the abbreviations $pdf(x_m, x_{sd})$ in the text and in legends for the probability density of the standard normal distribution that is defined by the equation

$$pdf(x_m, x_{sd}) = \frac{1}{\sqrt{2\pi} \cdot x_{sd}} \cdot \exp\left(-\frac{1}{2} \left(\frac{x - x_m}{x_{sd}}\right)^2\right) \quad (8.11)$$

The bell-shaped $pdf(0, 1)$ is shown in Fig. 8.8, together with the corresponding S-shaped $cdf(0, 1)$ (dashed line).

Questions

What is the physical unit of the argument of exp in Eq. 8.11?¹⁹

What is the physical unit of pdf and $pdf \cdot dx$?²⁰

¹⁹ The argument in exp is dimensionless, because the nominator and denominator have the same unit.

²⁰ $[pdf] = 1/[x_d]$, a density; $[pdf \cdot dx] = \text{unit-less}$, a probability.

Do the following two spellings yield different results in EXCEL: $Y = \text{EXP}(-2^2)$ and $Y = \text{EXP}(-(2^2))$?

Same question for `np.exp (-2**2)` and `np.exp (-(2**2))` in Python.²¹

Distribution function or cdf

In mathematical literature, the integral of a probability density is called the distribution function; it is monotonously increasing and spans the value range (the range of the output values) from 0 to 1. In Python, it is called *cdf*, “cumulated (probability) density function”. We shall adopt this notation together with *pdf* and *ppf* in this text.

The distribution function of the normal distribution cannot be represented in a closed form, but this is not a disadvantage for us, since EXCEL offers a spreadsheet function `NORM.DIST(x; xm; xd; TRUE)` and Python offers `scipy.stats.norm(xM, xSD).cdf(x)`, which are good approximations.

In Fig. 8.8a, the probability density of the normal distribution is shown up as a bell curve and the distribution function as a monotonously increasing S-shaped curve. The open diamond marks the distribution function at $x = 1$, corresponding to the value of the integral over the probability density from $-\infty$ to 1, which is represented as a grey-filled area under the bell curve, representing the probability of finding a value of the random number below $x = 1$.

The function *ppf*(p) is the inverse of the distribution function *cdf*(x), giving the value of x for a specified p , as demonstrated by the kinked line in Fig. 8.8a: A value p_0 on the vertical axis between 0 and 1 is assigned a value x_0 on the horizontal axis (see arrow). It is the probability of finding a value $x \leq x_0$, corresponding to the grey area under the probability density.

Probability that a random number falls into an interval

What is the probability $p(x_1, x_2)$ that a normally distributed random number lies in the interval $[x_1, x_2]$? To calculate this, we have to integrate over the probability density *pdf*(x):

$$p(x_1, x_2) = \int_{x_1}^{x_2} pdf(x) dx = cdf(x_2) - cdf(x_1) \quad (8.12)$$

The integral in Eq. 8.12 is the difference of the distribution function at the two interval limits. To give an example: in Fig. 8.8b, an interval from 0.2 to 0.7 has been selected. The probability of finding a normally distributed random number in this interval corresponds to the grey area under the probability density *pdf*, which, in turn, corresponds to the difference of the values of the distribution function *cdf* at the two positions marked with open diamonds.

²¹ Yes, there is a difference, but only in EXCEL. In a spreadsheet, the argument of the Gaussian function must be spelled as $-(x^2)$, because negation has operator precedence over potentiation, quite surprisingly for mathematically-educated readers and an annoying error source, so: $-2^2 = 4$; $-(2^2) = -4$.

The integral over an interval can be *approximated* by the product of the probability density in the center x_C of the interval and the width Δx of the interval:

$$p(x_1, x_2) \approx pdf\left(\frac{x_2 - x_1}{2}\right) \cdot (x_2 - x_1) = pdf(x_C) \cdot \Delta x \tag{8.13}$$

In Fig. 8.8b, this value is indicated by the horizontal line between the interval limits. The more linear the probability density between the selected interval limits and the narrower the interval, the better this approximation. The frequency of occurrence in that interval for a sample of size N is $pdf(x_C) \cdot \Delta x \cdot N$.

8.5.2 Random-Number Generator and Frequencies of Occurrence

The function `npr.randn(N)` of the library `numpy.random` returns an array of N numbers randomly distributed according to the standard normal distribution with $x_m = 0$ and $x_{sd} = 1$. In EXCEL, there is no simple function for doing this, and thus a nested function, `NORM.INV(RAND();0;1)`, has to be used. The literal equivalent in Python is `scf.norm.ppf(npr.random(N))`.

Figure 8.9 displays the two frequency distributions of 10,000 random numbers created with the two Python functions `randn` and `ppf(random)`, together with the theoretically expected distribution. We see that, indeed, the two functions act alike, and the numbers provided seem to be normally distributed.

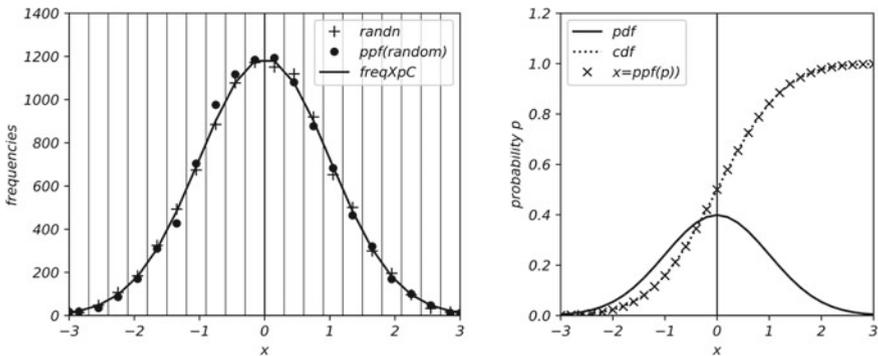


Fig. 8.9 **a** (left) Frequency distribution of the numbers generated with the two generators of normally distributed random numbers in Table 8.12 (`randn` and `ppf(random)`); the vertical bars represent the interval boundaries; the polyline connects the frequencies $freq_{XpC}$ expected for a normal distribution (obtained with $cdf(0, 1)$). **b** (right) Data for $pdf(x)$, $cdf(x)$, and $ppf(p)$ of the standard normal distribution obtained from Table 8.11.

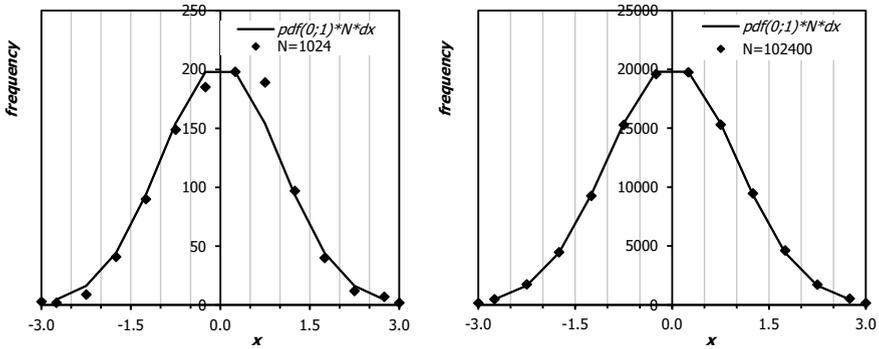


Fig. 8.10 **a** (left) Frequency distribution for 1024 random numbers. **b** (right) Frequency distribution for 100 times more random numbers than in **a**

8.5.3 Where Do Observed and Theoretical Frequencies Fit Better Together?

Figure 8.10a displays experimentally determined frequencies of occurrence for 1024 random numbers generated in EXCEL with `NORM.INV(RAND();0;1)` as data points over the interval centers x_c . The numbers of events outside of the minimum and maximum interval boundaries are shown in the picture on these interval boundaries (here, -3 and 3). Figure 8.10b shows a similar picture for a hundred times more random numbers.

The polylines in Figs. 8.10a and Fig. 8.10b represent the frequencies of occurrence in the interval $[x_1, x_2]$:

$$freq(x_1 \text{ to } x_2) = ppf(x_c) \cdot N \cdot \Delta x \tag{8.14}$$

where N is the total number of data and x_c and Δx are the center and width, respectively, of the sorting intervals, also called the “bins”.

Questions

What additional information would significantly increase the information content of the legends Fig. 8.10a, b? Compare Fig. 8.16b!²²

► **Mag** Where do observed and theoretical frequencies fit better together, in the left or right picture of Fig. 8.10?

► **Alac** Quite clearly, in the right picture. The experimental and theoretical points are much closer together than in the left picture.

²² Indicating the result of a Chi² test would considerably increase the information in the figure.

- **Tim** This surely is a trick question.
- **Mag** Indeed. Many unbiased observers believe, like Alac, that, in Fig. 8.10b, model distribution and experimental frequencies coincide better than in Fig. 8.10a.
- **Alac** Sure.
- **Mag** Well, it's an optical illusion. For Fig. 8.10a, the Chi^2 test yields a value of 0.333, giving no reason to doubt that Eq. 8.14 correctly describes the random experiment with 1024 single values. For 100 times more single values, e.g., Fig. 8.10b, the Chi^2 test yields a value of 0.0001, i.e., only in 1 of 10,000 cases does an even greater deviation occur between the data of a sample and the model for the population.
- **Alac** Then, we can presumably no longer claim that the theoretical distribution describes the experimental distribution well. Why not?
- **Mag** For 102,400 individual values, the approximation of Eqs. 8.13 and 8.14 is not good enough. For so many data, statistical science expects a deviation that is even smaller than we can note with the eye.

Exactly calculated frequencies of occurrence

The probability of finding a random number in an interval is determined exactly by the difference of the cumulative distribution function cdf at the interval boundaries, as prescribed in Eq. 8.12. To determine the frequency in the interval, the probability must be multiplied by the total number N of the analyzed data:

$$freq(x_1 \text{ to } x_2) = (cdf(x_2) - cdf(x_1)) \cdot N \quad (8.15)$$

In this formula, the interval width Δx does not show up, in contrast to Eq. 8.14.

A comparison of the frequencies expected according to Eq. 8.15 with the experimental frequencies in the intervals in Fig. 8.10b results in $\text{CHISQ.TEST} = 0.37$. So, everything is fine again.

- The more individual data are available for a statistical test, i.e., the larger N is, the easier it is to detect discrepancies between experimental frequencies and those predicted by a model distribution. For only a few measurements, deviations from a wrong model distribution may not be noticeable, because deviations from the correct model distribution can also be large.

What is exact, and what is practical?

- ▶ **Mag** We have often calculated the theoretical frequency in an interval with Eq. 8.14 as (*pdf* in the center of the interval) times (interval width) times (total number of data). What can you say about this approach?
- ▶ **Tim** It's wrong. You have to take the difference of the distribution function at the interval limits times the total number, according to Eq. 8.15.
- ▶ **Alac** However, this is a little cumbersome to program. The approximate method gives faster results, and you can graphically display the probability density with a polygon line.
- ▶ **Mag** That's true. The path via probability density is an approximation. But in many cases, especially in experiments with smaller amounts of data, it does not lead to noticeable differences from the exact method with the distribution function *cdf*.
- ▶ **Alac** That's what I'm saying. So, it's more practical.
- ▶ **Tim** Agreed! Nevertheless, I only use this as a first approximation and will take the trouble to apply the method with the *cdf* at the end of my work.

What can we learn philosophically ;-)?

The philosopher Karl Popper stated that scientific theories cannot be confirmed, but only falsified. We find an example of this in our exercise. Let us forget the arguments in the above dialogue and again accept Eq. 8.14 as a valid theory.

In Chi^2 tests, this model provides sufficiently large values for small amounts of data. However, this finding *does not confirm* the theory; it only gives us *no reason to reject* it. It is only for large amounts of data that the Chi^2 test yields such small values that we may reject the theory with a very small probability of error.

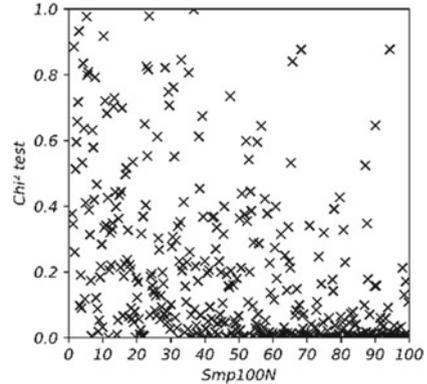
In the history of physics, there have always been cases in which only a refined measuring technique was able to falsify a theory that then turned out to be merely an approximation. An example: In Newtonian mechanics, the mass of a body is independent of the velocity of the body, but this is not so in Einstein's theory of special relativity.

Chi^2 test with ever bigger sample sizes

We are going to perform an experiment in which, four times for each integer $Smp100N$ (= 1 to 100), we:

- create an array with $1024 * Smp100N$ normally distributed random numbers
- determine their observed frequencies of occurrence f_{Obs}
- compare f_{Obs} with frequencies f_{Xpt} expected from the probability density $pdf(x_c)$ in the center x_c of the intervals by means of Chi^2 tests.

Fig. 8.11 Results of Chi^2 tests for sets with ever more samples ($1024*N$)



The results of the Chi^2 test are shown in Fig. 8.11 as a function of N . The number of random numbers is $1024 * \text{Smp}100N$. For a small size of the set ($\leq 10 * 1024$ for $\text{Smp}100N \leq 10$), the values of the Chi^2 test seem to be equally distributed, so that there is no reason to doubt that our approximation describes the empirical results correctly. For increasing size, the values of the Chi^2 test tend to be low, indicating that the theoretical model is not correct. The bigger the size of the set, the less probable the high values of the Chi^2 test are. Nevertheless, values above 0.5 may occur.

Take some time to think about this exercise! It will tell you a lot about the reliability of a Chi^2 test, a model for other statistical tests, about statistical reasoning and the Philosophy of Science.

8.5.4 Data Structure and Nomenclature

The normal distribution

x_m, x_{Sd}	mean and standard deviation of the normal distribution
$pdf(x_m, x_{Sd})$	$x \rightarrow p$, probability density function
$cdf(x_m, x_{Sd})$	$x \rightarrow p$, distribution function
$ppf(x_m, x_{Sd})$	$p \rightarrow x$, inverse of cdf , “percent point function”
x	array of arguments, equally spaced
p_D	pdf of x
p_C	cdf of x
p	array of probabilities, equally spaced between 0 and 1.

Normally distributed random numbers

N	size of the set of random numbers
x_N	array of numbers created with <code>randn(N)</code>
$freq_N$	frequency of occurrence of x
x_P	array of numbers created with <code>ppf(npr.random(N))</code>
x_b	boundaries of sorting intervals (bins)
$freq_P$	frequencies of occurrence of x_P
$freq_{x_{pC}}$	frequencies of occurrence expected from <i>cdf</i>
$freq_{x_{pP}}$	frequencies of occurrence expected from <i>ppf</i> .

8.5.5 Python

Normal distribution

Table 8.11 shows the program for generating arrays with values of these functions to be displayed as three data series in Fig. 8.9. The functions concerning the normal distribution have to be made available from the library `scipy.stats`, usually imported under the abbreviated name `sct` with `import scipy.stats as sct`.

Random Gaussian generator

The Python program in Table 8.12 creates a random sample of 10,000 normally distributed numbers in two ways, first, with the function `npr.randn(10,000)`, and second, with the function `npr.random` (generating random numbers between 0 and 1) inserted into `sct.norm.ppf`, the inverse of the distribution function. The second version corresponds to the nested EXCEL function and can generally be applied to other distributions, e.g., to get the \cos^2 -distribution needed in Exercise 8.7.

Next, the empirical frequency distribution of the two samples is determined within 23 intervals with the boundaries $I_b = -\infty, -3, -2.7, \dots, 2.7, 3, \infty$. This is done with the function `np.histogram` that expects the sample and the interval boundaries as input and returns the empirical frequencies on row 0 ($freq[0]$ in Table 8.12) and the unchanged interval boundaries on row 1. The data obtained on x and x_P are displayed in Fig. 8.9a, together with the theoretically expected frequencies $freq_{x_{pC}}$.

Table 8.11 Python functions related to the normal distribution

```

1 import scipy.stats as sct
2 xN=np.linspace(-4,4,41)
3 pD=sct.norm(0,1).pdf(xN)      #Probability density function
4 pC=sct.norm(0,1).cdf(xN)     #Cumulated density function
5 xP=sct.norm(0,1).ppf(pC)     #Percent point function

```

Table 8.12 Normally distributed random numbers x and x_p

```

6  N=10000
7  xN=np.random.randn(N)
8  xP= sct.norm.ppf(np.random.randn(N)) #Corresp. to Excel formula
9
10 b=np.linspace(-3,3,21)
11 dx=b[1]-b[0] #Interval width
12 xb=np.empty(len(b)+2) #Interval boundaries
13 (xb[0], xb[1:-1], xb[-1])=(-np.inf,b, np.inf)
14 freqXpC=(sct.norm(0,1).cdf(xb[1:])) #Expected freqs.
15          -sct.norm(0,1).cdf(xb[:-1]))*N
16 xc=np.ones(len(xb)-1) #Centers of intervals ("bins")
17 xc[1:-1]=(xb[1:-2]+xb[2:-1])/2
18 xc[0],xc[-1]=xb[1],xb[-2]
19
20 freqN =np.histogram(xN,bins=xb)
21 freqP =np.histogram(xP,bins=xb)

```

x	[1.20 0.88 1.67 ... -0.78 -1.82]
xb	[-inf -3.00 -2.70 ... 3.00 inf]
freqN[0]	[17 22 50 ... 16 18]
freqXpC	[13.5 21.2 47.3 .. 21.2 13.5]

8.5.6 Excel

Normal distribution

We create maps of the probability density of the normal distribution and its distribution function. In the following spreadsheet, Fig. 8.12 (S), 41 values of the probability density pdf and the distribution function cdf are generated and shown in the diagrams of Fig. 8.8.

	A	B	C	D	E	F	G
2	dx	0.2					
3	$=A5+dx$	$=NORM.DIST(x;0;1;FALSE)$	$=NORM.DIST(x;0;1;TRUE)$			$=NORM.INV(p;0;1)$	
4	x	pdf(0,1)	cdf(0,1)		p		
5	-4.0	0.000	0.000		0.0	#NUM!	
6	-3.8	0.000	0.000		0.1	-1.28	
14	-2.2	0.035	0.014		0.9	1.28	
15	-2.0	0.054	0.023		1.0	#NUM!	
45	4.0	0.000	1.000				

Fig. 8.12 (S) Function table for the normal distribution (probability density $pdf(0,1)$) with the mean value 0 and the standard deviation 1, column B), the associated cumulative (integral) normal distribution or distribution function ($cdf(0,1)$, column C), and the inverse of the distribution function in column F

	A	B	C	D	E	F	G	H
1	=COUNT(xN)	dx		0.5				
2	1024			N	1024	=SUM(freq)		
3	=NORM.INV(RAND();0;1) =AVERAGE(D5:D6) =D5+dx {=FREQUENCY(xN;xb)} =NORM.DIST(xc;0;1;0)*N*dx =CHISQ.TEST(E6:E17;freqXpC)							
4	xN	xc	xb	freq	freqXpP			
5	-0.38	-3	-3	0				
6	-1.72	-2.75	-2.5	4	4.7	0.51		
17	1.41	2.75	3	6	4.7			
18	-0.64	3		0				
1028	-1.14							

Fig. 8.13 (S) Table layout used to generate normally distributed random numbers and to determine their experimental (column E) and theoretical (column F) frequencies; the Chi² test is performed in G6

A spreadsheet function for a Gaussian random generator

To create a set of normally distributed numbers in EXCEL, we use

$$\text{Norm.Inv}(\text{Rand}(); 0; 1)$$

corresponding to `set.norm(0, 1).ppf(npr.rand())` in Python. It claims to return random numbers distributed according to the probability density of a Gaussian bell curve with the mean value 0 and the standard deviation 1 (standard normal distribution).

We check this claim in Fig. 8.13 (S) by entering the above function into 1024 cells named x_N , thus generating 1024 random numbers. A frequency distribution $freq$ is calculated from them for the interval boundaries x_b and displayed in a diagram over the interval centers x_c (see Fig. 8.10a). The expected theoretical frequency distribution $freq_{XpP}$ is calculated *with the probability density* in the center of the intervals.

For a normal distribution with mean value 0 and standard deviation 1 (standard normal distribution), frequencies as those for $freq_{XpP}$ in column F are expected. The corresponding formula (F3) is composed of three terms:

- $\text{NORM.DIST}(x_C; 0; 1; 0)$, the probability density *pdf* in the center x_c of the interval, where the second and third positions indicate the mean value (here, 0) and standard deviation (here, 1) of the Gaussian curve. The fourth position (FALSE or 0, TRUE or 1) determines whether the probability density or, otherwise, the distribution function is to be returned.
- The width dx of an interval (here, 0.5).
- The total number N of random numbers (here, 1024).

These expected frequencies are shown in Fig. 8.10a as a polyline. The Chi² test in G6 gives a value of 0.51. So, we have no reason to doubt that our function $\text{NORM.INV}(\text{RAND}();0;1)$ produces standard normally distributed (=

	A	B	C	D	E	F
1				50	15000	=SUM(fSum)
2	=RAND()			{=FREQUENCY(x;x:b)} from Macro		
3	x	fb	Freq	fSum		
4	0.30	0.1	3	1511		
5	0.31	0.2	6	1530		
11	0.73	0.8	9	1523		
12	0.17	0.9	7	1468		
13	0.71		6	1538		
53	0.76					

1	Sub SumHist()	1
2	For rep = 1 To 100	2
3	Application.Calculation = xlCalculationManual	3
4	For r = 4 To 13	4
5	Cells(r, 5) = Cells(r, 5) + Cells(r, 4)	5
6	Next r	6
7	Application.Calculation = xlCalculationAutomatic	7
8	Next rep	8
9	End Sub	9
10		10

Fig. 8.14 **a** (left, S) The frequency of 50 numbers x in column A is determined in column D. The sum of the frequencies in column E has been obtained with SUB *SumHist* in **b** (P). **b** (right, P) Procedure adding up the frequencies in column D into f_{Sum} . Two nested loops apply: (FOR $r = \dots$) adds up the frequencies once, the superordinate loop (FOR $rep = \dots$) repeats this process. CELLS(4,5) IN THE LOOP (FOR $R = 4 \dots$) IN THE PROCEDURE IS CELL E4 IN THE SPREADSHEET

Gaussian-distributed) random numbers, and that their frequency of occurrence is theoretically described using the probability density in the center of the intervals.

We shall use this function in later chapters with various standard deviations to simulate noise during measurements.

Questions

The distribution of 102,400 random numbers is displayed in Fig. 8.10b. How can you supplement the EXCEL solution in Fig. 8.13 (S) designed for 1024 random numbers so as to also get such a large number?²³

Summing Up Frequencies with a VBA Routine

It is often practical to set up a calculation model in which frequencies are first determined on a small sample. If the spreadsheet calculation runs without errors, then the statistics can be made more extensive by repeating the random experiment several times and adding up the frequencies found. An example is given in Fig. 8.14.

The procedure *SumHist* adds, in the inner loop ($r = \dots$), the values in *Freq* to the values in f_{Sum} and repeats this 100 times. Before the ($r = \dots$) loop, the automatic calculation is switched off (line 3), because, otherwise, all random numbers would be generated anew and every entry in the spreadsheet and all frequencies recalculated. This would make the check number in E1 unequal to the total number of sample points as it should not be. Such a discrepancy could therefore be used to identify incorrect programming. After finishing the ($r = \dots$) loop, the automatic calculation is switched on again to generate the random numbers x anew.

Questions

Questions concerning Fig. 8.14:

²³ With a rep-log procedure adding up the frequencies in *freq* 100 times. Continue reading!

How often was *SumHist* of (P) called to yield the results in Fig. 8.14a (S)?²⁴

Why is the automatic calculation in SUB *SumHist* Fig. 8.14b (P) switched off before adding the frequencies?²⁵

What would happen if the automatic calculation were not switched on again after a summation of the frequencies?²⁶

► **Task** Write a log procedure for repeating the random experiment of the previous task in Fig. 8.13 one hundred times and add up the frequencies! Do not forget to switch off the automatic calculation while the current frequency distribution is added to the sum! Next, create a frequency distribution and do a Chi^2 test!

8.6 Random-Number Generator, General Principle

A random-number generator for a desired probability distribution *pdf* can be created if the inverse function *ppf* of the associated cumulative distribution function *cdf* exists and is “fed” with random numbers equally distributed between 0 and 1.

Why do `Norm.Inv(Rand();0;1)` and `sct.norm.ppf(npr.random)` generate normally distributed random numbers?

The answer to this question is made plausible with Fig. 8.15. There, the probability density *pdf* of the standard normal distribution is represented with a bell curve, and the kink points of the right angles lie on the distribution function *cdf*. The interval limits of the uniformly subdivided *x*-axis are transferred to the *y*-axis by the *cdf* of the normal distribution.

The width of the intervals on the *y*-axis is $\Delta y = dcdf(x)/dx \cdot \Delta x$. So, if the *y*-axis is “fed” with random numbers between 0 and 1 and transferred by *ppf* to the *x*-axis, a distribution of the *x*-values corresponding to $dcdf(x)/dx$ results, i.e., according to the derivative of the *cdf*, and thus a distribution proportional to the associated probability density $pdf(x) = dcdf(x)/dx$.

²⁴ SUB *SumHist* ran three times. Each time, the random experiment was repeated one hundred times, so that the summed frequency distribution captures $3 \times 100 \times 50 = 15,000$ numbers (see cell E1 in Fig. 8.14a).

²⁵ Every single summation would otherwise lead to a recalculation of all random numbers, and thus to changed frequencies of occurrence. This would be noticed thanks to the fact that, in cell E1 of Fig. 8.14a (S), there would be no number corresponding to $3 \times 100 \times 50$. The sum in cell E1 is therefore a check as to whether an error has been made in the spreadsheet or in the procedure.

²⁶ The random numbers would not be determined anew, and the same frequencies would always be added up.

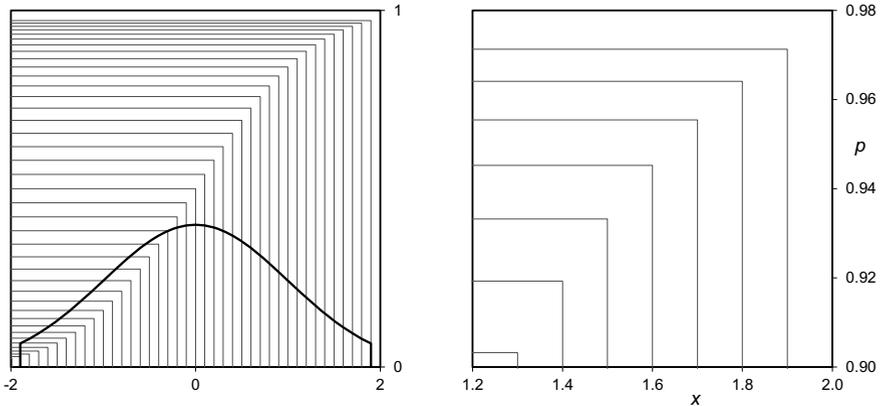


Fig. 8.15 **a** (left) Bell curve: Normal distribution. **b** (right) Blow-up of **a**, $\Delta y = (dy/dx) \cdot \Delta x$

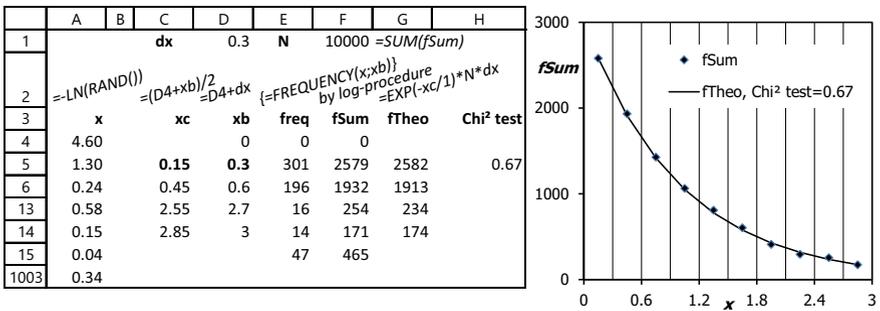


Fig. 8.16 **a** (S) Generating exponentially distributed random numbers. **b** (right) Display of data from **a**

Distributions generated with simple mathematical functions

We generalize our findings: The inverse function $ppf(p)$ of a distribution function $cdf(x)$ can be used as a random generator for the associated probability density $pdf(x)$:

$$y = P_I(x) \quad (\text{distribution function, } cdf)$$

$$\frac{dy}{dx} = p(x) \quad (\text{probability density, } pdf)$$

$$x = P_I^{-1}(y) \quad (\text{ppf, inverse of the distribution function})$$

All distributions for which the inverse function of its distribution function can be built can be generated using standard functions. We still have to make sure that the argument range of the inverse function is correctly covered by the range from 0 to 1. Examples can be found in Table 8.13.

Table 8.13 Some functions, their integrals, and the inverse functions of the integral, suited to generating random number distributions (EXCEL)

pdf(x)	cdf(x)	ppf(p)
Equi-distribution 0 to 1		RAND()
Normal distribution	NORM.DIST(X;X _m ;X _σ ;TRUE)	NORM.INV(RAND())
Exponential	Exponential	-LN(RAND()), natural logarithm
Cosinus	Sinus	Arcus sinus(2*(Rand()-0.5)
2x for x = 0 to 1	x ²	(Rand()) ^{1/2}
3x ² for x = 0 to 1	x ³	(Rand()) ^{1/3}
1/(π · (1 + x ²))	Arcus tangens(x)	Tan(π(Rand()-0.5)
Cauchy-Lorentz		

Questions

How do you construct a distribution $p(x) = c \cdot x^3$ for $0 \leq x < 1$? How big is c ?²⁷

Exponential distribution

As an example, we construct a random-number generator that produces a decreasing exponential distribution from $x = 0$ to ∞ . We know that

$$\text{pdf}(x) = \exp(-x) \rightarrow \text{cdf}(x) = 1 - \exp(-x) \rightarrow \text{ppf}(p) = -\ln(x)$$

So, -LN(RAND()) should do the job in EXCEL; it is implemented in Fig. 8.16a (S) with the results shown in **b**.

Questions

How many random numbers are generated in Fig. 8.16a under the name x ?²⁸

What is the total size of f_{Sum} displayed in Fig. 8.16b?²⁹

How was f_{Sum} most likely calculated from $freq$?³⁰

► **Task** Generate random numbers that are distributed according to a cosine arc and check the results with a Chi² test!

²⁷ The antiderivative of $p(x)$ is $P(x) = c/4 \cdot x^4$; $P(1)$ must be 1 $\rightarrow c = 1/4$. The inverse function of $P(x)$ is $\sqrt[4]{Rand()}$. The random function is thus [=RAND()^0.25] in EXCEL or np.random(N)**0.25 in Python.

²⁸ The array x comprises 1000 numbers (rows 4 to 1003).

²⁹ The array f_{Sum} comprises $N = 10,000$ numbers (cell F1).

³⁰ By a rep-log procedure summing up $freq$ 10 times (similar to Fig. 8.14b).

► **Task** Generate random numbers distributed between 0 and 1 according to $p(x) = 3x^2$. Determine the theoretically expected frequencies, approximated with the probability density, Eq. 8.14, as well as exactly with the distribution function, Eq. 8.15. Perform χ^2 tests to see if the formulas given in the table are correct and for what number of random numbers the approximation of Eq. 8.14 is good enough! In EXCEL, use a *rep-log procedure* that repeats the random experiment!

8.7 Diffraction of Photons at a Double-Slit

We simulate the diffraction of photons at a double-slit, intending to demonstrate the wave-particle duality of light. For this, we need a random-number generator distributing the impact points of the photons on a screen according to a \cos^2 probability density. The goal concerning programming is to learn how to implement such a random-number generator using a finite polyline.

8.7.1 Physical Background: Wave-Particle Dualism

In Fig. 8.17a, b, the diffraction of electrons or photons at a double slit is simulated to illustrate the wave-particle dualism. The diffraction image is created by many particles that hit a screen behind the double-slit randomly. The image does not consist of stripes behind each slit, but is rather an interference pattern with a maximum behind the middle position between the two slits.

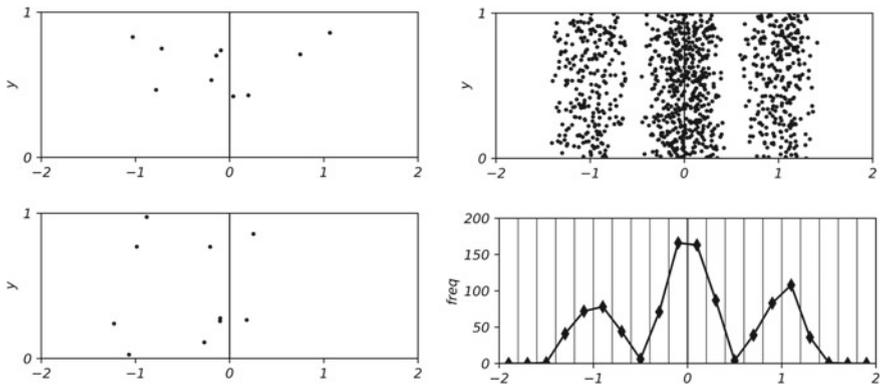


Fig. 8.17 **a** (left) Two diffraction images of ten photons each. **b** (right) Top: Diffraction image of 1000 photons, bottom: Distribution of the x -coordinate of the 1000 photons (the y -coordinate is evenly distributed)

In physical reality, the distance of the screen to the slits is large against the distance between the slits, and the independent variable of the interference pattern is the angle with respect to the mid-perpendicular of the double-slit.

We may imagine the screen upon which the diffraction image is captured as a gelatine film with silver grains or as a modern CCD detector with pixels. The impact of a photon is shown by the fact that a single silver grain or a single pixel is activated. Figure 8.17a shows two typical images after the impact of 10 photons. Such experiments in the real world show that light is “granular”, i.e., consists of energy packets that are locally confined.

The events in Fig. 8.17a seem to be evenly but randomly distributed over the entirety of the detector surface. However, after 1000 photons have been detected, a pattern like that in Fig. 8.17b emerges; the positions $x = -1/2$ and $x = 1/2$ have never been hit by any photon. This is a consequence of the wave character of light resulting in a diffraction image consisting of a central maximum and secondary maxima of the same width. The distance between the maxima is determined by the reciprocal of the distance between the two slits. The ratio of the intensities depends on the width of a single slit. Here, we have arbitrarily chosen a ratio of 2 to 1 between the central maximum and the side maxima.

If ... then ... else ...

The x -coordinates x_{Cos} generated by a \cos^2 -generator are to be distributed according to the diffraction figure:

- 50% in the central maximum of width 1 (position at $x = 0$),
- 25% in the right side-maximum of width 1 (position at $x = 1$), and
- 25% in the left side-maximum of width 1 (position at $x = -1$).

This can be achieved with a random number rnd between 0 and 1 and a logical query:

If $0.5 < rnd$, then $x = x_{\text{Cos}}$ (in 50% of cases).
 else If $rnd < 0,75$, Then $x = x_{\text{Cos}} + x_0$ (in $75\% - 50\% = 25\%$ of cases).
 else, $x = x_{\text{Cos}} - x_0$ (in the remaining 25% of cases).

Questions

How do the critical numbers in the IF query have to be changed when the maxima are to occur at x_0 , $x_0 + 1$, and $x_0 + 2$ and intensity ratios of 6:3:1 are to be obtained?³¹

In a spreadsheet, we can make a “living picture”; with each change of the spreadsheet (e.g., when an already empty cell is “deleted”), the coordinates are

³¹ If $rnd < 0.6$, then $x = x_{\text{Cos}} + x_0$; Else, if $rnd < 0.9$, then $x = x_{\text{Cos}} + x_0 + 1$; Else, $x = x_{\text{Cos}} + x_0 + 2$.

recalculated and the points in the diffraction image with ten photons, Fig. 8.17a, jump around erratically. The attentive observer may suspect that the points stay away from the straight lines $x = -1$ and $x = 1$, but this observation is not convincing.

In Python, we can create such a “living picture” through animation as shown in Sect. 6.2.5.

8.7.2 Cos² Distribution

Our task is to build a random generator that distributes the particles according to the diffraction pattern. We only want to simulate the central maximum of the diffraction figure at $x = 0$ and the first two secondary maxima around $-x_0$ and $+x_0$ (here $x_0 = 1$, arbitrary units on the screen). The intensity distribution in each maximum shall be approximated by the same \cos^2 probability density (see Fig. 8.18a),

$$\text{CosSq_pdf}(x) = 2 \cdot \cos^2(\pi x) \tag{8.16}$$

however, with different amplitudes.

The distribution function $\text{cdf}(x)$ is the integral of this probability density function $\text{pdf}(x)$ with an argument range from -0.5 to 0.5 and a value range from 0 to 1:

$$\text{CosSq_cdf}(x) = \frac{\sin(2\pi x)}{2\pi} + x + \frac{1}{2} \tag{8.17}$$

It is shown in Fig. 8.18a (“CosSq_cdf”).

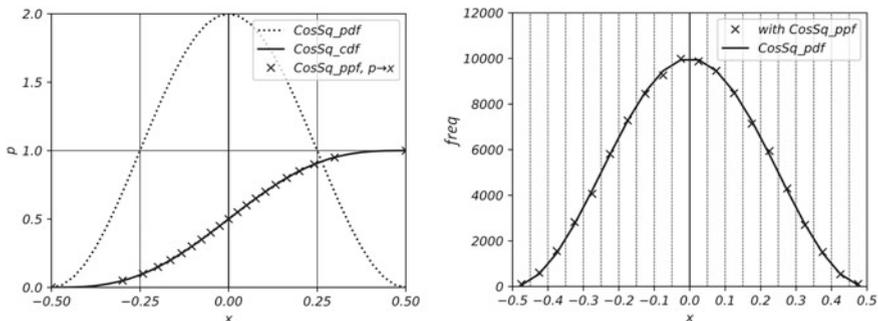


Fig. 8.18 **a** (left) Functions related to the \cos^2 distribution, data from Table 8.15. **b** (right) The frequencies of occurrence of 100,000 outputs of the function $\text{CosSq_ppf}(\text{rand})$ are actually \cos^2 distributed

Questions

How big are the values $CosSq_cdf(-0.5)$, $CosSq_cdf(0)$, $CosSq_cdf(0.5)$ in Eq. 8.17³²

How big is the area under the curve $CosSq_pdf$ in Fig. 8.18a?³³

For our random-number generator, we need the inverse function of $CosSq_cdf(x)$ in Eq. 8.17.

Solution in three steps

- We create a function that distributes \cos^2 -distributed values in the range -0.5 to 0.5 .
- We generate ten points according to the desired diffraction image (as in Fig. 8.17a).
- We repeat the random experiment with the ten points one hundred times and determine the frequency distribution of the x -values (as in Fig. 8.17b).

The gradual emergence of the diffraction pattern can be demonstrated with animation.

8.7.3 Data Structure and Nomenclature

$CosSq_pdf(x)$	$p(x) = 2 \cdot \cos^2(\pi \cdot x)$
$CosSq_cdf(x)$	integral of $CosSq_pdf(x)$
$CosSq_ppf(p)$	inverse of $CosSq_cdf(x)$
x_1	array of x , 33 values from -0.5 to 0.5
p_1	$CosSq_cdf(x_1)$
$m[0], \dots$	slope of p_1 in interval $x_1[0]$ to $x_1[1]$
x_{Cos}	x -values, random according to a \cos^2 distribution.

8.7.4 Python

We define the three functions related to the \cos^2 distribution:

- $CosSq_pdf(x)$, probability density function, argument range from $-0.5 \leq x \leq 0.5$, normalized so that the area under the curve is 1.

³² $p(-0.5) = 0$; $p(0) = 1/2$; $p(0.5) = 1$.

³³ (Area under \cos^2) = (triangle $(0.5 - (-0.5)) \cdot 2/2$) = 1. This is the condition for a probability density function.

Table 8.14 Python functions related to the \cos^2 distribution

```

1  def CosSq_pdf(x):
2      return 2*(np.cos(x*np.pi)**2)
3
4  def CosSq_cdf(x):
5      return np.sin(2*np.pi*x)/(2*np.pi)+x+1/2
6
7  xI=np.linspace(-0.5,0.5,33)
8  pI=CosSq_cdf(xI)
9  m=((pI[1:]-pI[:-1])/
10     (xI[1:]-xI[:-1]))      #Slope
11 #m[0] is slope of interval pI[0] to pI[1]
12
13 def CosSq_ppf(p):
14     for i in range(1,33):
15         x0=0
16         if pI[i-1]<=p<=pI[i]:
17             x0=xI[i-1]+1/m[i-1]*(p-pI[i-1])
18             return x0      #Why return within if?

```

- $\text{CosSq_cdf}(x)$, cumulative density function (distribution function), argument range from $-0.5 \leq x \leq 0.5$, monotonously increasing from 0 to 1.
- $\text{CosSq_ppf}(p)$, percent point function, the inverse of cdf , argument range $0 \leq p \leq 1$.

These are all displayed in Fig. 8.18a.

The three functions are defined in Table 8.14. The probability density function CosSq_pdf essentially returns a value of \cos^2 ; CosSq_cdf is its antiderivative. For the inverse of the distribution function (cdf), a closed expression does not exist. We therefore approximate it with a polyline with 33 vertices. CosSq_cdf for the x -values of the vertices is calculated in array p_I and the slope within the 32 intervals in m . The three arrays thus obtained are used to get the linear approximation for CosSq_ppf , with p as the independent variable and x as the dependent variable, and the slope $1/m$ between the vertices.

Questions

Is there a variant of CosSq_ppf that can suffice with 5 if queries for the 32 intervals?³⁴

How do you plot the results of Table 8.15?³⁵

³⁴ See the program code in Table 8.16.

³⁵ Plot with: `plt.plot(xC,hist[0], 'kx')`; `plt.plot(xC,theo, 'k- ')`.

Table 8.15 $N = 100,000$ outputs of the \cos^2 -generator

```

19 N=100000
20 rn=np.random(N)
21 xCos=list(CosSq_ppf(r) for r in rn)
22 dx=0.05 #Interval width
23 xb=np.arange(-0.5,0.5+dx,dx) #Endpoint not included
24 xc=(xb[1:]+xb[:-1])/2
25 hist=np.histogram(xCos,xb) #Empirical frequencies
26 theo=N*dx*CosSq_pdf(xc) #Theoretical frequencies

```

Table 8.16 Interval search with five queries

```

1 def CosSq_ppf2(p):
2     iLow=0
3     iHigh=32
4     for i in range(5):
5         iMid=np.int((iLow+iHigh)/2)
6         if p <= pI[iMid]:iHigh=iMid
7         else:iLow=iMid
8     x0=xI[iLow]+1/m[iLow]*(p-pI[iLow])
9     return x0

```

CosSq_ppf is used in Table 8.15 to create 100,000 random numbers, the frequency distribution of which is displayed in Fig. 8.18b, together with the theoretical curve obtained with *CosSq_ppf*.

Interval search with five queries

Table 8.16 realizes the search in 32 intervals with 5 queries.

8.7.5 Excel

User-defined spreadsheet function

We develop a user-defined spreadsheet function that approximates the random-number generator for a \cos^2 -distribution piecewise linearly (see Fig. 8.19 (P)), as has been done in the Python program. Remember: a user-defined spreadsheet function must be in a module. It must not be assigned to a specific worksheet (see Exercise 4.9).

Global data arrays for x_I , p_I , and m are defined at the top of the VBA sheet, in lines 1 to 3 of Fig. 8.19 (P), and can be called by all procedures and functions in the same VBA sheet or module. They must be initialized with the correct numbers before the first call of the function.

1 Public xI(32) As Single	Sub init()	14
2 Public pI(32) As Single	xI(0) = -0.5	15
3 Public m(32) As Single	xI(1) = -0.46875	16
4	17
5 Function CosSq_ppf(p0)	xI(32) = 0.5	18
6 For i = 1 To 32		19
7 If (pI(i - 1) <= p0 And p0 <= pI(i)) Then _	pI(0) = 0	20
8 co = xI(i - 1) + m(i) * (p0 - pI(i - 1))	pI(32) = 1	21
9 Next i		22
10 CosSq_ppf = co	For i = 1 To 32	23
11 End Function	m(i) = (xI(i) - xI(i - 1)) / (pI(i) - pI(i - 1))	24
12	Next i	25
13	End Sub	26

Fig. 8.19 (P) Function *COSSQ_PPF* approaches the inverse function of $P_1(x)$, *CosSq_cdf(x)* in Eq. 8.17, piecewise linearly as a polyline. The global fields x_1 , p_1 , and m must be initialized with the correct values, here, with *SUB init*, before the first call of the function. The text of the statements in *init* is generated in a spreadsheet (see Fig. 8.20 (S), columns E and F)

The function *COSSQ_PPF* corresponds to the Python function of the same name. The (FOR i = ..) loop queries in lines 7 and 8 for each of the 32 intervals whether p_0 lies in this interval. If this is the case, the value of the function is calculated as a linear interpolation between the vertices of the segment, the coordinates of which are stored globally in the data arrays $x_1(32)$ and $p_1(32)$. The slopes in the intervals are in the global array $m(32)$.

Questions

Is there a variant of *Cos2_ppf* that can suffice with five If queries for the 32 intervals?³⁶

VBA code generated in a spreadsheet

Initialization of the global arrays x_1 and p_1 is done in a total of 66 lines in the procedure *INIT()*, also shown in Fig. 8.19 (P). The array $m(I)$ contains the slope of the curve between points $i - 1$ and i and is calculated from x and p_1 within *init* (lines 23 to 25).

We could enter all 66 lines by hand, which is, of course, tedious and unpleasant, although it would not be much more time-consuming than tracking down errors in spreadsheets and other programs. But there is a more elegant way: we let EXCEL work for us. In various tasks, we have had a VBA procedure write formulas into a spreadsheet. Text in cells preceded by an = sign is interpreted as a formula. Now, we will simply do it the other way around: A text is assembled in a spreadsheet and copied into a VBA procedure to be interpreted there as a formula. This is done in Fig. 8.20 (S).

³⁶ Compare the Python code in Table 8.16.

	A	B	C	D	E	F	G	H
1	0.031 =2/32							
2	=A4+\$A\$1 =SIN(2*PI()*xI)/2/PI()+xI+0.5 ="xI("&n&") = "&xI ="pI("&n&") = "&pI =cosSq_ppf(B5)							
3	xI	pI	n	Array x	Array y			
4	-0.500	0.000	0	xI(0) = -0.5	pI(0) = 0			0.000
5	-0.469	0.000	1	xI(1) = -0.46875	pI(1) = 0.00020041090			0.000
6	-0.438	0.002	2	xI(2) = -0.4375	pI(2) = 0.00159404009			0.000
35	0.469	1.000	31	xI(31) = 0.46875	pI(31) = 0.9997995890			0.000
36	0.500	1.000	32	xI(32) = 0.5	pI(32) = 1			0.000

Fig. 8.20 (S) In columns A and B, there are 33 points on the curve $p_I(x)$, Eq. 8.17; in columns D, E, and F, the VBA code for initialization of the arrays x_I and p_I is generated from columns A and B. This text is to be copied into the VBA editor, SUB *Init* in Fig. 8.19 (P)

A program is just a text, composed of code words, that is translated into computer instructions.

The column vectors x_I and p_I correspond to the lists with the same names as in the Python program (Table 8.14). The code for the VBA -function of the inverse $ppf(p)$ of the distribution function $cdf(x)$ ($x_I(0) = \dots, p_I(0) = \dots$) in columns E and F is generated in the spreadsheet with text processing and copied into SUB *INIT()* in Fig. 8.19 (P), which initializes the global data arrays x_I, p_I , and m .

In columns E and F of Fig. 8.20 (S), text corresponding to VBA code is generated. The spreadsheet formulas in the individual cells consist of text elements and numbers, e.g., in E5: [= "xI("&n&") = "&xI], and yields $[x_I(1) = -0.4687]$. The ranges E4:E36 and F4:F36 written in this way are then transferred to the Visual Basic editor by text copying.

- ▶ **Alac** The values can be calculated in the routine itself using Eq. 8.17.
- ▶ **Mag** Yes, that is possible. But by detouring via the spreadsheet, we have practiced the way in which code is generated as text in a spreadsheet and transformed into formulas. Code is nothing more than structured text interpreted by a programming language interpreter.

8.7.6 Simulation in a Spreadsheet

We will simulate the evolution of an interference pattern in a spreadsheet with two figures like the upper two in Fig. 8.17. Ten new photon impacts are shown every second in a snapshot (left); they are accumulated for the right figure.

Random experiment with 10 to 1000 photons

In Fig. 8.21 (S), we calculate the coordinates of ten photon impacts to be presented in the snapshot. The preliminary to the x -component is calculated as x_{Cos} with the user-defined spreadsheet function [= *COSSQ_PPF(RAND())*]. The random numbers x_0

	A	B	C	D	E	F	G	H	I	J	K
1			=IF(m<0.5;xCos;IF(m<0.75;xCos+1			1004	490	bnd)	0.16	1000	
2	=cosSq_ppf(RAND()) =RAND()		=IF(m<0.5;xCos;IF(m<0.75;xCos+1						=AVERAGE(J5:J14) =J4+\$J\$1		=FREQ
3	xCos	rn	x	y	x.1000	y.1000	bndC	bnd	freq		
4	-0.22	0.09	-0.22	0.08	-0.28	0.51	-2.00	-2.00	0		
5	-0.09	0.84	-1.09	0.75	-0.22	0.29	-1.92	-1.84	0		
13	-0.32	0.90	-1.32	0.91	-0.04	0.56	-0.64	-0.56	20		
29					0.98	0.23	1.92	2.00	0		
30					1.14	0.81	2.00		0		
31					-0.07	0.86					

Fig. 8.21 (S) Ten random points within the diffraction image of a double-slit are generated in C:D. The total 1000 points in F, G have been accumulated (a 100 times) by SUB *More10* in Fig. 8.22 (P). The formula in K2 is = FREQUENCY (x.1000;bnd)

are distributed over the three maxima at 0, 1, and -1 by means of a second random number *rn* with (see formula in C2 and C1).

$$x = [= IF(0, 5 < rn; xCos; IF(m < 0, 75; xCos + 1; xCos - 1))]$$

according to the specified ratio of the intensity of the maxima 0.5: 0.25: 0.25. Remember: the structure of the logical query in EXCEL is [=IF(LOGICAL_TEST; VALUE_IF_TRUE; VALUE_IF_FALSE)].

The y-component is distributed uniformly between 0 and 1. The points (x, y) are shown in Fig. 8.17a; they change their position with every change in the spreadsheet.

Sub More 10

The upper picture in Fig. 8.17b shows the accumulated photon impacts with the coordinates (x.1000, y.1000) from Fig. 8.21 (S). This range (columns F and G) is successively filled by SUB *More10* in Fig. 8.22 (P). It transfers the ten random coordinate pairs (x, y) from columns C and D consecutively to columns F and G using an index *r2* updated in G1 of the spreadsheet.

Sub Run, random experiment with 1000 photons

With a rep-log procedure, SUB *Run* in Fig. 8.23 (P), we repeat the random experiment *More 10* with 10 photons 100 times and display the increasing number of points in the cumulating diagram, Fig. 8.17b (top), getting at recognizing the pattern better and better.

Over the course of time, we recognize the diffraction pattern more and more clearly. The frequency distribution of the 1000x values in Fig. 8.17b reveals the cos² distribution in the main maximum and in the two side maxima.

```

1 Sub More10()                                1
2 r2 = Sheets("Coord").Cells(1, 7) 'G1        2
3 Application.Calculation = xlCalculationManual 3
4 For i = 0 To 9                               4
5   Sheets("Coord").Cells(i + r2, 6) = Sheets("Coord").Cells(i + 4, 3) 'C4 -> F4, etc. 5
6   Sheets("Coord").Cells(i + r2, 7) = Sheets("Coord").Cells(i + 4, 4) 'D4 -> G4, etc. 6
7   Next i                                       7
8 Application.Calculation = xlCalculationAutomatic 8
9 Wait                                           9
10 Sheets("Coord").Cells(1, 7) = r2 + 10 'index for columns F und G 10
11 End Sub                                       11

```

Fig. 8.22 (P) SUB *More10* writes the ten random coordinate pairs in columns C and D of Fig. 8.21 (S) *successively* into columns F, and G. SUB *Wait* is reported in Fig. 8.23(P). The pointer r_2 for the next free row is updated in `CELLS(1,7) = G1`

1 Sub Run()	Sub Wait()	9
2 init	Dim m As Integer	10
3 Sheets("Coord").Range("F4:G6000").ClearContents	h = Hour(Now)	11
4 Sheets("Coord").Cells(1, 7) = 4	m = Minute(Now)	12
5 For n = 1 To 100	s = Second(Now) + 1	13
6 More10	waittime = TimeSerial(h, m, s)	14
7 Next n	Application.Wait waittime	15
8 End Sub	End Sub	16

Fig. 8.23 (P) In the master procedure SUB *Run*, the coefficients for *CosSq_ppf* are generated by calling SUB *Init* of Fig. 8.19 (P); the old coordinates are deleted in line 3. SUB *More10* (Fig. 8.22 (P)) is called in a loop a hundred times

SUB *Run* is a log procedure that first deletes the old data in columns F and G in Fig. 8.21 (S). These are the coordinates of the points in Fig. 8.17a (top), so that the chart is now empty, and then calls the procedures *More10* and *Wait* a 100 times so that the chart fills up again (within 100 s) with 1000 points. We may call it a “master procedure”, because it controls the program flow.

SUB *Wait* in Fig. 8.23 (P) stops the calculation for 1 s, so that the viewer can better follow how Fig. 8.21b top is filled up with new points and the frequency distribution in Fig. 8.21b bottom takes shape.

Questions

concerning Fig. 8.22 (P):

The VBA procedure reads the index r_2 of the next free line in the range into which the new coordinates of the photon impacts are to be written from the spreadsheet (G1). Alternatively, the index r_2 could also be updated in the

main program SUB RUN() in Fig. 8.23(P). What is the difference between the two alternatives?³⁷

How would you change Sub More10 if only 5 photons are to pass the double-slit?³⁸

8.8 Chi² Distribution and Degrees of Freedom

Distributions of χ^2 values, obtained from statistical experiments on normally-distributed random numbers, are compared with theoretical distributions with appropriate degrees of freedom. The degree of freedom is the number of intervals reduced by 3 if the mean and standard deviation are estimated from the sample and reduced by 1 if they are fixed a-priori.

In this experiment, we generate $N = 1000$ standard-normally distributed random numbers, determine their frequencies of occurrence in 10 intervals, and compare them in Chi² tests with theoretically expected frequencies obtained from two models.

In the first model, we use the values 0 and 1 for mean x_m and standard deviation x_{sd} so that no parameters of the distribution are estimated from the sample, and the degree of freedom is $dof = 9 = \text{number of intervals} - 1$. The reduction by 1 is due to the fact that the frequency in the last interval is not free but determined by the total number and the frequencies in the other intervals.

In the second model, we estimate x_m and x_{sd} from the sample so that $dof = 9 - 2 = 7$. We repeat the statistical experiment with 1000 data $N_c = 10,000$ times and determine the frequencies of occurrence of the values of Chi².

The result of the statistical experiments is shown in Fig. 8.24a. The lines represent frequencies obtained with the theoretical probability density function of Chi², available from the library `scipy.stats`, usually imported as `sct`, here, `sct.chi2.pdf(xc, dof = 9)` and `sct.chi2.pdf(xc, dof = 7)`, where x_c are the centers of the sorting intervals.³⁹ We see that the results of our simulation fit very well with the theoretical curves. So, we have got the *dof* right.

Fig. 8.24b presents the *pdf* of the Chi² distribution for $dof = 5$ and $dof = 15$. The shaded areas represent the results of Chi² tests. They are the accumulated

³⁷ Update of r_2 in *Sub Run()*: initial index always starts at a fixed value of r_2 ; old data are overwritten.

Read r_2 from the spreadsheet: old data remain; new data are appended below the already existing data.

³⁸ Change: line 4 $\rightarrow i = 0$ to 4; line 10 $\dots r_2 + 5$. The program becomes more flexible if an additional variable N is introduced: $i = 0$ to $N-1$; $\dots r_2 + N$.

³⁹ The sorting intervals are often called *bins* into which the data set is to be sorted.

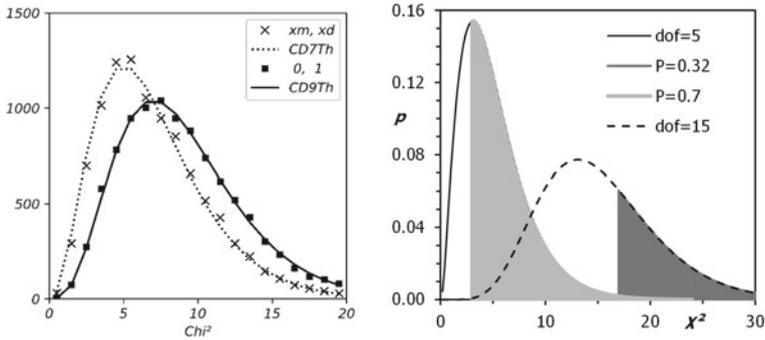


Fig. 8.24 **a** (left) Empirical (x, \bullet) and theoretical ($\dots, \text{—}$) distribution of values of Chi^2 for 7 and 9 degrees of freedom. **b** (right) Pdf of Chi^2 ; the P -value returned in a Chi^2 test corresponds to the shaded area ($P = 0.32$ for $\text{dof} = 5$ and $P = 0.7$ for $\text{dof} = 15$)

probabilities that a sample drawn from the population yields a higher value of Chi^2 when compared to the theoretical distribution of the population.

Keep in mind that, for 10 sorting intervals for normally distributed random numbers,

- $\text{dof} = 9$ when $x_m = 0$ and $x_{\text{Sd}} = 1$ are a-priori fixed,
- $\text{dof} = 7$ when x_m and x_{Sd} are estimated from the sample.

What is the Chi^2 test good for?

We check the hypothesis that empirical frequencies of occurrence arise from a theoretical distribution. The error probability to reject that hypothesis is given by the result of a Chi^2 test.

8.8.1 Data Structure, Nomenclature

- N number of random numbers
- X_N sample of N random numbers, standard-normally distributed
- x_m estimated mean of the sample
- x_{Sd} estimated standard deviation of the sample
- x_b interval boundaries, including $-\infty$ and ∞ , defining 10 sorting intervals (“bins”)
- f_O observed frequencies of x_N in the sorting intervals
- f_{x_9} frequencies expected for a normal distribution $\text{norm}(0, 1)$ in 10 intervals
- f_{x_7} frequencies expected for $\text{norm}(x_m, x_{\text{Sd}})$ with x_m and x_{Sd} estimated from the sample.
- N_{rep} number of repetitions of the statistical experiment
- dof degree of freedom

- cS_7 array for storing N_{rep} Chi^2 values comparing f_O and fx_7
 cD_7 theoretically expected frequencies of cS_7 , pdf of Chi^2 for $dof = 7$
 cS_9 array for storing N_c Chi^2 values comparing f_O and fx_9 , pdf of Chi^2 for $dof = 9$
 cD_9 theoretically expected frequencies of cS_9 .

8.8.2 Python

The basic set-up is given in Table 8.17. 10 intervals from $-\infty$ to ∞ are specified and the frequencies of occurrence for a standard-normal distribution therein for $N = 1000$.

The interval boundaries x_b specified in lines 6 and 7 range from $-\infty$ to ∞ , to capture the whole value range of normally distributed random numbers. They are reported in the lower cell of Table 8.17. The frequencies expected in the intervals are calculated with the cumulated density function at the interval boundaries. For the first and last interval, we have to consider that $cdf(-\infty) = 0$ (line 9) and $cdf(\infty) = 1$ (line 12), which is true, by the way, for any cdf .

The program in Table 8.18 organizes the repetition ($N_{\text{rep}} = 10,000$ times) of the statistical experiment in a for-loop. For every iteration i , a sample x_n of N random numbers is generated, and their frequency distribution f_O , mean x_m , and standard deviation x_d are determined. The frequencies f_O are compared with a Chi^2 test to:

- fx_9 , expected when 0 and 1 are taken as the mean and standard deviation, calculated before the loop (Table 8.17), lines 8 to 11;

Table 8.17 Expected frequencies fx_9 of a normal distribution in 10 intervals with boundaries x_b , $x_m = 0$, $x_{sd} = 1$ set a-priori

```

1 import numpy.random as npr
2 import scipy.stats as sct
3 N=1000                #Size of the set of random numbers
4 db=0.5                #Width of the sorting intervals
5 xbb=np.arange(-2.0,2.0+db,db)
6 xb=np.zeros(len(xbb)+2) #Interval boundaries
7 (xb[0],xb[1:-1],xb[-1])=(-np.inf,xbb,np.inf)
8 fx9=np.zeros(len(xb)-1)
9 fx9[0]=sct.norm(0,1).cdf(xb[1])*N      #Below first bound.
10 fx9[1:-1]=(sct.norm(0,1).cdf(xb[2:-1])
11             -sct.norm(0,1).cdf(xb[1:-2]))*N
12 fx9[-1]=(1-sct.norm(0,1).cdf(xb[-2]))*N #Above last bound.

```

xb	[-inf	-2.00	-1.50	-1.00	-0.50	0.00				
			0.50	1.00	1.50	2.00	inf]			

Table 8.18 Histograms of cS9 and cS7, together with the theoretical probability densities CD9Th and CH7Th of Chi² for dof = 9 and 7 degrees of freedom

13	Nrep=10000	<i>#Number of repetitions</i>
14	cS9=np.zeros(Nrep)	
15	cT9=np.zeros(Nrep)	
16	cS7=np.zeros(Nrep)	
17	for i in range(Nrep):	
18	xn=npr.randn(N)	<i>#Std-normal distribution</i>
19	f0=np.histogram(xn,xb)	<i>#Empirical frequencies</i>
20		
21	ChiSq=sct.chisquare(f0[0],fx9,ddof=0)	
22	cS9[i]=ChiSq[0]	
23		
24	fx7=np.zeros(len(f0[0]))	<i>#For expected frequencies</i>
25	xm=np.average(xn)	
26	xd=np.std(xn,ddof=1)	
27	fx7[0]=sct.norm(xm,xd).cdf(Ib[1])*N	
28	fx7[1:-1]=(sct.norm(xm,xd).cdf(Ib[2:-1])	
29	-sct.norm(xm,xd).cdf(Ib[1:-2]))*N	
30	fx7[-1]=(1-sct.norm(xm,xd).cdf(Ib[-2]))*N	
31	ChiSq7=sct.chisquare(f0[0],fx7,ddof=2)	
32	cS7[i]=ChiSq7[0]	
33	xbC=np.linspace(0,20,21)	<i>#For distrib. of Chi²</i>
34	xc=(xbC[1:]+xbC[:-1])/2	<i>#Centers of intervals</i>
35	CD9=np.histogram(cS9,xbC)	<i>#Empirical freqs.</i>
36	CD9Th=sct.chi2.pdf(xc,df=9)*Nrep	<i>#Theoretical freqs.</i>
37	CD7=np.histogram(cS7,xbC)	
38	CD7Th=sct.chi2.pdf(xc,df=7)*Nrep	

– $fx7$, expected when x_m and x_{sd} are estimated from the sample, calculated individually for every iteration within the loop.

The results of the Chi² tests are stored in cS9 and cS7. The frequency distributions CD9 and CD7 thereof are determined in Table 8.18. The results are shown in Fig. 8.24a.

8.9 Questions and Tasks

Explain the following broom rules:

1. Ψ *Chance is blind and checkered*
2. Ψ *Always one more! But of what and than what? (Concerning frequency distribution.)*
3. Ψ *Come to a decision! Sometimes, it will be wrong.*

Frequencies of occurrence

Initial situation: In the range A1:A1000 of a spreadsheet, named *data*, there are 1000 numbers. In range D2:D10, three interval boundaries $I_b = 0.1; 0.2$ and 0.3 are specified.

4. How many intervals are defined by the interval boundaries?
5. Which numbers are captured in the second and last intervals?
6. The frequencies are to be calculated in a column range starting with E2. Which range must be activated for the spreadsheet function `FREQUENCY` if all numbers are to be sorted into intervals? With which “chord” do you complete the formula input?
7. Over which x values should the frequencies be displayed in a diagram?
8. We want to determine the same frequency distribution with the `numpy` function `np.histogram(data; Inp)`. How are the interval boundaries I_{np} to be defined?

The 1000 numbers are now supposed to be random numbers equally distributed between 0 and 0.5.

9. Which equations do you use in `EXCEL` and `numpy` to generate such random numbers?
10. What mean value do you expect for the 1000 numbers?
11. What frequencies do you expect in the first and last intervals for interval limits of 0.1, 0.2, and 0.3 in the `EXCEL` function?

Normal distribution

Figure 8.25a shows the distribution function $norm.cdf(0;1)$ and the probability density $norm.pdf(0;1)$ of the standard normal distribution. Assume that you have generated 1 million normally distributed random numbers and answer the following questions within the reading accuracy of the figure:

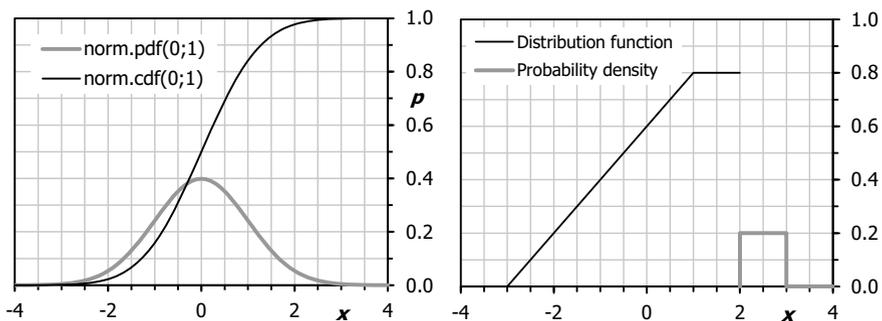


Fig. 8.25 **a** (left) Distribution function $norm.cdf(0;1)$ and probability density $norm.pdf(0;1)$ of the standard normal distribution. **b** (right) A distribution function and its probability density; the distribution function is not displayed above, and the probability density not below $x = 2$

12. How many numbers may be expected to have precisely the value 0?
13. How many numbers are expected to have a value between -0.1 and 0.1 ?
14. How many numbers are expected to have a value between -0.5 and 0.5 ?

Distribution function and random-number generators

15. Within what range do you expect 95% of the output of a function generating normally distributed random numbers?
16. Consider the EXCEL function `NORM.INV(X;5;3)`! What is its argument range? Where is the maximum of the distribution that results if $x = \text{RAND}()$? How do you generate 1000 such numbers in a Range B1:B1000? What is the equivalent function for 1000 numbers in the `numpy.random` library?
17. Figure 8.25b shows an incomplete distribution function and an incomplete probability density. Complete the two functions!
18. What are the inverse functions of $y = \cos(x)$ for $x = 0$ to π and of $y = \exp(-x)$ for $x \geq 0$? Which are their argument ranges and which are their value ranges?
19. Which distribution is generated by the spreadsheet function `(arcus cosinus) ACOS((RAND()-1)*2)`?

User-defined spreadsheet function

In Fig. 8.26a, you find a spreadsheet layout for calculating a house-shaped polyline (see Fig. 8.26b). Implement a function $\text{House}(x)$ for calculating the y -values from the x -values!

The entry in cell A25 of Fig. 8.26a is $6.38\text{E}-16$, instead of the expected zero. This deviation is due to the sum of the rounding errors in the binary addition of $dx = 0.10$. In the user-defined spreadsheet function $\text{House}(x)$, the value of x must therefore be rounded, with `ROUND(X,14)`.

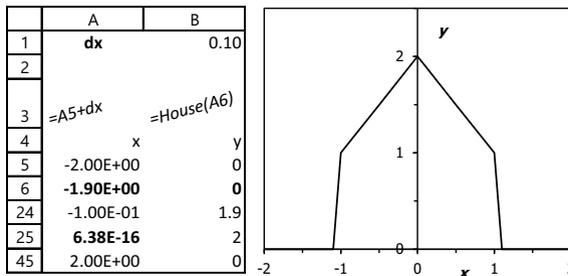


Fig. 8.26 a (left) The values of y are calculated with a user-defined spreadsheet function $\text{House}(x)$. b (right) Graph of the $\text{House}(x)$ function with the data from Fig. 8.26a

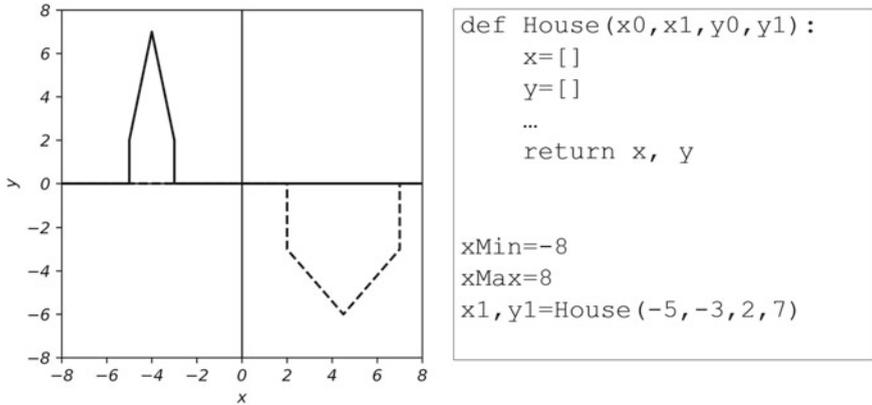


Fig. 8.27 a (left) Two shapes produced by a function *House* partly shown in b (right)

20. Why does the 20-fold addition of 0.10 to -2.00 not yield the smooth value 0.00, but rather 6×10^{-16} ?⁴⁰
21. Which statement can be used to make the function *House*(x) zero for $x \leq -1$ and $x \geq 1$?
22. Write a user-defined spreadsheet function that produces a shape like the “House” in Fig. 8.26b!
23. Figure 8.27a shows two shapes produced with a Python function *House* partly displayed in Fig. 8.27b. Complete the code!

⁴⁰ The numbers are binary coded so that decimal “smooth” numbers are not binary “smooth”.

We simulate measurement experiments by assuming “true” (known) values of a quantity and masking them with different noise levels that enter as standard deviation into normally distributed random numbers. Then, we “forget” the true values and estimate the measurand’s now unknown value from the noisy signal. ψ *We know everything and play stupid.* This way, the students should get an impression of how unreliable measurements can be, learn to indicate the reliability, and understand how measurement uncertainties propagate. The importance of t statistics, connecting confidence interval and confidence level (or error range and error probability), is illustrated in several exercises.

9.1 Introduction: We Know Everything and Play Stupid

Solutions of Exercises 9.2 (Excel), 9.3 (Python), 9.4 (Excel), and 9.5 (Python) can be found at the internet address: go.sn.pub/E3k8Ps.

- ▶ **Alac** Statistics? That’s all Greek to me.
- ▶ **Mag** Ψ *If in doubt, count!*
 Ψ *Calculate with variances, report the standard error!*

A simple measurement?

We shall determine the mass of a thin film on a glass substrate by weighing the substrate before and after coating. Estimating the mass is straightforward. However, stating a confidence interval requires this whole chapter, with t statistics and error propagation.

How should the poor student learn statistics?

Many students find it challenging to understand statistical statements. They are confronted with mathematically accurate but awkward-looking formulations and have to work according to strict rules.

In this chapter, the rules for statistical evaluation are not derived mathematically, but illustrated through simulations, particularly by *multiple tests for hit rates*. By repeating a noisy measurement series several times, we determine how often the standard error of the series's mean value captures the true value to specify the confidence level.

We know everything and play stupid

A writer of crime novels thinks up a criminal case, blurs the traces, and lets the detective reconstruct the case. So do we, by specifying the “true” values ourselves, turning them into measured values by adding noise, and then evaluating the noisy data. Finally, we report the measurement result with specified uncertainty. The true value may lie outside the confidence interval (the “error range” in laboratory jargon) of the measurement series' mean value. However, since we know the “true” value, we notice this and can observe regularities for our error.

Sherlock Holmes or Inspector G. Lestrade?

Some crime novel authors who want to be well received by the reader let the sharp-witted gentleman detective (Sherlock Holmes) or the curious lady detective (Miss Marple) appear alongside police officers, who stubbornly follow the rules and come to false results.

- ▶ **Tim** To which type does our evaluation belong?
- ▶ **Alac** To the smart detective, I hope.
- ▶ **Mag** No, unfortunately not! We must proceed strictly according to the rules. After all, intuition is often misleading when it comes to statistical issues. To get the best of both worlds, we follow the broom rule: *Ψ Trust (your intuition), but verify!*

Standard formulas

We apply the following formulas for a set of n measurements $x = \{x_i\}$.

The *mean* x_m of the set:

$$x_m = \frac{\sum_{i=1}^n x_i}{n} \quad (9.1)$$

The *variance* var_x of a set $x = \{x_i\}$:

$$var_x = \left(\sum_{i=1}^n (x_i - x_m)^2 \right) \cdot \frac{1}{n - n_p}, \quad n_p = 1 \text{ in this chapter} \quad (9.2)$$

where n_p is the number of parameters estimated from the data set. In this chapter, $n_p = 1$, because only one parameter, the mean, is estimated. In Chap. 10, trendlines are calculated and n_p is equal to the number of their parameters.

The *standard deviation* of the set:

$$x_{Sd} = \sqrt{\text{var}_x} \quad (9.3)$$

The *standard error* of the mean x_m of the set:

$$x_{Se} = \frac{x_{Sd}}{\sqrt{n}} \quad (9.4)$$

Standard error range

The standard error has the same physical unit as the elements of the sample and their mean value. Therefore, the mean x_m plus-minus the standard error x_{Se} is often reported as the result of a series of measurements:

$$\text{Standard error range} = x_m \pm x_{Se} \quad (9.5)$$

The standard error x_{Se} is calculated straightforward from Eqs. 9.2 (variance) and 9.3 (standard error).

Not to be forgotten: in your report, you also have to state the number of measurements. If at least eight measurements have been made, the following broom rule applies:

Ψ Two within and one out of (the standard error range).

The standard error range captures the true value of the measured quantity with a probability of about 2/3.

C-spec error

For statistically more precise reasoning, we have to connect confidence level and confidence interval by Student's¹ t statistics. We specify:

$$C - \text{spec error range} = x_m \pm t \cdot x_{Se} \quad (9.6)$$

For example, with eight measurements, the error range $x_m \pm 2.4x_{Se}$ captures the true value of the measurand with a probability of 95%. The interval $[x_m - t x_{Se}, x_m + t x_{Se}]$ is called the *confidence interval* (or *error range*) of the mean value x_m and the associated probability (95% in the example) *confidence level* C . The complementary probability (5% in the example) is called the *error probability* E :

$$C + E = 100\%$$

¹ Student is the pseudonym of W.S. Gosset.

We call the error for a well-specified confidence level *C-spec error* (“confidence-specified error”).

The parameter determining the *t* distribution is the degree of freedom *dof* defined as

$$dof = n - n_p \quad (9.7)$$

where *n* is the number of measurements and *n_p* the number of parameters estimated from the data series. In the exercises in this chapter, *n_p* is 1 because only 1 parameter, the mean, is estimated from the data series. In the next chapter, coefficients of trend lines are estimated from the data and *n_p* > 1.

Multiple tests for obtaining hit rates

Statistical laws specify error probabilities for hypotheses. In this chapter’s exercises, the hypothesis is that the estimated error range does not capture the true value. We do not derive such laws from axioms, but apply *multiple tests for hit rates* instead. They repeat statistical experiments multiple times and check whether the empirical hit or miss rates are compatible with the corresponding theoretical probabilities.

► **Tim** If this is not the case, then we have falsified the theoretical assumption.

► **Mag** Or have made a programming error. More about that later.

Simulation-based *t* adaptation

Many statistical laws are based on assumptions that are often not justified in our exercises or in real-life experiments, e.g., because the number of measurements is too small or the noise is too large. In such cases, the formally calculated error ranges may not exhibit the expected confidence level, e.g., in Exercise 9.9 where error propagation plays a role. In Exercise 9.8, we present a method to get C-spec errors, by adapting the *t*-factor so that a hit rate corresponding to a pre-specified confidence level is achieved. This method is suited also for real-world experiments.

Ψ Calculate with variances, report the C-spec error!

Mathematical formulas concerning confidence are formulated for variances. Statistical rules are based on properties of variances. We calculate with them to grasp the basic mathematical dependencies. For the final result, we strive to report the C-spec error because it is related to a confidence level and has the same physical unit as the measurand. In this way, we simulate the error propagation in sums, products and powers (Exercises 9.7 and 9.8).

Broom rules for measurements

We start with an exercise on a simple experiment, weighing a glass substrate, and end with a more challenging one, determining the mass of a thin film on such a glass

substrate. In between, we have to improve our understanding of statistics with the following broom rules:

Ψ *Mostly, but not always.* No statistical statement is 100% certain.

► **Mag** This is the fundamental rule of statistical reasoning. More rules:

Ψ Always round to *relevant* digits; to convey the result concisely! (Exercises 9.2 and 9.3).

Ψ *Twice as good with four times the effort.* The measurement inaccuracy is halved if measurements are made four times as often (Exercise 9.4).

Ψ *Two within and one out of.* In one-third of all tests, the measurand's true value is outside of the standard error range if at least 8 measurements have been made (Exercise 9.5).

Ψ *Bad makes good even better, usually, but not always.* Even measurement results with a relatively large statistical uncertainty may improve the overall result by entering a weighted mean (Exercise 9.6).

9.2 Weighing a Glass Substrate

The weighing process is simulated by adding noise to the glass substrate's true weight and estimating the weight from the noisy data. We determine the standard error of the result and round the result to the relevant number of digits.

9.2.1 Discussion on the Accuracy of a Balance

► **Mag** We are going to determine the weight of a glass substrate with a typical weight of 1 g. How precise is this procedure if the measuring precision of the balance is 1 mg?

► **Alac** A simple calculation: the substrate's mass is about $1 \text{ g} \pm 1 \text{ mg}$, in other words, it can be determined with a relative accuracy of 1 per mill.

► **Mag** Be careful! The balance's specification only states that the mass is displayed in grams with 3 digits after the decimal point. However, the accuracy of the measurement may be lower than the precision of the balance's display, for example, due to ventilation or building vibrations.

► **Tim** But we cannot take such influences into account, because they are random and out of our control.

► **Mag** That's a good point, because when they are simply random, their influence on the weighing accuracy can be reduced if we repeat the measurement several

times. With these multiple values, we can also specify the uncertainty of the result, in addition to the value of the mass.

► **Tim** I've experienced that before. In the physics lab course for beginners, we always have to report all results with measurement errors.

► **Alac** It's quite simple. *We already know the true values from our fellow students* in previous semesters, thus we only need to adjust the error range for our measured values so that the true value lies within the error bars. In this way, we avoid annoying questions from the supervisors.

► **Mag** That is similar to, but not exactly like, the way that we will do it in this exercise. As we generate the measurement data ourselves, we know the true values in advance, but then estimate them again from noisy data. However (!), we rigorously (!) calculate the standard error according to the rules.

► **Tim** Textbooks on measuring theory emphasize “confidence interval” and “confidence level”.

► **Mag** We will learn the meaning of these terms later in Exercises 9.4 and 9.5, in which we will show that if 9 groups present their lab results with standard error ranges capturing the true value, 3 will probably have cheated.

9.2.2 Data Structure and Nomenclature

m_S	true mass of the substrate
m_{Ns}	measurement noise (standard deviation of a normal distribution)
dsp	display precision of the scales
m_X	measurement series
n	number of measurements in m_X
m_M	mass of the substrate estimated from the measurements
m_{Se}	standard error of m_M .

9.2.3 Excel

Fig. 9.1 (S) presents an EXCEL solution, with the parameter specification in **a** and the process simulation in **b**. The raw data are generated in E6:E12 with normally distributed noise obtained with `NORMINV(RAND();0;1)` (see Exercise 8.5). They are transformed into the balance display, our measurement data, in column F. In G6:J6, the data are evaluated with the preliminary result for the estimated mass:

$$m_S (esti.) = (m_M \pm m_{Se})$$

$$m_s (esti.) = (0.995 \pm 0.022)g$$

	A	B	C	D				E	F	G	H	I	J	K
1	mS	1 g	"true" mass of the substrate											
2	dsp	3	display accuracy of the balance, typical display: 1.001 g											
3	mNs	0.05 g	noise of the weighing process											
4	$=mS+mNs*NORM.INV(RAND();0;1)$ $=ROUND(E7;dsp)$ $=AVERAGE(mX)$ $=STDEV.S(mX)$ $=COUNT(mX)$ $=mSd/SQRT(n)$													
5			mX	mM	mSd	n	mSe							
6	1.0062099		1.006	0.995	0.057	7	0.022							
7	1.0303005		1.030											
12	1.0878611		1.088											

Fig. 9.1 (S) Weighing a glass substrate. **a** (top) Specifications. **b** (bottom) Simulation of the process

Table 9.1 Weighing the mass of a substrate

1	mS=1	# [g]										
2	dsp=3	#Digits displayed										
3	mNs=0.05	# [g] Noise										
4	n=7	#Number of measurements										
5	mX=np.round(mS+mNs*np.random.randn(n), 3)											
mX	[1.076 1.050 1.049 0.990 1.010 0.993 1.018]											
6	mM=np.mean(mX)					mM 1.027						
7	mSd=np.std(mX)					mSd 0.030						
8	mSe=mSd/np.sqrt(n) #Standard error					mSe 0.011						

still to be rounded to the relevant digits:

$$m_s(\text{rounded}) = (1.00 \pm 0.02)\text{g}.$$

First, the standard error is rounded to one digit, and then, the mean value is rounded to the same number of decimal places as the standard error.

9.2.4 Python

The Python solution in Table 9.1 is straightforward.

Question

How do you report the results of Table 9.1 sensibly rounded?²

² 1.027 ± 0.011 becomes 1.03 ± 0.01 .

9.3 A Procedure for Rounding to Relevant Digits

A formula network and a Python function are presented that round a measurement result to the relevant number of digits determined by the standard error. The equations rely on logarithms and powers.

9.3.1 Numerical Evaluations

The numerical evaluation of measurement series generally results in numbers with many digits, e.g., $x_m = 0.008702$ g and $x_{Se} = 0.000602$ g. As we have learned in Exercise 9.2, the final result should be $(0.87 \pm 0.06) \times 10^{-2}$ g or (8.7 ± 0.0) mg. We will obtain such results with formulas. To achieve this, we have to calculate with logarithms and powers of ten and link text and numbers.

9.3.2 Spreadsheet Calculation

The method is shown as a formula network in Fig. 9.2 (S), based on the example of Exercise 9.2, in which the mass of a glass substrate is determined. The estimated mean value x_M and its standard error x_{Se} are specified with 3 digits, the display accuracy of the balance.

The value of x_{Se} ($0.022 = 2.22 \times 10^{-2}$) is broken down into power of ten ($n_{Se} = -2$) and first digit ($x_{SeR} = 2$) and reproduced as x_{SeRR} in decimal form with only one non-zero digit. The value for x_M is transformed into an integer x_{Mr} . Thus, the number of relevant digits corresponds to the precision of x_{SeR} , and is reproduced in decimal form with the reduced number of digits. The final result displayed in D7 is obtained with the formula in D8, concatenating text and variables.

Another variant, reporting the final result in exponential form, i.e., $(9.8 \pm 0.2)E-1$ g, is shown in Fig. 9.3 (S). The value of x_M is transformed into a number x_{MrP} greater or equal to 1 and smaller than 10 and then rounded to the first non-zero digit of x_{Se} .

	A	B	C	D	E	F	G
1	Name	Weight		Power of xSe	nSe	-2	=INT(LOG10(xSe))
2	xM	0.977		First digit of xSe	xSeR	2	=ROUND(xSe*10^(-nSe);0)
3	xSe	0.022		Reduced xSe	xSeRR	0.02	=ROUND(xSeR*10^nSe;ABS(nSe))
4	Unit	g			xMr	98	=ROUND(xM*10^(-nSe);0)
5				Reduced xM	xMrr	0.98	=ROUND(xMr*10^nSe;ABS(nSe))
6							
7				Weight = (0.98 ± 0.02) g			
8				=Name&" = ("&xMrr&" ± "&xSeRR&") "&Unit			

Fig. 9.2 (S) The final result is obtained by rounding with formulas using logarithms

	A	B	C	D	E	F	G
10	Name	Weight		Power of xM	nM	-1	=INT(LOG10(xM))
11	xM	0.977		Reduced xM	xMrP	9.77	=xM/(10^nM)
12	xSe	0.022		Power of xSe	nSe	-2	=INT(LOG10(xSe))
13	Unit	g		Reduced xSe	xSeRP	2.2	=xSe/(10^nSe)
14				Rounded to relevant	xMred	9.8	=ROUND(xMrP;nM-nSe)
15				Rounded to relevant	xSeRed	0.2	=ROUND(xSeRP;0)*10^(nSe-nM)
16							
17				Weight = (9.8 ± 0.2) E-1 g			
18				=Name&" = ("&xMred&" ± "&xSeRed&") E"&nR&" "&Unit			

Fig. 9.3 (S) Same as Fig. 9.2 (S); however, separating the power in the result

Table 9.2 Function for returning the final result rounded to the relevant digits

```

1 def FinRes (Name, xM, xSe, Unit) :
2     #Power of the standard error:
3     n=int(np.floor(np.log10(xSe)))
4     #First digit of xSe:
5     xSeR=np.round(xSe*10**n, 0)
6     #Rounding to the certain digit:
7     xSeRR=np.round(xSeR*10**n, np.abs(n))
8     #Rounding xM to the certain digits:
9     xMr=np.round(xM*10**n, 0)
10    xMrr=np.round(xMr*10**n, np.abs(n))
11    return str(Name) + '=' + str(xMrr) + "±" \
12           + str(xSeRR) + Unit
    
```

9.3.3 Python Function

A user-defined function, performing the calculation of Fig. 9.2 (S), is introduced in Table 9.2 and applied in Table 9.3 in which two series of (xM, xSe) pairs are evaluated to check the validity of the formulas implemented in *FinRes*.

9.3.4 VBA Function

The rounding can also be performed with the user-defined VBA function *FinRes* in Fig. 9.4a (P), which reads-in the name, the value, and the uncertainty of the measurement result, and outputs the rounded result as shown in Fig. 9.4b. If this function is inserted into a VBA project module for user-defined functions (in our case, VBA PROJECT (*Dieters.functions.XLAM*, Sect. 4.9.1), then it can be called in any EXCEL file.

Table 9.3 a (top) Continuation; applying the program in Table 9.2; b (bottom) Results

<pre> 1 xM=[100.234, 1.234, -2.334,-0.004, 10000.1233] 2 xSe= [0.5, 0.0002, 0.02, 0.001, 0.0004] 3 for i in range(len(xM)): 4 outpl=FinRes("a"+str(i),xM[i],xSe[i]," mg") 5 print(outpl) 6 print("\n") #Second series 7 xM=[1234.345,-12345.678,2.383,-0.0991297, 0.000930] 8 xSe=[35.023, 34.56, 0.01, 0.00223046,0.000017] </pre>	
a0=100.2±0.5 mg	a6=1230.0±40.0 mg
a1=1.234±0.0002 mg	a7=-12350.0±30.0 mg
a2=-2.33±0.02 mg	a8=2.38±0.01 mg
a3=-0.004±0.001 mg	a9=-0.099±0.002 mg
a4=10000.1233±0.0004 m	a10=0.00093±2e-05 mg

<pre> 1 Function FinRes(Name, xM, xSe, Unit) 2 'Input: Name of the variable, mean, uncertainty, physical unit 3 n = Int(Log(xSe) / Log(10)) 'power of the uncertainty 4 xSeR = Round(xSe * 10 ^ -n, 0) 'first digit of the uncertainty 5 xSeRR = xSeR * 10 ^ n 'rounding to the certain digit 6 xMr = Round(xM * 10 ^ -n, 0) 'rounding the measured value 7 xMrr = Round(xMr * 10 ^ n, Abs(n)) 8 FinRes = Name & "=" & xMrr & "±" & xSeRR & Unit 9 End Function </pre>		<pre> 1 =FinRes(A4;B4;C4;D4) 2 a0=100,2±0,5 mg 3 a1=1,234±0,0002 mg 4 a2=-2,33±0,02 mg 5 a3=-0,004±0,001 mg 6 a4=10000,1233±0,0004 mg 7 a5=-234,6±0,3 mg 8 a6=12350±40 mg 9 a7=-12350±30 mg 10 a8=2,38±0,01 mg 11 a9=-0,099±0,002 mg 12 a10=0,0093±0,00002 mg </pre>																													
<table border="1"> <thead> <tr> <th></th> <th>A</th> <th>B</th> <th>C</th> <th>D</th> <th>E</th> </tr> </thead> <tbody> <tr> <td>2</td> <td></td> <td></td> <td></td> <td></td> <td>=FinRes(A4;B4;C4;D4)</td> </tr> <tr> <td>3</td> <td>a0</td> <td>100.234</td> <td>0.5 mg</td> <td></td> <td>a0=100,2±0,5 mg</td> </tr> <tr> <td>4</td> <td>a1</td> <td>1.234</td> <td>0.0002 mg</td> <td></td> <td>a1=1,234±0,0002 mg</td> </tr> <tr> <td>5</td> <td>a2</td> <td>-2.334</td> <td>0.02 mg</td> <td></td> <td>a2=-2,33±0,02 mg</td> </tr> </tbody> </table>		A	B	C	D	E	2					=FinRes(A4;B4;C4;D4)	3	a0	100.234	0.5 mg		a0=100,2±0,5 mg	4	a1	1.234	0.0002 mg		a1=1,234±0,0002 mg	5	a2	-2.334	0.02 mg		a2=-2,33±0,02 mg	
	A	B	C	D	E																										
2					=FinRes(A4;B4;C4;D4)																										
3	a0	100.234	0.5 mg		a0=100,2±0,5 mg																										
4	a1	1.234	0.0002 mg		a1=1,234±0,0002 mg																										
5	a2	-2.334	0.02 mg		a2=-2,33±0,02 mg																										

Fig. 9.4 a (left, P) The user-defined function *FinRes* (top) reads in the name and the value of a measured quantity (columns A and B in the spreadsheet), as well as its measurement error (column C), and (bottom) outputs the rounded measurement result as text (column E). The spreadsheet has to be treated with a main program that calls *FinRes*. b (right) Ten results of the FUNCTION *FinRes*

9.4 Increasing the Measuring Accuracy Through Repetition

We illustrate the meaning of the standard error with repetition procedures to get the hit rate, i.e., how often the error range captures the true value. The more often a quantity is measured, the more accurately its value can be determined. If the same measurement setup is always used, the standard error of the mean value of the measurement series is inversely proportional to the root of the number of measurements. *Ψ Twice as good with four times the effort.*

9.4.1 Standard Deviation and Standard Error of the Mean Value of a Measurement Series

How can we halve the standard error?

- ▶ **Alac** If we want to be twice as good, we just have to measure twice as often.
- ▶ **Mag** No, keep the following broom rule in mind and study the next section:
Ψ Twice as good with four times the effort.
- ▶ **Tim** I remember the reason for this. It is the variance of the mean value of a measurement series that is inversely proportional to the number of measurements. The standard error is the square root of the variance.
- ▶ **Mag** One more hint: All mathematically justified theorems make statements about variances. To estimate the measurement error, however, the standard error with the same physical unit as the mean must be quoted:
Ψ Calculate with variances, report the C-spec error!

In this section, we learn about the quantities: variance var_X of a set of values $x = \{x_i\}$, standard deviation x_{Sd} of the data in the set, and standard error x_{Se} of the mean x_m of the set.

The standard error x_{Se} of the mean value x_m of a set of measured values, used to indicate the measurement result's uncertainty, is smaller than the standard deviation x_{Sd} of the set, measuring the scatter in the set. The standard deviation x_{Sd} is, in principle, independent of the number n of measurements, while the standard error x_{Se} decreases with increasing square root of n .

Visual estimate

Question

How big do you estimate the uncertainty of the mean value in Fig. 9.5a to be?³

By what factor is the standard error of the mean smaller than the standard deviation of the set of measurements?⁴

Within which range do you expect a 10th measured point in Fig. 9.5?⁵

Calculated error range

Figure 9.6 displays measured data with calculated means (vertical bar), standard deviations of the set of measured values (horizontal square brackets above), and standard errors of the mean (horizontal square brackets below).

³ To check your judgement, have a look at Fig. 9.6!

⁴ By the factor $1/\sqrt{n}$, where n is the number of measurements, as you will learn in the course of this exercise.

⁵ The 10th measured point should be within the standard deviation of about 1 with a probability of about 2/3.

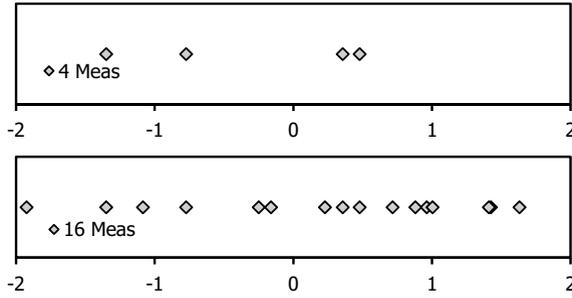


Fig. 9.5 Estimate! Estimate mean value, standard deviation, and standard error of **a** (top) a series of 4 measurements and **b** (bottom) a series of 16 measurements! The standard deviation for both series is about 1

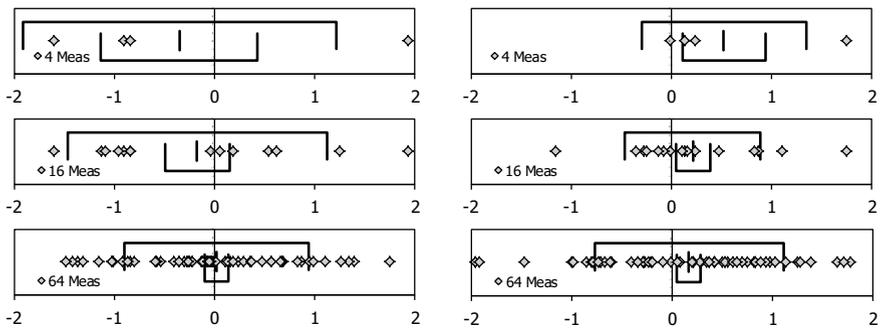


Fig. 9.6 Results for $n = 4, 16,$ and 64 measurements of a physical quantity whose true value is 0 ; mean value x_m indicated by vertical lines, scatter $x_m \pm x_{sd}$ of the measurement series by square brackets above, and confidence interval (error range) $x_m \pm x_{se}$ of the mean value as square brackets below; the true value 0.00 is **a** (left) within the standard error range of the mean value or **b** (right) outside of the standard error range of the mean value

In the analysis and discussion of measurement series, we must clearly distinguish between

- the standard deviation x_{sd} of the data set (Eq. 9.3) and
- the standard error x_{se} of the mean value x_m (Eq. 9.4).

The standard deviation of the series characterizes the scatter of the measured values within the measurement series. A new measurement is expected to lie approximately within this range, “approximately” meaning:

Ψ Two within and one out of (the standard deviation).⁶

⁶ Namely, with a probability of $2/3$ within plus or minus the standard deviation from the mean.

The *mean value* x_m of all measured values can be determined by visual judgment more precisely than $\pm x_{SD}$, as is visible in the diagrams in Figs. 9.5 and 9.6. According to theory, the *standard error* of the mean value x_{Se} is equal to the standard deviation of the series x_{SD} divided by the square root of the number n of measurements, Eq. 9.4.

The corresponding range

$$[x_m - x_{Se}, x_m + x_{Se}] \quad (9.8)$$

called the standard error range, is indicated in the diagrams by bottom square brackets.

In the next section, we examine the meaning of the error range in more detail by determining hit rates.

- ▶ **Mag** What is the informative value of the standard error?
- ▶ **Tim** The standard error has the same physical dimension as the measured value. Therefore, it is possible to specify the measurement uncertainty with plus-minus the standard error.
- ▶ **Mag** Is the true value always within the standard error?
- ▶ **Alac** No, only if we've done everything right.
- ▶ **Tim** I've learned that this is only true in about 68% of cases, to be remembered with:
 Ψ Two within and one out of.
- ▶ **Alac** Is that to say that the true value is outside of the standard error limits for every third experiment? There was never any trouble like this in our introductory lab course.
- ▶ **Mag** If the true value for nine lab groups lies within the standard error of the measurement $\pm x_{se}$, then three groups have cheated, I suspect.
- ▶ **Tim** We have also learned that the broom rule applies only if the same quantity is measured at least eight times. We never measured that often in the beginner's lab course. What is the error specification good for, then?
- ▶ **Mag** We will learn about that in this exercise by determining hit rates. Anyway, if you specify the number n of measurements, together with x_m and x_{Se} , you can make well-defined statements with t statistics, addressed in Exercise 9.5.

4, 16, or 64 measurements of the same quantity

Let the true value of a quantity be 0 and the standard deviation of the measurement process be 1. We simulate this process with a Gaussian generator with a mean value of 0.00 and a standard deviation of 1.00 and determine the mean value x_m , standard deviation x_{SD} , and standard error x_{SE} for measurement series (samples) with different numbers of measurements. The results of the simulation are presented in Fig. 9.6.

In the first row of Fig. 9.6, the physical quantity was measured four times, in the second row, 16 times, and in the third row, 64 times. In the left subpictures, the mean value's error ranges include the true mean; in the right subpictures, they do not.

Standard deviation of the measurement series

The measured standard deviation x_{SD} is approximately the same for all measurement series, independent of the number of measurements. So, if an additional measurement is made, the expected deviation of this new measurement from the mean value is independent of how often the measurement has been made before. The dispersion of the data set does not depend on the number of data.

How precisely can the mean value of a measurement series be determined?

In Fig. 9.6, two measurement series are plotted for each number of measurements. The mean value fluctuates more strongly for the series with a smaller number (visible here when comparing the left and right subfigures). The theory states that the expected measurement error for the mean value of a measurement series is the standard deviation of this series divided by the root of the number of measurements (Eq. 9.4). The error range (also called the confidence interval) of the mean value is also included in the diagrams mentioned. It corresponds approximately to the range in which we would locate the mean value with the eye.

► **Tim** Are we really twice as good with four times the effort?

► **Mag** The formula only states that the *standard error* of the mean calculated according to the rules decreases with $1/\sqrt{n}$, narrowing the confidence interval ("error range") correspondingly. We still have to know whether the confidence level ("hit rate") changes as well.

- – The standard deviation of a measurement series is in principle independent of the number of measurements; it fluctuates only by chance.
- The standard error of the mean of the measurement series decreases with the inverse of the square root of n , the number of measurements.
- The probability that the standard error range captures the true value depends on the number of measurements and increases with increasing n up to 68%.

Hit rate by repetition of the statistical experiment

A simulation repeats the standard error range calculation and determines how often $x_m \pm x_{Se}$ captures the true value. The “hit rates” are:

- 0.60 (0.61) for 4 measurements,
 - 0.66 (0.67) for 16 measurements,
 - 0.68 (0.68) for 64 measurements,
- obtained with Python (EXCEL) in the following sections.

9.4.2 Data Structure and Nomenclature

x_{meas}	results of 64 measurements, true value = 0
x_{m4m}	mean value of 4 measurements
x_{Sd4m}	standard deviation of 4 measurements
x_{Se4m}	standard error of the mean of 4 measurements
in_{4m}	true, if the error range captures the true value
$within4m$	counts the in_{4M} and correspondingly for 16 and 64 measurements.
in_M	3 values, [in_{4m} , in_{16m} , in_{64m}].

9.4.3 Python Program

Hit rate by repetition of the statistical experiment

In the diagrams of Fig. 9.6b, the true value of the measured quantity (namely, 0.0) is outside of the standard error range. With a rep-log procedure, we examine how often this happens on average. The function *WithinSe* in Table 9.4 generates an array of 64 normally distributed random numbers, calculates, in lines 4 to 6, the characteristic parameters for the first 4 numbers of this array, and determines, in line 7, whether the standard error range captures the true value ($in4m$ is True or False). The

Table 9.4 Does the standard error range capture the true value? Only the statements for 4 measurements are reported; those for 16 and 64 measurements are the same, but with 16 or 64 instead of 4

```

1  def WithinSe():
2      x_meas = npr.randn(64)
3
4      xm4m = np.mean(x_meas[0:4]) #Entries 0, 1, 2, 3
5      xSd4m = np.std(x_meas[0:4], ddof=1)
6      xSe4m = xSd4m/np.sqrt(4)
7      in4m = xm4m- xSe4m < 0 < xm4m+xSe4m
           #0 is the true value
.....
18     return in4m, in16m, in64m

```

Table 9.5 Rep-log procedure for calculating hit rates, *WithinSe* from Table 9.4

1	<code>within4m =0</code>			
2	<code>within16m=0</code>			
3	<code>within64m=0</code>			
4	<code>Ntrials=10000</code>			
5	<code>for i in range(Ntrials):</code>			
6	<code> inx=WithinSe()</code>			
7	<code> within4m+=inx[0]</code>	<code>within4m</code>	<code>6018</code>	<code>hit rate 0.60</code>
8	<code> within16m+=inx[1]</code>	<code>within16m</code>	<code>6632</code>	<code>hit rate 0.66</code>
9	<code> within64m+=inx[2]</code>	<code>within64m</code>	<code>6759</code>	<code>hit rate 0.68</code>

instructions for the first 16 and first 64 measurements are analogous, resulting in *in16m* and *in64m*, which are also returned in line 18.

In the `numpy` function `np.std`, a parameter `ddof` must be specified, meaning *Delta Degrees Of Freedom*. The divisor used in calculating the variance in Eq. 9.2 is $n - ddof$, where n is the number of data points. By default, `ddof` in `np.std` is zero. If we determine only the mean of a data series, as we do in this chapter, $ddof = n_p = 1$.

In the main program in Table 9.5, we call *WithinSe* several times (here, $N_{\text{trials}} = 10,000$) and increment the variables *within4m* and the like by 1 if the output of *WithinSe* is true. This is possible because the logical values are coded as binary numbers, 1 for *True* and 0 for *False*. We see from the bottom cell output that the hit rate increases with the number of measurements, from 0.60 for 4 measurements to 0.68 for 64.

Theoretically, the confidence interval for $\pm x_{se}$ is 68.3% when the number of measurements approaches infinity. In other words: in more than 30% of the cases, the “true” value is also theoretically outside $\pm x_{se}$. We simplify to the broom rule: *Ψ Two within and one out of (the standard error range)*. It applies if the measurement is repeated at least 8 times. If measurements are taken less often, the hit rate is lower. For 4 measurements, it is only 60%, according to the bottom cell in Table 9.5.

9.4.4 Spreadsheet Layout for This Task

The spreadsheet layout corresponding to the `Python` program in Table 9.4 is shown in Fig. 9.7 (S). Columns A to H reproduce the calculation of the `Python` function *WithinSe* with the result in column H.

The role of the main program in `Python` is played in `EXCEL` by the VBA procedure in Fig. 9.8 (P), taking the logical values in H3:H5 as input and, if *True*, incrementing the counts in I3:I5 individually.

	A	B	C	D	E	F	G	H	I	J	K	L
1	=NORM.INV(RAND();0;1)											
2	x_meas	n	xm_m	xSd_m	xSe_m	in_m		within_m	hitrate	Ntrials		
3	2.04	4	0.87	1.22	0.61	4 Meas	FALSE	6116	0.612	10000		
4	0.30	16	0.12	0.96	0.24	16 Meas	TRUE	6738	0.674			
5	1.71	64	0.02	0.91	0.11	64 Meas	TRUE	6840	0.684			
66	0.02											

Fig. 9.7 (S) 64 random measurements x_{meas} are generated in column A, while subsets of these are evaluated in columns D to H. The logical values in column H are read by SUB *Protoc3* in Fig. 9.8 (P) and summed in column I after a 10,000-fold generation of new measurement series

1	Sub <i>Protoc3</i> ()	12
2	Ntrials = Range("K3")	13
3	Range("I3:I5").ClearContents	14
4	For i = 1 To Ntrials	15
5	Application.Calculation = xlCalculationManual	16
6	If Range("H3") = True Then Range("I3") = Range("I3") + 1	17
7	If Range("H4") = True Then Range("I4") = Range("I4") + 1	18
8	If Range("H5") = True Then Range("I5") = Range("I5") + 1	19
9	Application.Calculation = xlCalculationAutomatic	20
10	Next i	21
11	End Sub	22

Fig. 9.8 (P) Rep-log procedure for incrementing the number of hits in I3:I5 of Fig. 9.7 (S); if a value in H3:H5 is True, the corresponding cell in column I is incremented by 1

Questions

- concerning Fig. 9.7 (S)
 - How does the “true” value of the measurand enter the simulation of the measurement process in A3:A66?⁷
 - What is the formula in H5?⁸

9.4.5 How to Report a Measurement Result

► **Tim** People often say that our rule Ψ *Two within and one out of* only applies if measurements are taken more than seven times. We never measure that often in our lab course, at most, four times.

⁷ As mean of a normal distribution in column H generated with NORM.INV (RANDOM(), *Mean*, *StDev*). Here, *Mean* = 0 and *StDev* = 1.
⁸ See formula printed in H1; the expression is *True* if the error range does capture the true value. The formula in H5 processes entries in row 5 of xm_m and xSe_m .

- **Alac** That doesn't matter. Even if you measure a quantity only twice, you can apply the formulas for mean value and standard error. That's enough.
- **Mag** Not quite. You have to specify the number of measurements in addition to the mean value and the standard error. An example of a complete measurement protocol: The acceleration due to gravity in our laboratory was measured twice with a pendulum, resulting in an average value of 9.8 m/s^2 with a standard error of 0.2 m/s^2 .
- **Tim** Should we report the details of the experimental setup as well? For example: How exactly we determined the length of the pendulum and the period of oscillation.
- **Mag** Of course, to allow for conclusions about systematic errors. The three aforementioned statistical specifications are enough to make statistically relevant statements. E.g., for the two measurements referred to by Alac, the calculated standard error must be doubled to achieve a measurement uncertainty according to Ψ *Two within and one out of*.⁹

9.5 The t Statistics Connects Confidence Interval with Confidence Level

The t -value of Student's t distribution relates an extended confidence interval t times the standard error to a confidence level (or an error probability). For at least eight measurements, our rule for the standard error range Ψ *Two within and one out of* (the standard error range) applies.

9.5.1 Student's t Distribution

Practical handling

The following list explains the terms that play a role in t statistics:

- *Standard error range* $x_m \pm x_{Se}$ with the standard error calculated from the standard deviation of the measurement series x_{Sd} as $x_{Se} = \frac{x_{Sd}}{\sqrt{n}}$.
- *Degree of freedom* $dof = n - n_p$ with n being the number of measurements, and n_p the number of parameters estimated from the measurement series. In this chapter, $n_p = 1$, because only the mean x_m is estimated.

⁹ The degree of freedom for 2 measurements is $dof = 1$ and the t value for this is $t = 1.84$ for a probability of error of 38% (see Exercise 9.5). For an error probability of 5%, the t value is 12.7.

- *C-spec error range* $x_m \pm t_C \cdot x_{Se}$, standard error multiplied by a factor t_C determined by a pre-specified confidence level C or error probability E and the degree of freedom dof .
- *Confidence level* $C(t, dof)$, corresponds to the hit rate in our statistical experiments. The error probability has the complementary value: $E(t, dof) = 1 - C(t, dof)$.

Error probability, confidence level, and *t*-value are obtained with the following functions, with *dof* being the degree of freedom:

```
EXCEL      E(T,DOF) = T.DIST.2T(T;DOF)
           C(T,DOF)= 1- T.DIST.2T(T;DOF)
           T(E,DOF) = T.INV.2T(E,DOF)
```

```
Python    E(t, dof) = (1-sct.t.cdf(t, df=dof)) * 2
           C(t, dof) = 2*sct.t.cdf(t, df=dof) - 1
           T(E, dof) = sct.t.ppf(1-E/2, df=dof)
```

In Fig. 9.9a, the error probability E is obtained for different degrees of freedom f and t values of 1 and 1.96. The t value for pre-specified E and dof is given in **b** (F:H).

For 4 measurements, the degree of freedom is 3, and a hit rate $1-0.391 \approx 0.61$ for $t = 1$ is expected. Actually, in Sect. 9.4.1, hit rates of 0.60 for the Python simulation and 0.61 for the EXCEL one come close to this expectation.

Mathematics of the *t* distribution

The entity t is defined as

$$t = (x_m - x_{True})/x_{Se} \tag{9.9}$$

	A	B	C	D	E	F	G	H	I
1									
2	dof	E			$= (E-1/3)^3$	Et	dofT		$= T.INV.2T(Et;dofT)$
3	100000	0.317	$= T.DIST.2T(1;dof)$	-0.05		0.317	100000	1.00	
4	7	0.351	$= T.DIST.2T(1;dof)$	0.05		0.317	7	1.08	
5	3	0.391	$= T.DIST.2T(1;dof)$	0.17		0.317	3	1.20	
6	100000	0.050	$= T.DIST.2T(1.96;dof)$			0.05	100000	1.96	
7	7	0.091	$= T.DIST.2T(1.96;dof)$			0.05	7	2.36	
8	3	0.145	$= T.DIST.2T(1.96;dof)$			0.05	3	3.18	

Fig. 9.9 a (left, A:D) Error probability E for t values 1 and 1.96, each for various degrees of freedom dof . **b** (right, F:H) t values for E_t values of 0.317 and 0.05, as well as various degrees of freedom dof_T

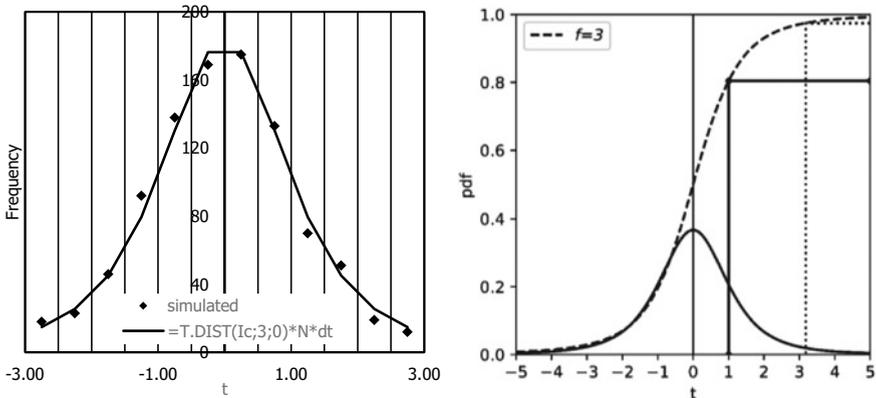


Fig. 9.10 **a** (left) Frequency of occurrence of t values obtained in a series of simulations with 4 measurements together with the theoretically predicted values for $dof = 3$; **b** (right) pdf and cdf (dashed line) of Student's t distribution for $dof = 3$ (dof is named f in the legend); the right-angled straight lines indicate $t.cdf(1, dof)$ and $t.ppf(0.975, dof)$ (dotted line)

describing the distance of the sample mean x_m to the true value x_{True} , divided by the standard error x_{Se} of the mean value.

In order to get an idea of t statistics, we:

- perform a simulation of a series of 4 measurements with a noise level between 0 and 1 from which a single value of t can be obtained,
- repeat the single experiment to get a representative set of t values, with each time a new noise level,
- and determine their frequencies of occurrence.

The result (diamonds) is shown in Fig. 9.10a together with the values predicted by the theoretical t distribution for $dof = 3$ (line).

Figure 9.10b displays the pdf and cdf of Student's t distribution for $dof = 3$; the right-angled straight lines indicate $t.cdf(1, dof)$ starting at $t = 1$ and $t.ppf(0.975, dof)$ (dotted line, starting at $cdf = 0.975$). We get.

$$\begin{aligned} \text{sct.t.cdf}(1, 3) &= 0.8045 \\ C(1, 3) &= 2 * \text{sct.t.cdf}(1, 3) - 1 = 0.609 \\ T(0.05, 3) &= \text{sct.t.ppf}(\{0.975, 3\}) = 3.18 \end{aligned}$$

We see that for $t = 1$, the cdf is about 0.8, i.e., in about 80% of the experiments a measurement result is below the upper limit of the error range; 20% are above. As the pdf is symmetric to $t = 0$, likewise 20% are below the lower limit of the error range so that the error probability is $E = 0.4$ and correspondingly $C(1, 3) \approx 0.6$.

For $dof = 1,000,000$, we get

$$\text{sct.t.cdf}(1, 1, 000, 000) = 0.841,$$

$$p = 0.16 \text{ that } x_{T_{\text{true}}} > x_m + 1 \cdot x_{Se}.$$

$$C(1, 1, 000, 000) = 2 * \text{sct.t.cdf}(1, 1, 000, 000) - 1 = 0.683.$$

$$E(1, 1, 000, 000) = (1 - 0.683) = 0.317,$$

$$p = 0.317 \text{ that } x_{T_{\text{true}}} \text{ is outside of the error range } x_m \pm 1 \cdot x_{Se}.$$

$$T(0.05, 1, 000, 000) = \text{Sct.t.ppf}(0.975, 1, 000, 000) = 1.96.$$

a confidence level of $C = 0.95$ (error probability $0.05 = 2 \cdot 0.025$) is obtained for an error range $x_m \pm 1.96 \cdot x_{Se}$.

Set-up of the simulations to determine C-spec errors

We are simulating measurement series according to Ψ *We know everything and play stupid* by generating normally distributed random numbers with mean = 0 and standard deviation = 1, and determining the experimental hit rate, for series comprising 2 to 16 measurements, within error ranges with t values that correspond to error probabilities 0.317 and 0.05.

Proven or not disproven? $8 = \infty$?

- ▶ **Tim** Summarizing: with the mean value and its standard deviation, and the number of measurements, statistically correct statements with confidence levels can be made.
- ▶ **Alac** Our simulation proves that.
- ▶ **Tim** No, our simulation simply gives us no reason to doubt that statement.
- ▶ **Alac** Your nitpicking's a pain in the neck.
- ▶ **Mag** But it is indispensable, because we don't do logical derivations.
- ▶ **Tim** Let's trust in statistical textbooks.
- ▶ **Alac** In the case of measurement series, we are content with a rough estimate: infinity already starts with 8 measurements.
- ▶ **Tim** $8 = \infty$? Isn't that twisting the facts?
- ▶ **Mag** Discuss!

Questions

concerning Fig. 9.9 (S) By what factor t should the standard deviation of 4 measurements be increased so that the true value is within the C-spec error range with a probability of 68.3%?¹⁰

What is the probability of error if, for eight measurements, the standard error is specified as the measurement uncertainty?¹¹

What is the error probability for four measurements if the standard error times 1.96 is used to specify the error range?¹²

9.5.2 Data Structure and Nomenclature

x_{True}	True value of a measurand
n	Number of measurements in a series
x	Series of n measurements
x_m	Mean value of the measurement series
x_{Se}	Standard error of x_m
dof	Degrees of freedom, $dof = n - 1$
C	Confidence level
E	Error probability, $E = 1 - C$
t	t -value (Student's) for the C-spec error range, determined by C and dof
x_{SeT}	x_{Se} multiplied with a t -value appropriate for C and E
n_T	Number of trials (repetitions of a statistical experiment)
$hitRate$	Counts how often the error range captures the true value
p_{Out}	$(n_T - hitRate)/n_T$, probability that the error range misses the true value.

9.5.3 Spreadsheet Calculation

In column A of Fig. 9.11 (S), 32 normally distributed random numbers (mean = 0, standard deviation = 1) are generated. Subsets of 2 to 32 of them are specified by cell range addresses in column C. Indirect addressing of these ranges is used to calculate their mean (in column D) and their C-spec error for $E = 0.317$ (in column F).

Questions

concerning Fig. 9.11 (S)

¹⁰ Figure 9.9 (S), the degree of freedom for four measurements is $dof = 3$, $t = 1.20$ in cell C5.

¹¹ Figure 9.9 (S), $dof = 7$, B4 = 35%.

¹² Figure 9.9 (S), $dof = 3$, B8 = 14.5%.

	A	B	C	D	E	F	G	H	I	J	K
1	$=NORM.INV(RAND();0;1)$ $=AVERAGE(INDIRECT(Rng))$ $=T.INV.2T(0.317;n-1)$ $=STDEV.S(INDIRECT(Rng))/SQRT(n)*t$ $=OR(xm<-xSeT;xm>xSeT)$ from Rep-Log procedur										
2	x	n	Rng	xm	t	xSeT	outSeT	pOut	100,000	t0.05	pOut0.05
3	-0.44	2	A3:A4	0.34	1.84	1.45	FALSE	0.317		12.7	0.049
4	1.13	4	A3:A6	0.73	1.20	0.60	TRUE	0.317		3.2	0.050
5	0.33	8	A3:A10	-0.19	1.08	0.58	FALSE	0.317		2.4	0.049
6	1.88	16	A3:A18	-0.02	1.04	0.30	FALSE	0.314		2.1	0.050
7	-2.97	32	A3:A34	-0.11	1.02	0.20	FALSE	0.318		2.0	0.049
34	0.35										

Fig. 9.11 (S) Evaluation of series from column A with 2 to 32 measurements; in column F, C-spec error ranges are listed for an error probability of $E = 0.317$. The formula in column G checks whether the true value (here, 0) lies outside of the error limits. In column H, the experimental error rate is recorded after 100,000 repetitions of the statistical spreadsheet experiment (*Multiple tests for hit rates*). The simulation is repeated for an error probability $E = p_{Out}$ of 0.05 (t values in column J), with the results reported in column K

Which numbers addressed in D6 are averaged?¹³

How is the argument of the spreadsheet function AVERAGE in column D constructed?¹⁴

Which cells does the statement in F4 refer to?¹⁵

In column E, the spreadsheet function $t = T.INV.2T(E;dof)$ reports the t value for an error probability of $E = 0.317$. The extension “.2T” indicates that deviations of x_m to both sides of the error range are taken into account.

In column F, the mean value’s standard error is multiplied by t , so that the error rate is expected to be $p_{Out} = E = 0.317$. Column G contains the logical expression $[=OR(x_m < -x_{SeT}; x_m > x_{SeT})]$, true if x_{mTrue} , namely, 0, is outside of the error range. In the example of Fig. 9.11 (S), this is the case (by chance) only for one measurement series. With a rep-log procedure, we perform the random experiment 100,000 times and get the error rates in column H. They are actually close to the expected 0.317.

In column J, t values that correspond to an expected error probability of $E = 0.05$ are calculated. Repeating the simulation with the corresponding C-spec error range leads to experimental error probabilities, column K, close to the theoretical value.

¹³ The mean value of the numbers in A3:A18 is calculated.

¹⁴ The argument of AVERAGE is defined using INDIRECT from the column range with name *Rng*. From *Rng*, the value in the same row is taken.

¹⁵ The statement in F4 (1) calculates the standard deviation of the subset A3:A6 (addressed with INDIRECT(RNG) , *Rng* in C4), (2) divides the result by \sqrt{n} (B4) to get the standard error, (3) multiplies the standard error with a t (E4) for $E = 0.317$ and $dof = n - 1 = 3$, appropriate for $n = 4$).

Table 9.6 **a** (top) Specification of arrays for taking up the results of simulations with t statistics; **b** (bottom) A single run of the simulation

```

1  nList=[2,4,8,16,32]
2  lnL=len(nList)
3  E=0.317
4  t=np.empty(lnL)          #t values for E
5  for n in range(lnL):
6      t[n]=sct.t.ppf(1-E/2,df=nList[n]-1)
7  out=np.zeros(lnL)
8  x=npr.randn(32)
9  xm=np.array([np.average(x[0:n]) for n in nList])
10 xSeT= np.array([np.std(x[0:n],ddof=1)/np.sqrt(n)
                  for n in nList])*t
11 outSeT=np.logical_or((xm<-xSeT),(xm>xSeT))
12 out+=outset             #Boolean added as 0 or 1

```

Table 9.7 Results of a run of the program in Table 9.6b

E	0.317
t	[1.839 1.198 1.077 1.035 1.017]
xm	[-1.026 -0.871 -0.279 -0.079 -0.04]
xSeT	[1.784 0.547 0.347 0.277 0.196]
outSeT	[False True False False False]
out	[0. 1. 0. 0. 0.]

9.5.4 Python Program

Single experiment

The program for checking the consistency of confidence intervals and confidence levels and its result is distributed over three cells. In the first cell of Table 9.6, three lists of the same length ($nList$, t , out) are generated, with their elements specific to the 5 series with different numbers of measurements. In the second cell, the simulation is run once; all logged lists are reported in Table 9.7 A numerical array out is created to sum the logical values of $outSeT$ as numbers 0 or 1. It counts the number of missed hits, i.e., when the true value is outside of the extended error range when the simulation is repeated.

Questions

Why does t have to be an array? Why is it not sufficient to specify it as a list?¹⁶

Replace lines 4 to 6 of Table 9.6a with a list comprehension!¹⁷

Which columns of Fig. 9.11 (S) are equivalent to Table 9.6b?¹⁸

¹⁶ Line 10 of Table 9.6 multiplies the array of standard errors with t , element-wise. This must be a numerical multiplication. Therefore, t has to be an array.

¹⁷ $t = np.array([sct.t.ppf(1-E/2, df = n-1) for n in nList])$

¹⁸ Columns A, D, F, G; set x in A, x_m in D, x_{SeT} in F, $outSeT$ in G.

Table 9.8 Multiple repetition of the simulation experiment; **a** (top cell) program code; **b** (middle cell) Table 9.6b changed into a function; and **c** (bottom cell) results for *t* values theoretically valid for an error probability of $E = 0.317$ (**b**) and $E = 0.05$ (**c**)

<pre> 13 nOut=np.zeros(lnL) 14 nT=10000 #Number of trials 15 for m in range(nT): nOut+=Check_if_out() 16 np.set_printoptions(precision=3) </pre>					
<pre> 1 def Check_if_out(): 2 x=npr.randn(nList[-1]) 3 xm=np.array([np.average(x[0:n]) for n in nList]) 4 xSeT= np.array([np.std(x[0:n],ddof=1)/np.sqrt(n) \ 5 for n in nList])*t #nList and t from table above 6 outSeT=np.logical_or((xm<-xSeT), (xm>xSeT)) 7 return outSeT </pre>					
E	0.317	nT	100000		
nOut/nT	[0.318	0.316	0.319	0.319	0.319]
E	0.05	nT	100000		
nOut/nT	[0.0501	0.05	0.05	0.0493	0.0493]

How do you have to change the logical query in line 11 of Table 9.6b if you want to count the hits instead of the misses?¹⁹

Error rates

The statistical experiment is multiply repeated in Table 9.8, in a for-loop calling the function *Check_if_out()* and summing up its logical output in a numerical array *nOut*. The statistical experiment is performed twice, first with $E = 0.317$ in line 3 of Table 9.6a (results in Table 9.8b) and second with $E = 0.05$ (results reported in Table 9.8c). The experimentally found miss rates $p_{Out} = n_{Out}/n_T$ are close to the expectations, $E = 0.317$ and $E = 0.05$.

Questions

Which lines of Table 9.6 have to be integrated into the function *Check_if_out*? What variable has to be returned?²⁰

9.6 Combining Results from Several Measurement Series

A combined result of two measurement series is obtained as a weighted average of the individual results with the squares of the reciprocal C-spec errors

¹⁹ `inSeT = np.logical_and(-xSeT < xm, xm < xSeT)`; the true value of x_m is 0.

²⁰ Lines 8 to 11 with `return outSeT`. The EXCEL equivalent is a rep-log procedure that calculates p_{Out} .

as weights. The C-spec errors are derived from internally and externally consistent variances.

Ψ *Worse makes good even better. Mostly, but not always.*

9.6.1 Combining Two Measurement Results

Calculating with variances

► **Mag** Two research groups use different methods, A and B, to determine the value of the same measurand. They report their non-identical results with different standard errors. For example, the acceleration due to gravity at a particular location has been determined by a drop test and a pendulum. As the head of the two groups, which result do you write in the project's final report?

► **Alac** Clearly the one with the smaller standard error.

► **Tim** I simply report both results, each with its own number of measurements and standard errors. The readers can then decide for themselves.

► **Mag** Neither answer is correct. As a supervisor, you should know how to combine both measurements and specify a value for the measurand whose standard error is even smaller than that for the better of the two measurements.

► **Tim** Alright, I have found a formula in the internet. The final result x_{mAB} is a weighted mean of the two results x_{mA} and x_{mB} , with the *reciprocals of the variances* var_A and var_B of the measurement series as weights w , Eq. 9.10.

$$x_{mAB} = \frac{w_A \cdot x_m(A) + w_B \cdot x_m(B)}{w_A + w_B}$$

$$w_A = \frac{1}{var_A} \quad w_B = \frac{1}{var_B} \quad (9.10)$$

Furthermore, the reciprocal value of the combined results' variance is calculated as the sum of the *reciprocals of the individual variances*:

$$var_{AB}^{-1} = var_A^{-1} + var_B^{-1} = w_A + w_B$$

$$x_{Std}(AB) = \sqrt{w_A + w_B} \quad (9.11)$$

► **Alac** Equation 9.3 is another neat formula, plus, it's trustworthy, because it fits our rule Ψ *Calculate with variances, report the standard error.*

► **Tim** Do we have to take *Student's t-value* into account? That's the *t-question*. My internet resource does not deal with it.

► **Alac** Let's check it with Ψ *Two within and one out of*.

► **Mag** Good idea. By the way, Eq. 9.11 is called the *internally consistent* variance. The greater the variance, the lower the measurement's weight in the combined result.

Questions

Due to Eq. 9.11, will the combined result's variance be greater or smaller than the variance of the best measurement?²¹

Provided the noise is the same for two measurement series with $n = 4$ and $n = 16$, by what factor should the *standard errors* of their mean values differ?²²

Calculation with C-spec errors

Equation 9.11 makes a statement about the variances of measurement series. We are, however, interested in confidence intervals and know already that the standard error is:

$$x_{Se} = \frac{x_{sd}}{\sqrt{n}} \quad (9.12)$$

and the C-spec error, with the t value obtained from the confidence level C and the degrees of freedom dof is:

$$x_{Ce} = x_{Se} \cdot t(C, dof) \quad (9.13)$$

Furthermore, we recall our broom rule:

Ψ *Calculate with variances, report the C-spec error!*

Consequently, we use in the formulas for mean and C-spec error the weights:

$$\text{weight } w = \frac{1}{x_{Ce}^2} \quad (9.14)$$

and check the results with hit rates.

Simulation

We set up two independent measurement series for the same measurand, one, A , with 16 individual measurements and the other one, B , with 4 individual measurements, so that they exhibit different variances. They are combined into a third series C of size 20. Furthermore, the respective results of A and B are combined into AB using the method described above.

²¹ The reciprocals of the variances are added; the variance of the overall result becomes smaller.

²² The standard errors of the mean values should differ by a factor of two (the ratio of the square roots of the numbers of measurements).

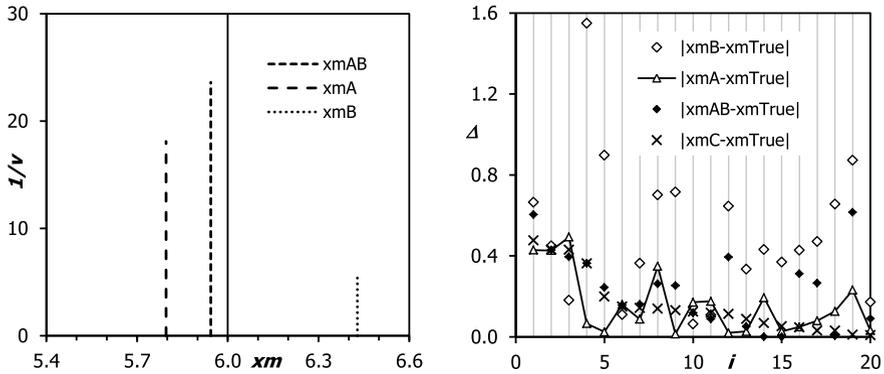


Fig. 9.12 a (left) A result for one measurement series each for A, B, and AB; the length of the vertical lines indicates the respective weight in the weighted sum; the true value is $x_m = 6$. b (right) Deviation of the mean values of the measurement series A, B, C, and AB from the true value, ordered according to the deviation $|x_{mC} - x_{mTrue}|$

We calculate the mean values x_{mA} , x_{mB} , and x_{mC} together with the squares of the C-spec errors according to Sect. 9.5.1. The combined result x_{mAB} is calculated as the weighted mean according to Eq. 9.10, together with its C-spec error according to Eq. 9.13, both with the weights in Eq. 9.14. The results for one random experiment are graphically shown in Fig. 9.12a. The “true” value of the measurand is 6. If the random experiment is repeated, different positions and heights of the vertical lines result.

The positions of the vertical lines in Fig. 9.12a correspond to the mean value x_m of the measurement series, the height to $1/x_{mC}e^{*2}$. In this example, the value of the combined measurements, x_{mAB} , is closest to the true value of 6.0.

We repeat the random experiment 20 times with a rep-log procedure and note the deviations of the four mean values for A, B, C, and AB from the true value; the result is shown in Fig. 9.12b. The experiments are ordered according to the outcome of experiment C, the direct evaluation of 20 measurements. The value for the series with four measurements (B) is usually farther away from, but sometimes closer to, the true value than the results of the series with 16 measurements (A).

Ψ Mostly, but not always! Fundamental rule of statistical reasoning:-).

The weighted mean values of (A) are joined with a solid line. We see that they are sometimes better and sometimes worse than those for the combined series (AB).

► **Tim** So, it would sometimes be better to report the result of experiment (A) instead of the combined result?

► **Mag** Keep in mind that we perform simulations according to *Ψ We know everything and play stupid*. In real life, the true value is unknown, and we have to be content with *Ψ Decide! Sometimes it will be wrong*. Let’s have a look at the hit rates from the rep-log procedures in Table 9.9. and Fig. 9.14 (S).

Table 9.9 Internally (left two cells) and externally (right two cells) consistent evaluation of the measurement series; the first line corresponds to a variant of line 8 in Table 9.11

vABmax=vABint	vABmax=vABint	vABmax=vABext	vABmax=vABext
Theor. 0.683	Theor. 0.950	Theor. 0.683	Theor. 0.950
Hits A 0.682	Hits A 0.950	Hits A 0.682	Hits A 0.952
Hits B 0.684	Hits B 0.950	Hits B 0.681	Hits B 0.951
Hits C 0.683	Hits C 0.950	Hits C 0.681	Hits C 0.951
HitsAB 0.633	HitsAB 0.932	HitsAB 0.448	HitsAB 0.406

► **Alac** The hit rates of A, B, and C are close to the theoretical value of 0.683 and 0.950, but AB’s hit rates are only about 0.63 and 0.93; they do not meet the expectation.

► **Tim** Therefore, the estimation of the standard error according to Eq. 9.11, called “internally consistent”, must be wrong.

► **Mag** It is not wrong but does not fully satisfy our requirements. Before drawing conclusions, let’s evaluate another formula to estimate a combined mean’s variance called “externally consistent”. Take a look at Eq. 9.15.

► **Tim** But that’s even worse. Look at the right two cells in Table 9.9 with *HitsAB* = 0.406.

► **Mag** That’s right. In the end, we have to calculate both variances and choose the bigger one to avoid too small an error range. In doing so, we yield hit rates *HitsAB* = 0.667 and 0.935, slightly better than those reported when taking the internally consistent variances alone.

Externally consistent variance

The *externally consistent variance* is calculated with the mean values x_{mA} and x_{mB} of series A and B and the combined mean value x_{mAB} :

$$v_{ABext} = \frac{w_A \cdot (x_{mA} - x_{mAB})^2 + w_B \cdot (x_{mB} - x_{mAB})^2}{(M - 1) \cdot (w_A + w_B)} \tag{9.15}$$

with M being the number of measurement series whose results are to be combined, here, $M = 2$, and w_A, w_B being the weights defined in Eq. 9.14.

9.6.2 Data Structure and Nomenclature

- A series A, N_A values
- B series B, N_B values

C	A and B together as one series
AB	results of A and B combined into one result
x_{True}	true value of the measurand
x_{mA}, x_{mB}, x_{mC}	mean values of series A and B
$x_{CeA}, x_{CeB}, x_{CeC}$	C-spec errors errors of $x_{mA}, x_{mB},$ and x_{mC}
w_A, w_B	weights of series x_{mA} and x_{mB} , Eq. 9.14
x_{mAB}	mean value of x_{mA} and x_{mB} combined
v_{ABint}	internal variance of the combined result
v_{ABext}	external variance of the combined result.

9.6.3 Spreadsheet Calculation

In Fig. 9.13 (S), two data sets A and B are generated, comprising, respectively, 16 and 4 normally distributed random numbers and the union C = A|B (see cell E3) with mean x_{mTRUE} and standard deviation x_{Ns} . They are evaluated for mean values x_m , and C-spec errors x_{Ce} , here for an error probability $E = 0.05$.

	A	B	C	D	E	F	G	H
1	xmTrue	6	xA	6.06	=AVERAGE(A)	E	0.05	
2	xNs	1	xmB	5.69	=AVERAGE(B)	tA	2.13	=T.INV.2T(E;15)
3			xmC	5.99	=AVERAGE(A;B)	tB	3.18	=T.INV.2T(E;3)
4						tC	2.09	=T.INV.2T(E;19)
5	=NORM.INV(RAND());xAtrue;xNs)					xCeA	0.55	=STDEV.S(A)/SQRT(16)*tA
6	A		B			xCeB	0.75	=STDEV.S(B)/SQRT(4)*tB
7	6.42		5.15			xCeC	0.44	=STDEV.S(A;B)/SQRT(20)*tC
22	4.98							

Fig. 9.13 (S) Sets A and B, normally distributed random numbers; evaluation of the data sets and C = A|B for mean value (C1:D3), as well as the C-spec error of the mean (F5:G7) for E in G1

	I	J	K	L	M	N	O	P
1	wA	3.55	=1/xCeA^2	xmAB	6.01	=(xmA*wA+xmB*wB)/(wA+wB)		
2	wB	0.51	=1/xCeB^2	vAB	0.25	=(wA+wB)^-1		
3	wC	4.62	=1/xCeC^2	xsAB	0.50	=SQRT((wA+wB)^-1)		
4								100000
5				hitA	TRUE	=AND(xmA-xCeA<xmTrue;xmTrue<xmA+xCeA)		0.941
6				hitB	TRUE	=AND(xmB-xCeB<xmTrue;xmTrue<xmB+xCeB)		0.950
7				hitC	TRUE	=AND(xmC-xCeC<xmTrue;xmTrue<xmC+xCeC)		0.944
8				hitAB	TRUE	=AND(xmAB-xsAB<xmTrue;xmTrue<xmAB+xsAB)		0.925
9								0.950 =1-E

Fig. 9.14 (S) In O5:O8 are the hit rates of the experiments with the four sets A, B, C, and AB for 100,000 repetitions. The value expected from statistical theory is given in O9

Table 9.10 Defining four measurement series A, B, C, and AB; calculating their means, variances, and C-spec errors; t_A , t_B , t_C are global variables from the main program in Table 9.12; continued in Table 9.11

```

1  import numpy.random as npr
2
3  def measurements():
4      A=xmTrue+xNs*npr.randn(16)
5      B=xmTrue+xNs*npr.randn(4)
6      C=np.concatenate([A,B]) #Not independent of A and B
7      xmA=np.mean(A)
8      xmB=np.mean(B)
9      xmC=np.mean(C)
10     #C-spec errors by considering t:
11     xCeA=np.std(A, ddof=1)/np.sqrt(len(A))*tA
12     xCeB=np.std(B, ddof=1)/np.sqrt(len(B))*tB
13     xCeC=np.std(C, ddof=1)/np.sqrt(len(C))*tC
14     hitA=(xmA-xCeA)<xmTrue<(xmA+xCeA)
15     hitB=(xmB-xCeB)<xmTrue<(xmB+xCeB)
16     hitC=(xmC-xCeC)<xmTrue<(xmC+xCeC)

```

Question

Which two of the three sets A, B, and C are pairwise independent?²³

In Fig. 9.14 (S), first, the results of A and B are combined into one final result x_{mAB} with a calculated error x_{sAB} . Then, the statistical experiment is repeated 100,000 times with a rep-log procedure to count how often the error ranges $x_m \pm x_{Ce}$ capture the true value for the sets A, B, C, and AB. The theoretical rate is 0.950, which is closely reached for A, B, C, but with ≈ 0.925 not for AB. We conclude that the internally consistent variance is not sufficient to estimate the error range. Therefore, in the following section with Python programs, the alternative with externally-consistent error is also considered.

9.6.4 Python, Internally and Externally Consistent Error of the Combined Result

The Python function in Tables 9.10 and 9.11 simulates the measurement process. In the function in Table 9.10, the data sets A, B, and C are generated and evaluated for mean value x_m , variance v , and C-spec error x_{Ce} of the mean.

Table 9.11 continues the function *measurements* begun in Table 9.10. The combined result x_{mAB} , x_{sAB} is built, where x_{sAB} is obtained as the maximum of the

²³ A and B are independent of each other, because their members are generated in two different ranges. C, A, and C, B are not independent, because C is the union of A and B.

Table 9.11 Continuation of Table 9.11; line 22 is varied according to which variance is to be calculated

16	wA=(xCeA)**-2	#Weight
17	wB=(xCeB)**-2	
18	xmAB=(xmA*wA+xmB*wB)/(wA+wB)	
19	vABint=(wA+wB)**-1	#Variances combined
20	vABext=(wA*(xmA-xmAB)**2	
21	+wB*(xmB-xmAB)**2)/(wA+wB)	
22	vABmax=max(vABint,vABext)	#=vABint
23	xsAB=np.sqrt(vABmax)	
	#Confid. level not yet clear. Hit rate to be determ.:	
24	hitAB=(xmAB-xsAB)<xmTrue<(xmAB+xsAB)	
25	return hitA, hitB, hitC, hitAB	
hits	(True, False, True, True)	

Table 9.12 Main program calling the function measurements in a for-loop. The values in the right cell are for the current specification $E = 0.05$ and for another run with $E = 0.317$

1	import scipy.stats as sct	
2	E=0.05	theor. 0.950
3	tA =sct.t.ppf(1-E/2,df=15)	Hits A 0.951
4	tB =sct.t.ppf(1-E/2,df=3)	Hits B 0.951
5	tC =sct.t.ppf(1-E/2,df=19)	Hits C 0.948
6		Hits AB 0.936
7	xmTrue=6 #True mass	
8	xNs=1	
9	Nrep=10000	
10	hitA, hitB, hitC, hitAB=0,0,0,0	theor. 0.683
11	for rep in range(Nrep):	Hits A 0.677
12	hits=measurements()	Hits B 0.682
13	hitA+=hits[0]	Hits C 0.683
14	hitB+=hits[1]	Hits AB 0.688
15	hitC+=hits[2]	
16	hitAB+=hits[3]	

internally and externally consistent variances. Line 24 checks whether the true value x_{mTrue} is captured by $x_{mAB} \pm x_{sAB}$, and the four Boolean values are returned.

In the main program in Table 9.12, the statistical experiment is repeated $N_{rep} = 10,000$ times for a specified error probability E , and the hit rates are determined. The results are shown in the right cell. The hit rate for AB with 0.936 is not much closer to the theoretical value of 0.950 than the 0.925 reported in Fig. 9.14 (S) for the internally consistent variance.

9.7 Propagation of Standard Deviations

You cannot prevent errors from propagating. It is in their nature. Learn to live with it! We simulate the propagation of variances and standard deviations in sums, products, and powers with statistical experiments on sets with 100 elements. Note: For products, the relative variances add up; for sums, the absolute variances. The propagation of confidence intervals is treated in Exercise 9.8.

9.7.1 Rules for Propagation of Standard Deviations

General Rule

The final result z of a measurement series is often a function of one or more measured physical quantities x : $z = f(x_1, x_2, \dots)$. The standard theory of error propagation considers the x_i as random variables, normally distributed around the “true” value, with the result:

- ▶ The variance of the final result is a weighted sum of the variances $\text{var}(x_i)$ of the measured values of the individual variables x_i . Remember: It is the variances that propagate through the formulas to the final result. But, ultimately, the C-spec error has to be reported together with the estimate of the mean.

For two variables, x and y , the equation is

$$\text{var}_z = w_x \text{var}_x + w_y \text{var}_y \text{ with } w_x = \left(\frac{\partial z}{\partial x} \right)^2 \text{ and } w_y = \left(\frac{\partial z}{\partial y} \right)^2 \quad (9.16)$$

which can straightforwardly be extended for more variables. The variables v denote the empirical variance of the data series, Eq. 9.2. The weights w in the sum are the squares of the derivatives of z with respect to the associated variables; the greater the slope, the greater the weight.

As an example, we consider the calculation of the volume V_K of a sphere from its diameter d_K : $V_K = \frac{\pi}{6} d_K^3$

$$\text{var}(V_K) = \left(\frac{\pi}{2} d_K^2 \right)^2 \cdot \text{var}(d_K) \quad (9.17)$$

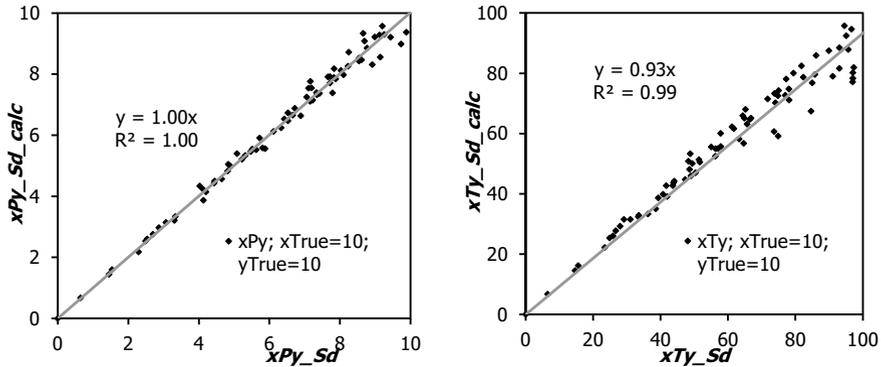


Fig. 9.15 **a** (left) Standard deviation of the sum $xPy = x + y$, calculated versus empirical values. **b** (right) Standard deviation of the product $xTy = x \cdot y$, calculated versus empirical values. The standard deviation is up to ten times greater than the mean value. $X_{\text{True}}, Y_{\text{True}}$ are the true values

Question

What are the physical units on both sides of Eq. 9.17?²⁴

In the following sections, we examine the propagation of the standard deviations of sets of 100 measurement points into the final result.

Error propagation in sums

For a sum $z = x + y$, the weights in Eq. 9.16 are $w_x = 1$ and $w_y = 1$ so that $\text{var}_z = \text{var}_x + \text{var}_y$. For a difference $z = x - y$, the coefficients are the same because the partial derivatives are squared so that we get the rule:

- The variance of a sum or difference is the sum of the variances of the summands.

Ψ Calculate with variances, report the C-spec error!

For the data in Fig. 9.15a, two data sets x and y with various standard deviations for a normally distributed noise are added element-wise to get xPy (x Plus y). The standard deviations of the set of the sums, as derived from the standard deviations of x and y , are plotted against the empirically determined standard deviations xPy_Sd of the set xPy . The trend line has a slope 1.00, indicating accordance with the theoretical expectation. In general, that cannot be taken for granted, as

²⁴ The dimensional analysis $\text{m}^6 = \text{m}^4 \cdot \text{m}^2$ shows that Eq. 9.17 does not contain gross errors.

can be seen in Fig. 9.15b for xTy (x Times y), where the trend line through a corresponding plot has a slope of only 0.93.

Questions

Let the two measurands x and y have standard deviations x_{sd} and y_{sd} , respectively. How large is, according to the propagation rules, the standard deviation of the sum $x + y$, and how large is the standard deviation of the difference $x - y$?²⁵

The proportionality factor in the trend line in Fig. 9.15b is, with 0.93, clearly smaller than 1. Does this contradict theory?²⁶

- ▶ **Task** Vary the mean values and standard deviations of x and y and log the mean value and standard deviation of the sum xPy ! Which regularity do you assume? Derive the expected result from Eq. 9.16!

Error propagation in products

In Fig. 9.15b, the experimentally determined standard deviation xTy_{sd} of the set xTy is compared with the theoretical standard deviation xTy_{sd_calc} calculated from the standard deviations of x and y . For the product $z = x \cdot y$, Eq. 9.16 yields the weights

$$w_x = \left(\frac{\partial z}{\partial x}\right)^2 = y^2 \text{ and } w_y = \left(\frac{\partial z}{\partial y}\right)^2 = x^2 \tag{9.18}$$

hence,

$$\begin{aligned} var_z &= y^2 \cdot var_x + x^2 \cdot var_y \\ \frac{var_z}{z^2} &= \frac{var_x}{x^2} + \frac{var_y}{y^2} \\ \left(\frac{z_{sd}}{z_m}\right)^2 &= \left(\frac{x_{sd}}{x_m}\right)^2 + \left(\frac{y_{sd}}{y_m}\right)^2 \end{aligned} \tag{9.19}$$

- ▶ The square of the relative standard variations of a product is the sum of the relative standard variations of the factors.

Ψ Calculate with variances, report the standard error!

Ψ Even better: report confidence level and confidence interval!

²⁵ The standard deviations of the sum and the difference are equal: $(x + y)_{sd} = (x - y)_{sd} = \sqrt{x_{sd}^2 + y_{sd}^2}$.

²⁶ The formulas for error propagation, here, the square root of Eq. 9.19 for products, are based on a Taylor series development and are, therefore, valid only for small variances of the independent variables, a condition no longer satisfied here.

Standard deviations that are too large distort the result

In Fig. 9.15b, the standard deviation of the product, calculated according to the propagation formula, is plotted against the empirical one. The slope of the regression line is 0.93, significantly smaller than 1. This is because the standard deviations of the measurands are too large. The formulas for error propagation are based on a Taylor series development of the function, in Eq. 9.16 only to the first order. So, they are valid only for a small interval around the independent variables. The standard deviations in Fig. 9.15b become too large. For small standard deviations (< 60), the slope is greater than that of the trend line drawn in the figure, actually 0.99, as an appropriate check shows.

Error propagation in powers

We now investigate the power function

$$z = x^n \tag{9.20}$$

The variance of the power function is derived from Eq. 9.16 to

$$var_z = a_x var_x \text{ with } a_x = \left(\frac{\partial z}{\partial x}\right)^2 = (n x_m^{n-1})^2 \tag{9.21}$$

Thus, the expected standard deviation of z is

$$z_{Sd} = n \cdot x_m^{n-1} \cdot x_{Sd} \tag{9.22}$$

Figure 9.16 displays the empirical standard deviation for a power $n = 2$ (in **a**) and $n = 4$ (in **b**), both for a noise level $x_{Ns} = 0.1$, as a function of the mean value x_m .

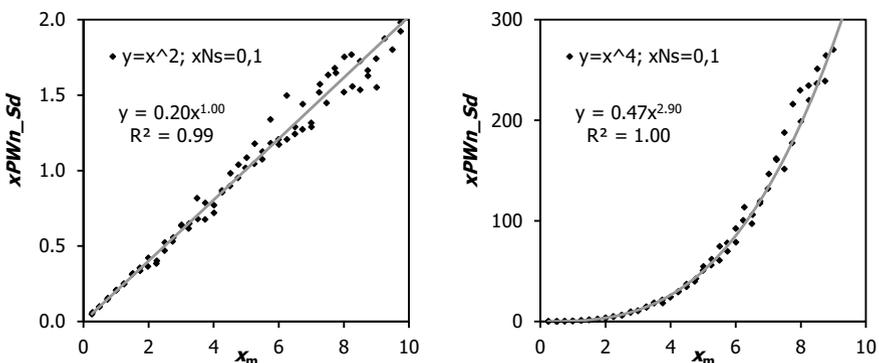


Fig.9.16 **a** (left) Standard deviation of the second power of the random variable x with the noise level $x_{Ns} = 0.1$ as a function of the empirical mean x_t of the data set. **b** (right) Same as **a**, but with a power of $n = 4$

Question

Do the formulas for the trend lines in Fig. 9.16a and b agree with the theoretical expectation?²⁷

Experimental procedure

We set up two data series (x, y) (size 100 each), defined as a true value (x_{True} or y_{True}) plus a normally distributed noise (standard deviations x_{Ns} , y_{Ns}). We then determine their mean values (x_m, y_m) and standard deviations (x_{Sd} , y_{Sd}).

We add and multiply the two sets pairwise to get the new series xPy (“ x Plus y ”) and xTp (“ x Times y ”) and determine their mean values xPy_m and xTy_m and standard deviations xPy_{Sd} and xTy_{Sd} , respectively. Furthermore, we create a new series $xPWn$ by raising the elements of x to the power n and determine its mean value $xPWn_m$ and standard deviation $xPWn_{\text{Sd}}$.

We vary the noise levels and log the mean values and standard deviations of the sets with a rep-log procedure. This is achieved:

- in EXCEL with a VBA routine that repetitively changes the parameters of the spreadsheet calculation,
- in Python by outsourcing the parts of the program equivalent to the spreadsheet into a function, repetitively called by the main program.

We plot the theoretically calculated standard deviations of xPy and xTy versus the empirical ones in a scatter plot and lay a straight trend line through the data points (see Fig. 9.15). The slope of such trend lines is 1 if the theoretically calculated values correspond to the empirical ones.

We set up another set $xPWn$ by raising the elements of the series x to the n th power and present its standard deviation as a function of the mean value x_m of the series x , together with a power trend line through the data (see Fig. 9.16).

Questions

We have called the collections x, y, xPy, xTy data series, not data sets. Why is the term “set” inappropriate here?²⁸

As a result of our simulations, we plot the *standard deviations* of sums and products against the *standard deviations* of the summands or factors, without considering either the number n of measurements or the t values. Why does that not hinder the illustration of the laws of error propagation?²⁹

²⁷ Yes, we expect $z_{\text{Sd}} \propto x^3$ for $f(x) \propto x^4$ and $z_{\text{Sd}} \propto x$ for $f(x) \propto x^2$ and find $y = 0.47x^{2.90}$ and $y = 0.20x^{1.00}$.

²⁸ A set is an unordered collection. However, our data are ordered. The n th element of xPy and xTy is obtained from the n th elements of x and y .

²⁹ The sets for all terms have the same number n of measurements, so that the factors n and t are common to all of them and to the results.

9.7.2 Data Structure and Nomenclature

x_{True}, y_{True}	true values
x_{Ns}, y_{Ns}	noise levels (standard deviation of a normal distribution)
x, y	two series of true values plus noise
x_m, y_m	mean of x and y
x_{Sd}, y_{Sd}	standard deviations of x and y
xPy	series x Plus y
xTy	series x Times y
$xPWn$	series x to the PoWer of n
xPy_m, xPy_{Sd}	mean and standard deviation of xPy , correspondingly for xTy and $yPWn$
xPy_{Sd_calc}	standard deviation of xPy calculated from x_s and y_s , correspondingly for xTy and $yPWn$.

9.7.3 Spreadsheet Calculation

In A13:B112 of Fig. 9.17 (S), we create two series x and y of normally distributed numbers, regarded as measurement values specified by their means (“true values”) and standard deviations (“noise”) in A1:B4. The series x and y are evaluated for mean value and standard deviation in D1:E4.

We then create three new data series in D13:E112 by adding x and y (xPy), multiplying x and y (xTy), and raising x to the power n ($xPWn$) specified in B5. These series are evaluated in Fig. 9.18 (S). There, the standard deviations xPy_{Sd_calc} , xTy_{Sd_calc} , and $xPWn_{Sd_calc}$ are also reported, theoretically derived from the standard deviations of the original series.

Error propagation in sums

In Fig. 9.18 (S), the empirical standard deviation xPy_{Sd} is calculated from the series xPy . The theoretical standard deviation xPy_{Sd_calc} is calculated as the root of the sum of the squares of the summands’ standard deviations. The two values are identical to the second decimal.

	A	B	C	D	E	F	G		A	B	C	D	E	F	
1	xTrue	10.00		xm	10.00	=AVERAGE(x)			=NORM.INV(RAND();xTrue;xNs)						
2	xNs	0.10		xSd	0.11	=STDEV.S(x)									
3	yTrue	10.00		ym	9.54	=AVERAGE(y)									
4	yNs	7.50		ySd	7.56	=STDEV.S(y)									
5	n	2.00													
11										=x+y	=x*y	=x^n			
12		x		y		xPy		xTy		xPWn					
13		10.06		17.89		27.95		180.02		101					
112		10.00		17.31		27.31		173.07		100					

Fig. 9.17 (S) Two normally distributed data series x and y of size 100, defined by their means x_{True} and y_{True} (“true” values) and standard deviations x_{Ns} and y_{Ns} (“noise”).in a Gaussian

We vary the standard deviations of x and y independently of each other systematically with a rep-log procedure, record the empirical and calculated standard deviations of the sum, and plot both as data points in Fig. 9.15a. The trend through the data points is a straight line with a slope very close to 1. Thus, we have retrieved the theoretical statement with our simulation; at least, we cannot object to it.

Error propagation in products

The product of x and y is stored as a new series xTy in column E of Fig. 9.17 (S) and evaluated in column M of Fig. 9.18 (S).

We vary the standard deviations of x and y independently of each other systematically with a log procedure and record the product’s empirical and calculated standard deviations. The standard deviation xTy_Sd_calc of the product xTy is calculated from those of the factors as (see Eq. 9.19)

$$xTySdCalc = zSdCalc = \sqrt{(y_m xSd)^2 + (x_m ySd)^2} \tag{9.23}$$

The results are shown in Fig. 9.15b. The trend line has a slope of 0.93, less than the expected 1.00. The reason for this is that the standard deviations xSd and ySd are too large (see discussion in Sect. 9.7.1).

Error propagation in powers

► **Task** Change n in Fig. 9.17 (S) or Table 9.13 and review your interpretation!

In Fig. 9.17 (S), column F, x is raised to the n th power; the result is stored as $xPWn$. The standard deviation $xPWn_Sd$ of this series is determined in Fig. 9.18 (S) in M8 together with the theoretical one $xPWn_Sd_calc$ in M9.

In a rep-log procedure, the value of x_{True} is varied from 0.5 to 10 for a noise level of 0.1. The results for $n = 4$ and $n = 2$ are shown in Fig. 9.16a and b, respectively; they are approximated by a power function as a trend line.

	I	J	K	L	M	N
1		x+y			x*y	
2	xPy_m	20.34 =AVERAGE(xPy)		xTy_m	103.51 =AVERAGE(xTy)	
3		20.34 =xm+ym			103.39 =xm*ym	
4	xPy_Sd	7.79 =STDEV.S(xPy)		xTy_Sd	77.78 =STDEV.S(xTy)	
5	xPy_Sd_calc	7.77 =SQRT(xSd^2		xTy_Sd_calc	77.65 =SQRT((ym*xSd)^2	
6		+ySd^2)			+(xm*ySd)^2)	
7		x^n				
8	xPWn_m	100 =AVERAGE(xn)		xPWn_Sd	1.75 =STDEV.S(xPWn)	
9	xm^n	100 =xm^n		xPWn_Sd_calc	1.75 =n*xm^(n-1)*xSd	

Fig. 9.18 (S) Continuation of Fig. 9.17 (S); evaluation of the data sets xPy (“ x Plus y ”), xTy (“ x Times y ”) and $xPWn$ (“ x to the power of n ”)

Table 9.13 Python function, part corresponding to Fig. 9.17 (S) continued in Fig. 9.18 (S)

```

1 import numpy.random as npr
2 I=np.zeros(14)
3 xTrue, yTrue, n = 10, 10, 2      #True values
4 def simu(xNs,yNs):
5     x=xTrue+xNs*npr.randn(100)  #Array with 100 items
6     y=yTrue+yNs*npr.randn(100)
7     xm=np.mean(x)
8     xSd=np.std(x)
9     ym=np.mean(y)
10    ySd=np.std(y)
11
12    xPy=x+y
13    xTy=x*y
14    xPWn=x**n

```

Question/task

Write the formula in N9 of Fig. 9.18 (S) in mathematical form!³⁰

9.7.4 Python Program

The spreadsheet calculations can be translated directly into a Python program, Tables 9.13 and 9.14. The program's core is the function *simu* calculating with the parameters x_{True} , y_{True} , n , x_{Ns} , y_{Ns} . The parameters x_{Ns} and y_{Ns} are chosen to be handed over in the function head, whereas x_{True} , y_{True} , and n are treated as global parameters.

The arrays x and y are created anew within the function with every call, and consequently, the scalars and arrays derived from them are recalculated. The means, empirical, and calculated standard deviations for every set are returned as an array I with 14 elements (Table 9.14).

Error propagation for sums and products

The main program for calculating the standard deviations of the sum, displayed in Fig. 9.15a, is given in Table 9.15. The standard deviations x_{Ns} and y_{Ns} are varied, whereas the true values x_{True} , y_{True} are specified before the for-loop and remain constant during the whole experiment.

Similar main programs work for the product with:

```
xTy.append(I[7]).
```

³⁰ $Z = x^n$; $Z_{Sd} = n x^{n-1} x_{Sd}$.

Table 9.14 Continuation of Table 9.13, part of the function *simu* corresponding to Fig. 9.18 (S)

```

15     I[1]=np.mean(xPy)
16     I[2]=xm+ym
17     I[3]=np.std(xPy)
18     I[4]=np.sqrt(xSd**2+ySd**2)
19     I[5]=np.mean(xTy)
20     I[6]=xm*ym
21     I[7]=np.std(xTy)
22     I[8]=np.sqrt((ym*xSd)**2+(xm*ySd)**2)
23     I[9]=np.mean(xPWn)
24     I[10]=xm**n
25     I[11]=xSd**n
26     I[12]=np.std(xPWn)
27     I[13]=n*xm**(n-1)*xSd
28     return I #Mean and standard deviation
           (empirical and theoretical) of x*y, x*y, and x^n

```

Table 9.15 Main program for calculating the standard deviation of a sum $x + y$ for varying standard deviations of x and y

```

29 xPy_Sd=[]
30 xPy_Sd_calc=[]
31 #Variation of the noise levels xNs and yNs
32 #Constant xTrue, yTrue
33 xTrue=10.0
34 yTrue=10.0
35 for noiseX in np.arange(0.5,8,1):
36     for noiseY in np.arange(0.5,8,1):
37         I=simu(noiseX,noiseY)
38         xPy_Sd.append(I[3])
39         xPy_Sd_calc.append(I[4])

```

```
xTy_Sd.append(I[8]).
```

Table 9.16 presents a program for achieving a plot such as that in Fig. 9.15a.

Power-law trend line

A power-law trend line $y = a \cdot x^n$ is obtained with the function *fitPowLaw* in Table 9.17, by fitting a linear trend line $\log y = a + m \cdot \log x$ to the logarithmized data $\log x = \log(x)$ and $\log y = \log(y)$. The transformation to $y = a_R x^{p_R}$ is achieved with $p_R = m$ and $a_R = \exp(a)$.

Error propagation for powers

Table 9.18 shows the main program for calculating the standard deviation of a power x^n as a function of the true value x_{True} . A program similar to Table 9.16 achieves a fit to the data points and results in a plot corresponding to Fig. 9.16.

Table 9.16 a (top) Program for producing a figure as in Fig. 9.15a, b (bottom) function to draw a straight line through (x, y)

```

1  FigStd('xPy_Sd',0,10,2,'xPy_Sd_calc',0,10,2)
2  plt.plot(xPy_Sd,xPy_Sd_calc,'kD',ms=3,
3           label='StDev Sum;\nxTrue='+str(xTrue)+\
4           ';yTrue='+str(yTrue))
5  PlotLin(xPy_Sd,xPy_Sd_calc,a=False)
6  plt.legend()
7  import statsmodels.api as sm
8
9  def PlotLin(x,y,a=False): #Linear regression line
10     if a==False:         #No y-axis intercept
11         model=sm.OLS(y,x)
12         results=model.fit()
13         mR=results.params[0]
14         yLin=mR*np.array(x)
15         plt.plot(x,yLin,'k-',label="y=%.2f*x"%mR)
16     if a==True:         #y-axis intercept allowed
17         xx=sm.add_constant(x)
18         model=sm.OLS(y,xx)
19         results=model.fit()
20         aR=results.params[0]
21         mR=results.params[1]
22         yLin=aR+mR*np.array(x)
23         lbl="y=%.2f+%.2fx"%(aR,mR)
24         plt.plot(x,yLin,'k-',label=lbl)

```

Table 9.17 Plotting a power-law trend line by a straight line through logarithmized data

```

1  def fitPowLaw(x,y):
2     logx=np.log(x)
3     logx=sm.add_constant(logx)    #y-axis intercept allowed
4     logy=np.log(y)
5     model=sm.OLS(logy,logx)      #Linear regression
6     results=model.fit()
7     aR=np.exp(results.params[0]) #Amplitude
8     pR=results.params[1]        #Power
9     R2=results.rsquared
10    #print(pR)    #Only in test phase
11    yPow=aR*x**pR
12    lbl="y=%.2f*x**%.2f,R²=%.2f"%(aR,pR,R2)
13    plt.plot(x,yPow,'k-',label=lbl)

```

Table 9.18 **a** (top) Main program for the power series $xPWn$, *simu* from Table 9.13; **b** (bottom) Plotting the original data and a power trendline through the data

```

1  xPWn=[]
2  xPWn_Sd=[]
3  xtp=[]
4  n=2
5  #Variation of xTrue for constant noise level
6  xNs=0.1
7  for x_1 in np.arange(0.25,10,0.25):
8      for rep in np.arange(2):
9          xTrue=x_1
10         I=simu(0.1,0.1)
11         xtp.append(I[10]**(1/n))
12         xPWn.append(I[12])
13         xPWn_Sd.append(I[13])
14
15  FigStd('xTrue',0,10,2,'xPWn_Sd',0,2,0.5)
16  plt.plot(xtp,xPWn_Sd,'kD',ms=3)
17  fitPowLaw(xtp,xPWn_Sd) #Function in table above
18  plt.legend()

```

9.8 Propagation of Confidence Intervals

We learn how to get from variances of two data sets to a confidence interval for a combined result, sum or product of the two measurands. We use a statistical simulation for the last step to get confidence-specified errors (“C-spec errors”).

9.8.1 From Variance to Confidence

In Exercise 9.7, we investigate the propagation of standard variations in sums, product, and powers by means of series of 100 elements. In real-world experiments, usually much less measurements are performed so that t factors have to be taken into account. Furthermore, the number of measurements can be different for the different operands. As a consequence, the C-spec error of the result cannot be deduced straightforwardly from the standard deviation of the result; we have to calculate with C-spec errors also for the operands.

Measurement series for single measurands

In Fig. 9.19 (S), two measurement series x_{Ai} and x_{Bi} are simulated, with the true values x_A and x_B and the noise levels ns_A and ns_B , specified in rows 1 and 2,

	A	B	C	D	E	F	G	H	I	J	K
1	x_{Ai}	x_{Bi}		x_A	4			x_B	2		
2	3.81	1.85		ns_A	0.2			ns_B	0.1		
3	3.98	1.89									
4	3.83	1.98		n_A	4 =COUNT(x_{Ai})			n_B	10 =COUNT(x_{Bi})		
5	3.84	1.92		x_{Am}	3.87 =AVERAGE(x_{Ai})			x_{Bm}	1.94 =AVERAGE(x_{Bi})		
6		1.98		x_{Ase}	0.04 =STDEV.S(x_{Ai})/SQRT(n_A)			x_{Bse}	0.03 =STDEV.S(x_{Bi})/SQRT(n_B)		
7		1.98									
8		2.12		Con	0.683						
9		1.83		t_A	1.20 =T.INV.2T(1-Con; n_A-1)			t_B	1.06 =T.INV.2T(1-Con; n_B-1)		
10		1.97		x_{Ace}	0.05 = $x_{Ase} * t_A$			x_{Bce}	0.03 = $x_{Bse} * t_B$		
11		1.88									
12				FALSE	6852			FALSE	6865		
13				=AND($x_{Am}-x_{Ace}<x_A$; $x_A<x_{Am}+x_{Ace}$)				=AND($x_{Bm}-x_{Bce}<x_B$; $x_B<x_{Bm}+x_{Bce}$)			
14					10000				10000		

Fig. 9.19 (S) Confidence intervals x_{Ace} and x_{Bce} for a confidence level Con for two measurement series x_{Ai} and x_{Bi}

entering our standard function $x_{Ai} = [=NORM.INV(RAND());X_A;NS_A]$ and $x_{Bi} = [=NORM.INV(RAND());X_B;NS_B]$ for the simulation of noisy measurements.

In rows 4 to 6, the number of elements (n_A , n_B), the mean values (x_{Am} , x_{Bm}), and the standard errors (x_{Ase} , x_{Bse}) of x_{Ai} and x_{Bi} are determined. Remember: the *standard error of the mean* is obtained as square root of the *variance of the series* (their standard deviation) divided by the square root of the number of elements in the series, Eq. 9.12.

To come from the standard error x_{Se} of the mean to a *confidence interval* x_{Ce} around the mean, we have to specify a *confidence level*. This is done in E8 with $Con = 0.683$, valid for the standard error when the number of measurements is large. Student’s *t*-values for this confidence level are determined in line 9 and multiplied in line 10 with the standard error to yield the C-spec errors x_{Ace} and x_{Bce} of the confidence intervals.

In E12 and I12, we check in the usual way whether the confidence intervals capture the true values. Furthermore, with a rep-procedure, we check how often that is the case for 10,000 repetitions of the statistical experiment (reported in F12 and J12). The numbers are close to the expected values.

9.8.2 Sum and Product of Two Measurands

In Fig. 9.20 (S), we determine product AtP (“A times B”) and sum ApB (“A plus B”) of the means x_{Am} and x_{Bm} of the two measurement series x_{Ai} and x_{Bi} and calculate C-spec errors AtB_{ce} and ApB_{ce} for the errors obtained according to the rules of error propagation explained in Exercise 9.7. They include tentative *t* factors $tAx_{B_{emp}}$ and $tApB_{emp}$, first set to 1 in N5 and S5. With these intervals, we get hit rates of more than 7000 for 10,000 repetitions of the statistical experiment,

	M	N	O	P	Q	R	S	T	U	V
1	AtBtrue	8.00	=xA*xB			ApBtrue	6.00	=xA+xB		
2	AtBemp	8.10	=xAm*xBm			ApBemp	6.01	=xAm+xBm		
3	varA	0.001	=(xAce/xAm)^2			varAp	0.01	=xAce^2		
4	varB	0.000	=(xBce/xBm)^2			varBp	0.00	=xBce^2		
5	tAtBemp	0.959				tApBemp	0.986			
6	AtBce	0.26				Apbce	0.12			
7								=SQRT(varAp+varBp)*tApBemp		
8								=AND(AtBemp-AtBce<AtBtrue;AtBtrue<AtBemp+AtBce)		
9	TRUE	6805	10000			TRUE	6833	10000		
10								1.03 =T.INV.2T(1-O9/P9;16)		1.00 =T.INV.2T(1-T9/U9;1000)
11								1.03 =T.INV.2T(1-Con;16)		1.00 =T.INV.2T(1-Con;1000)
12								1.01 =O11/O10		1.00 =T11/T10
13								0.96 =O12*tAxBemp		0.99 =T12*tApBemp

Fig. 9.20 (S) Product AtB (“times”) and Sum ApB (“plus”) of the results of the two measurement series of Fig. 9.19 (S)

significantly larger than expected from a confidence level $Con = 0.683$. The reason for this discrepancy is that the rules for error propagation set up in Exercise 9.7 are mathematically correct only if the number of measurements is big enough.

C-spec error by adapting the t factor

We do not delve deeper into the mathematical derivation but reduce the hit rates by adapting the tentative t factors. To do so, we calculate Student’s t value:

- for the empirically obtained hit rates (O12 and T12); they should approach the confidence level after some iterations,
- for the envisaged confidence level Con (O11 and T11); they do not change in the course of the simulation.

We build the ratio of the two t values and multiply with the current tentative t values (results in P13, T13) and copy/paste the results into $tAtB_{emp}$ and $tApB_{emp}$ (N5, S5). We repeat the statistical simulation and get hit rates 6738 (with $t = 0.940$ for A_B) and 6708 (with $t = 0.962$ for ApB). After a second iteration, the results are 6805 (with $t = 0.959$) and 6833 (with $t = 0.986$), indeed close to the values expected for the specified confidence level. The quantities AxB_{ce} and ApB_{ce} are called the confidence-specified errors, or short *C-spec errors*.

9.9 Mass of a Thin Film on a Glass Substrate

Our task is to determine the mass of a thin film on a glass plate. To this end, the glass plate is weighed several times on a microbalance, before and

after being coated. The mass of the film is obtained with recursion to error propagation and t statistics.

9.9.1 Instructions for Use for Accurate Measurements and Their Results

A measuring process ...

In an ideal coating process, a glass substrate of mass $m_{\text{Sub}} = 1 \text{ g}$ is coated with a thin film of mass $m_{\text{F}} = 1 \text{ mg}$. To determine the film's mass, the substrate is weighed before and after coating. The balance's nominal accuracy is $1 \mu\text{g}$, which means that the mass is displayed in grams, with six digits after the decimal point. However, the measurement's actual accuracy is significantly lower, e.g., due to disturbances by air currents or building vibrations.

Questions

What is the ratio of the masses of the substrate and the thin film?³¹

How many digits after the decimal point can the layer's mass be determined nominally, if given in mg?³²

By what percentage does the coating increase the mass of the sample?³³

... and its simulation

The substrate is weighed n times before and n times after being coated. The process is simulated by series $m_{\text{Bef}} = m_{\text{Sub}} + \text{noise}$ and $m_{\text{Aft}} = m_{\text{Sub}} + m_{\text{F}} + \text{noise}$, where the noise is normally distributed with m_{Ns} as the standard deviation.

EXCEL quotes each number with 15 digits, something like 0.569410526368089, Python even with 16 digits. Therefore, the results of the simulated weighing have to be rounded to six digits after the decimal point to mimic the display resolution of the balance: `ROUND(...;6) = 0.569411` and `np.round(..., 6)`.

The following explanations are valid for the preceding simulation but also for real measurements.

Immediate evaluation of the measurement results

The *standard errors* of the masses of the uncoated and coated substrate are calculated as the standard deviation of the measurement series divided by the square root of the number n of measurements in the series. In order to get the *C-spec errors* m_{BefCe} and m_{AftCe} (half the width of the *confidence intervals*), the standard error has to be

³¹ $1 \text{ g}/1 \text{ mg} = 1000$; the substrate is 1000 times heavier than the film.

³² Displaying an accuracy of $1 \mu\text{g}$, the films' mass could naively be specified as, e.g., 1.001 mg.

³³ 1 mg of 1 g , corresponding to 0.1%.

multiplied by Student's t -factor that is determined by the degree of freedom ($dof = n - 1$) and the specified confidence level Con . In this exercise, we choose $Con = 0.683$ which corresponds to the confidence level of the standard error if the number n of measurements is big enough.

The mass of the film is estimated as the difference in the means of the two measurement series,

$$m_{\text{fest}} = m_{\text{aft}} - m_{\text{Befm}}$$

Confidence interval of the estimated mass of the film

In a first approach, we estimate an error range $\pm m_{\text{Fc0}}$ of the film's mass with the rule for sums and differences as square root of the sum of the squares of the C -spec errors of the summands:

$$m_{\text{Fc0}} = \sqrt{m_{\text{BefCe}}^2 + m_{\text{AftCe}}^2} \quad (9.24)$$

We know from Exercise 9.8, that the confidence level of this estimate does not necessarily correspond to the confidence level Con of the estimated masses of the substrate before and after coating. In order to correct that, we have to multiply m_{Fc0} with a t factor that is to be determined through a statistical simulation.

For this simulation, we take as parameters the estimated masses m_{BefM} of the uncoated substrate and m_{Fest} of the film, and a noise level m_{Ns} estimated as the standard deviation of the measurement series of the uncoated and coated substrate. In this exercise, these values are the result of a simulation but in laboratory praxis such are the results of real measurements.

The number of trials in the simulation must be big enough so that the fluctuations of the hit rate for the same t value are sufficiently small to avoid too large jumping of the hit rate and assure improvement in the iterative adaptation of the t value.

Rounding the numerical results to relevant digits

The layer's mass and its C -spec error are to be determined from the simulated measurement data and written as a final result with rounded numbers. To give an example: If the layer's mass is determined as the difference of the weighings to be 0.0009748 g with an error of 6.57×10^{-5} , the final result is $9.7(7) \times 10^{-4}$ g or $9.7 \pm 0.7 \times 10^{-4}$ g:

$$\begin{array}{r} 0.0009748 \quad 0.00097 \\ \pm 0.0000657 \quad \pm 0.00007 \end{array}$$

This means that the result can only be expressed sensibly with two digits and that the uncertainty in the last digit is ± 7 . An alternative notation is $0.97(7)$ mg. For completeness, the *confidence level of the reported error* must be noted.

Questions

You get the following results for a single measurand: Mean value 0.0010818 g, standard error of the mean value 4×10^{-5} g. How do you report the final result?³⁴

Is 1.033 ± 0.037 more precise than 1.03 ± 0.04 ?³⁵

“Official” provisional measurement result

In the report of the measurement results, you should state the result, here the film’s estimated mass, and its *C-spec error*, and the level of confidence:

$$\text{Mass of the film (C = 0.683)} = 1.01 \pm 0.06\text{mg}$$

If you do not calculate the *C-spec error*, you must state all values necessary for the simulation.

Question

Which results of a measurement series for determining the film’s mass do you have to state if you do not calculate the *C-spec error*?³⁶

Syntactical differences between Excel and Python

In EXCEL, [= ROUND(NORM.INV(RAND());MSUB;MNS);DSP] is written into $n = 5$ cells with range name m_{Bef} , whereas in PYTHON, there is only one statement:

$$m_{\text{Bef}} = \text{np.round}(m_{\text{Sub}} + m_{\text{Ns}} * \text{npr.randn}(n_{\text{Bef}}), \text{dsp})$$

using the function `npr.randn` to generate normally distributed random numbers that have to be multiplied by m_{Ns} to get the right scale of the measurement noise.

EXCEL uses the function `STD.S` to get the variance of a sample, whereas Python uses `np.std` with `ddof`, the deduction of the degrees of freedom, to be specified as a key argument; `ddof = 1`, in our case, e.g., `np.std(mBef, ddof = 1)`.

³⁴ 1.08×10^{-3} g; 0.04×10^{-3} g \rightarrow (1.08 ± 0.04) mg; uncertainty of measurement given as one standard error. Ψ *Two inside and one out of* applies if enough repetitions of the measurement have been made or the error is extended by a t factor.

³⁵ No, the standard error is also only an estimate and is affected by statistical error.

³⁶ (1) Estimated masses of the uncoated substrate and the film, (2) the noise level of the measurement series. These are the parameters necessary to perform a simulation for getting the *C-spec error*.

9.9.2 Data Structure and Nomenclature

m_{Sub}	true mass of the substrate
m_F	true mass of the film
m_{Ns}	measurement noise (standard deviation of a normal distribution)
dsp	display precision of the scales
n_{Bef}, n_{Aft}	number of measurements before and after coating
m_{Bef}	array of the results of the weighing before coating
m_{Aft}	array of the results of the weighing after coating
m_{BefM}, m_{AftM}	means of m_{Bef} and m_{Aft}
m_{BefSe}, m_{AftSe}	standard errors of m_{Bef} and m_{Aft}
Con	confidence level
m_{BefCe}, m_{AftCe}	C-spec errors of m_{BefM} and m_{AftM} for Con
m_{Fest}	Estimated mass of the film, $= m_{AftM} - m_{BefM}$
m_{FSe}	standard error of m_{Fest}
m_{Fce}	C-spec error of m_{Fest} .

9.9.3 Spreadsheet Solution

Simulation of the weighing process

The parameters of the exercise are specified and the measurement series simulated in Fig. 9.21 (S). All values are stored in named cells, with the names listed in column A and in D1:E1. The uncoated substrate’s mass is measured as $m_{Bef} = m_{Sub} + noise$, that of the coated substrate as $m_{sub} + m_F + noise$. The noisy measurement is simulated with the spreadsheet function $NORM.INV(RAND());MSUB;MNS$, with mean value m_{Sub} (or $m_{Sub} + m_F$ for the coated substrate) and standard deviation m_{Ns} .

Evaluation of the simulated or real-world weighing process

The calculation with these parameters is shown in Fig. 9.22 (S). The values in D:E could also be the results of real measurement. The following evaluation and simulation would be the same.

	A	B	C	D	E	F
1	mSub	1 g		mBef	mAft	
2	mF	1.0E-03 g		0.999934	1.000833	
3	mNs	1.0E-04 g		1.000047	$1.001021 = ROUND(NORM.INV(RAND());mSub;mNs);dsp)$	
4				0.999899	1.001115	
5	dsp	6		0.999952	1.000912 $=ROUND(NORM.INV(RAND());mSub+mF;mNs);dsp)$	
6				1.000154	1.000854	

Fig. 9.21 (S) A:C, “True” values of the masses of the substrate and the film and an assumed value for the noise of the measurement process; dsp = number of digits in the scale display; D:E, simulated results of measurement series for the uncoated and coated substrate.

	H	I	J	K	L	M
1	mBefM	0.999997	=AVERAGE(mBef)			
2	mAftM	1.000947	=AVERAGE(mAft)			
3	nBef	5	=COUNT(mBef)	Con	0.683	
4	nAft	5	=COUNT(mAft)	tBef	1.14	=T.INV.2T(1-Con;nBef-1)
5				tAft	1.14	=T.INV.2T(1-Con;nAft-1)
6	mBefSe	4.62E-05	=STDEV.S(mBef)/SQRT(nBef)	mBefCe	5.28E-05	=mBefSe*tBef
7	mAftSe	5.32E-05	=STDEV.S(mAft)/SQRT(nAft)	mAftCe	6.08E-05	=mAftSe*tAft
8	noise	1.11E-04	=(STDEV.S(mBef)+STDEV.S(mAft))/2			

Fig. 9.22 (S) Evaluation of five weighing processes each of the uncoated and the coated substrate, here simulated in Fig. 9.21 (S). The results m_{BefSe} and m_{AftSe} are standard errors. The report could also be the result of a real-world experiment

	O	P	Q	R	S	T
1	varBef	2.79E-09	=mBefCe^2	TRUE	=AND(mFest-mFce<mF;mF<mFest+mFce)	
2	varAft	3.69E-09	=mAftCe^2	HitRate	673	From procedure
3	mFc.0	8.05E-05	=SQRT(varBef+varAft)	Trials	1000	
4	tAd	0.891		tHR	1.12	=T.INV.2T(1-HitRate/Trials;4)
5	mFce	7.17E-05	=mFc.0*tAd	tF	1.14	=T.INV.2T(1-Con;4)
6	mFest	9.50E-04	=mAftM-mBefM	tAd.New	0.91	=tAd*tF/tHR

Fig. 9.23 (S) Setting an error m_{Fce} and determining its hit rate and therefrom its t factor t_{HR} . The t factor t_F for the confidence level Con is calculated in S5

Questions

- Why are the measurement results rounded to six decimal places in Fig. 9.21 (S)?³⁷
- Which parameter in Fig. 9.22 (S) characterizes the measuring accuracy of the weighing process?³⁸

Confidence interval by statistical simulation

In Fig. 9.23 (S), we first calculate an error m_{Fc0} for the film mass (P3) with the rule of error propagation for sums and differences and then multiply it with a tentative t -value t_{Ad} to calculate a C-spec error m_{Fce} (P5).

In the beginning, we had set $t_{Ad} = 1$ and obtained a hit rate 731 for 1000 trials, more than the 683 expected from the confidence level Con . For the first iteration, we replaced t_{Ad} with t_{AdNew} to get a hit rate of 673, close enough to the expected 683 to state the resulting $m_{Fce} = 7E-05$ as the C-spec error for the confidence level $Con = 0.683$. The old tentative C-spec error is multiplied by the ratio of the t value t_F for the desired confidence level and the t value for the current hit rate.

³⁷ This corresponds to the *display* accuracy of the balance.

³⁸ The accuracy of the current measuring process is characterized by the noise during weighing. In the case of Fig. 9.22 (S) I8, it is 1.11E-4 g, estimated as the average standard deviation of the measurement series.

Table 9.19 Parameters chosen for the simulation

1	mSub=1.00	#[g]	True mass of the substrate
2	mF=1.0e-3	#[g]	True mass of the film
3	mNs=1.0e-4	#[g]	Measurement noise
4	dsp=6	#	Number of displayed digits
5	nBef=5	#	Number of measurements
6	nAft=5		
7			
8	Con=0.683	#	Confidence level

9.9.4 Python Program

The parameters for the simulation of the weighings are specified in Table 9.19.

The main program to determine the hit rates of the confidence intervals is given in Table 9.20. It is essentially a for-loop calling a function *InRange* which takes the *t* factor t_{Ad} as input and returns a Boolean *inRa* that states whether the confidence interval captures the true value, and the C-spec error m_{Fce} .

The function *InRange* is given in Table 9.21. It recurs to two functions *MassBefore* and *MassAfter* reported in Table 9.22. *MassBefore* simulates the measuring process and returns the estimated mass m_{BefM} of the substrate before coating together with its C-spec error m_{BefC} . *MassAfter* does the equivalent for the mass of the substrate after coating.

The result of the for-loop is a hit rate that is used to calculate a new *t* value t_{AdNew} expected to be related to the specified confidence level. For the next iteration, the value of t_{AdNew} is inserted into line 3 by hand.

The final result is rounded with the function *FinRes* from Sect. 9.3.3 to the relevant number of digits (see Table 9.23).

Table 9.20 Main program to determine the hit rate and the C-adjusted *t* value t_{AdNew} . For a new run to iteratively improve *t*, insert t_{AdNew} manually in line 3

```

1 Trials=10000
2 HitRate=0
3 tAd=1 #To be adjusted for chosen C
4 for i in range(Trials):
5     inRa,mFce=InRange(tAd) #Defined in table below
6     if inRa==True: HitRate+=1
7
8 tF=sct.t.ppf(1-(1-Con)/2,nAft-1) #Target value of t
9 tHR=sct.t.ppf(1-(1-HitRate/Trials)/2,nAft-1)
10 tAdNew=tAd*tF/tHR

```

tAd	1.00	HitRate	7171
tF	1.14		
tHR	1.24		
tAdNew	0.92	HitRate	6725

Table 9.21 Function to determine the C-spec error m_{Fce} of the film mass and the logical value whether the confidence interval captures the true value, recurs to functions in Table 9.22 that perform the simulations

```

11 def InRange(tAd):
12     #Estimate mass of the film!
13     mBefM,mBefCe=MassBefore()      #In table below
14     mAftM,mAftCe=MassAfter()      #In table below
15     mFest=mAftM-mBefM              #Film mass
16     varBef=mBefCe**2
17     varAft=mAftCe**2
18     mFc0=np.sqrt(varBef+varAft)
19     mFce=mFc0*tAd                  #C-spec error
20     inRa=mFest-mFce<mF<mFest+mFce
21     return inRa,mFce

```

Table 9.22 Simulation of the weighing of the substrate before and after being coated

```

1  import scipy.stats as sct
2
3  def MassBefore():
4      #Mass of the substrate before coating:
5      mBef=np.round(mSub+mNs*npr.randn(nBef),dsp)
6      mBefM=np.average(mBef)         #Mean
7      mBefSe=np.std(mBef,ddof=1)/np.sqrt(nBef)
8      tBef=sct.t.ppf(1-(1-Con)/2,nBef-1)
9      mBefCe=mBefSe*tBef             #C-spec error
10     return mBefM,mBefCe
11
12 def MassAfter():
13     #Mass of the substrate after coating:
14     mAft=np.round(mSub+mF+mNs*npr.randn(nAft),dsp)
15     mAftM=np.average(mAft)         #Mean
16     mAftSe=np.std(mAft,ddof=1)/np.sqrt(nAft)
17     tAft=sct.t.ppf(1-(1-Con)/2,nAft-1)
18     mAftCe=mAftSe*tAft            #C-spec error
19     return mAftM,mAftCe

```

mAftM	1.000913	mAftCe	8.06e-05
mBefM	0.999904	mBefCe	4.08e-05
mFest	0.001009		

Table 9.23 Rounding the result to the relevant number of digits with the function defined in Exercise 9.3

```

1  Result=FinRes('Mass of the film (C=0.683) ',
                mFest*1000,mFc*1000," mg")

```

Mass of the film (C=0.683) =1.01±0.06 mg
--

9.10 Questions and Tasks

Explain the broom rules:

1. Ψ Twice as good with four times the effort.
2. Ψ Two within and one out of.
3. Ψ Worse makes good even better.
4. Ψ Mostly, not always.
5. Ψ Report the C-spec errors but calculate with their squares!

Evaluation of a measurement series

A quantity x was measured 9 times, with the resulting mean value $x_m = 10$ and standard deviation of the nine individual measurements $x_{SD} = 1.8$.

6. Which spreadsheet and which Python function do you use to simulate this measurement series?
7. What is the standard error of the mean value?
8. A measurement series yields, as the mean value, $x_m = 7.12546 \times 10^4$ and, as the standard error of the mean value, $x_{Se} = 6.28743 \times 10^2$. How do you specify the measurement result correctly rounded?
9. Does the formula $x_1 = \text{ROUND}(0.847; 1) - \text{ROUND}(0.155; 1)$ yield the same value as $x_2 = \text{ROUND}(0.847 - 0.155; 1)$?
10. In a departing airplane, the acceleration a is measured with two different methods, ten times each. The measurement results are $a_1 = 20 \pm 1 \text{ m/s}^2$ and $a_2 = 21 \pm 0.5 \text{ m/s}^2$. Which value and which measurement uncertainty should the crew report to the ground station?

Error propagation

11. A quantity z is the difference of two quantities $z = s_1 - s_2$. The results of the summands are $s_1 = 10 \pm 2.24$; $s_2 = 20 \pm 2$. How large are the difference z and its error z_{Se} calculated from the standard errors of the summands?
12. A quantity is the product of two quantities, $p = p_1 \cdot p_2$. Measurements with many repetitions result in $p_1 = 10 \pm 1$ and $p_2 = 20 \pm 3.5$. How big are the product p and its error p_{Se} calculated from the standard errors of the factors?
13. What is the difference between the C-spec error of a quantity that is calculated as a function of several measurands and its error calculated from the standard error of the measurands?



We create points on user-specified functions, transform them into measurement points by adding noise to their y -values, and then fit appropriate trend lines to the noisy data. Confidence intervals of the trend lines' coefficients are obtained using t statistics. In doing so, we learn how far we can trust the parameters of the trend lines. We use functions for linear regression, `LINEST` of Excel and `OLS` of the `statsmodel` library of Python, and nonlinear regression, `OLVER` and `curve_fit`.

10.1 Introduction: Linear and Nonlinear Regression

Solutions of Exercises 10.2 (Excel), 10.3 (Python), 10.4 (Excel), and 10.6 (Python) can be found at the internet address: go.sn.pub/26leyH.

10.1.1 Straight Line Through Data Points by Sight

From the beginner's physics lab course, we are familiar with a procedure for getting physical parameters from measured data by suitably plotting them and drawing a straight line through the data points with a ruler. This way, the two characteristic parameters of a straight line, slope and y -axis intercept, can be obtained. Below are two examples.

- (a) The Curie–Weiss law for the temperature dependence of the magnetic susceptibility χ_m of a ferromagnet above the Curie–Weiss temperature Θ ,

$$\chi_m = \frac{C}{T - \theta} \quad (10.1)$$

leads to a linear plot of the reciprocal of the measured data, $1/\chi_m$, versus T

$$\frac{1}{\chi_m} = \frac{1}{C} \cdot T - \frac{\Theta}{C} \quad (10.2)$$

from which the parameters C (Curie–Weiss constant) and Θ can be obtained.

(b) The reaction rate R of a chemical reaction as a function of the absolute temperature T , with k being the Boltzmann constant

$$R = R_0 \cdot \exp\left(-\frac{E_0}{kT}\right) \quad (10.3)$$

is logarithmized and plotted over $1/T$ (Arrhenius plot):

$$\ln R = \ln R_0 - \left(\frac{E_0}{k}\right) \cdot \frac{1}{T} \quad (10.4)$$

from which the collision rate R_0 of the reactants and the activation energy E_0 of the reaction can be obtained.

These are examples of linear regression.

10.1.2 Multilinear Regression

One method for finding an optimum linear function through data points is linear regression. The linear regression models in EXCEL and PYTHON apply to functions with several independent variables x_i . They are based on the general *multilinear form*:

$$y = a + m_1x_1 + m_2x_2 + m_3x_3 + \dots \quad (10.5)$$

We will apply multilinear regression to points scattered around a straight line, a parabola $y = a + b \cdot x + c \cdot x^2$, and an exponential function $\ln(y) = \ln(A) + m \cdot x$.

Trend lines in charts, spreadsheets, Python

In EXCEL, you can insert trend lines of the linear, exponential, potential, logarithmic, polynomial, or power type into xy -diagrams (“scatter diagrams”). Coefficients of a multilinear trend line are obtained, together with their standard errors, with the LINEST function of EXCEL or the corresponding OLS (“ordinary least squares”) in Python.

Both methods, LINEST and `ols`, output a *standard error* for each coefficient whose exact meaning depends on the number of data points and the number of parameters estimated from the data set, generally speaking, on the degrees of freedom. In our exercises, we check, with multiple repetitions of a random experiment, whether we have correctly recognized the degrees of freedom and correctly set the confidence limits. If this is the case, our broom rule *Ψ Two within and one out of (the standard error range)* must hold.

This check is possible because our data points are generated with “true” values on a curve and newly blurred with random noise for every repetition; then, the trend parameters are again estimated from the noisy data set. *Ψ We know everything and play stupid.*

Procedure of a random experiment

In order to check whether the hit rate corresponds to an expected confidence level, we

- generate “true” data points (x, y) from x -values and a function $y = f(x)$,
- blur them with normally distributed noise,
- transform the noisy values into a linear form,
- calculate the coefficients of a linear trend line through the noisy data, together with their standard errors extended by a t -value,
- check whether the error ranges capture the true values,
- determine the hit rate, i.e., how often in a series of simulations the error range of the trend line’s coefficients does capture the true values.

10.1.3 Nonlinear Regression

If the measurements are subject to background noise, a constant must be added to the trend function. Then, no straight-line equations, such as those in Eq. 10.2 or Eq. 10.4, can be obtained by coordinate transformations. In these cases, nonlinear regression must be applied. Likewise, if the function itself is not linearizable by a coordinate transformation as, e.g., $y = \cos \omega_1 t + \cos \omega_2 t$.

In such cases, we generate the trend function to be adjusted with preselected parameters and vary them so that the function passes through the measured data. This can be done by hand and visual judgment, but also automatically with the EXCEL functions GOAL SEEK and SOLVER, or with the functions `curve_fit` from `scipy.stats` or `minimize` from the `scipy.optimize` library of Python.

For fitting trend lines to measurement data, there are special program packages, e.g., *Origin* for the general case, *SCOUT* for fitting model dielectric functions to optical spectra, and other programs for Rutherford backscatter spectra or impedance spectroscopy. However, it has proven to be very useful for improving their judgment when the students can organize simple adjustments in spreadsheets or computer programs themselves. This way, they are experiencing nonlinear regression behavior with examples in which they have all of the parameters in their own hands.

10.1.4 Coefficient of Determination R^2

Explained and residual variance

The coefficient of determination for any trend line is defined as

$$R^2 = \frac{\text{explained variance}}{\text{total variance}} = \frac{(v_{Tot} - v_{Res})}{v_{Tot}} \tag{10.6}$$

The terms in this formula are explained by means of Fig. 10.1 for a linear trend line through the data points y_{Ns} . In part **a**, the distances to the horizontal $y = y_m$ (mean y) are marked with vertical lines. The sum of the square of these distances is $v_{Tot} = 61.4$, entering Eq. 10.6 as *total variance* v_{Tot} of the data set. In part **b**, the trend line within the chart is plotted, together with its formula and the value of the coefficient of determination $R^2 = 0.71$ in the legend. Here, it is the distance of the data points to the trend line that is marked with vertical lines. The sum of the squares of these distances is $v_{res} = 17.9$, entering Eq. 10.6 as *residual variance* v_{Res} of the trendline.

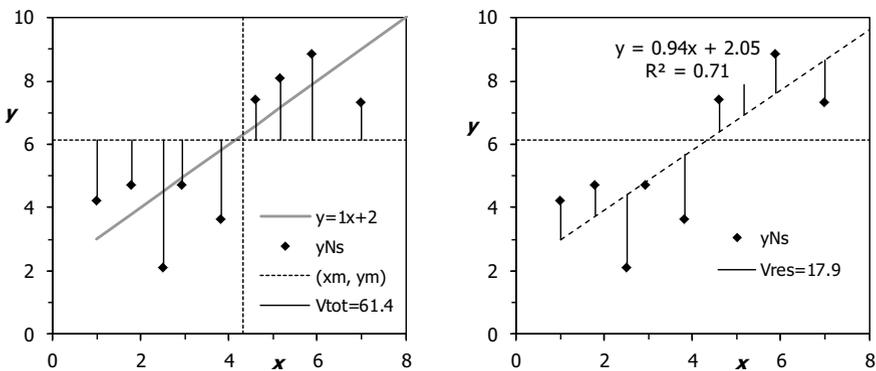


Fig. 10.1 **a** (left) Original curve and noisy data points; distances of the data points to the line $y = y_m$ are marked by vertical lines. **b** (right) The same data as in **(a)**, together with a linear trend line. Distances of the data points to the trend line are marked with vertical lines

The *variance explained by the trend line* is the difference between the residual variance and the total variance. In our example, Eq. 10.6 results in $R^2 = (61.4 - 17.9)/61.4 = 0.71$, the same value as reported in Fig. 10.1b.

Adjusted R^2

For the above calculation, the same degree of freedom has been supposed for v_{Tot} and v_{Res} , namely, $N - 1$, the number of data points minus 1. This is justified for v_{Tot} because only one parameter, the mean value, is estimated from the data set. Still, it is questionable for v_{Res} because more parameters describing the trend line are estimated from the data set, two for a straight line ($ddof = 2$ in Python). If the variances are calculated with the actual degree of freedom $N - ddof$, adjusted R^2 values result:

$$R_{adj}^2 = \frac{v_{Tot} - v_{Res} \cdot \left(\frac{N-1}{N-ddof} \right)}{v_{Tot}} \quad (10.7)$$

leading to an expression with the easily obtainable entities R^2 , N , and $ddof$.

$$R_{adj}^2 = 1 - (1 - R^2) \cdot \left(\frac{N - 1}{N - ddof} \right) \quad (10.8)$$

The term $ddof$ stands for “delta degrees of freedom”. For a set of N data points, we have the following degrees of freedom:

- $f = N - 1$ for the variance of the data set,
- $f = N - 2$ for the coefficients of a linear trend line (and an exponential trend line; see Exercise 10.4),
- $f = N - 3$ for the coefficients of a parabolic trend line.

10.1.5 C-spec Error with Iterative t Adaptation

Real-world data

We can apply the procedure of a random experiment presented in Sect. 10.1.2 (Ψ *We know everything and play stupid*) to real-world data when making two steps beforehand:

- taking the experimental trend line through the data points as the “true” line and
- estimate the noise level as standard deviation of the residuals. The residuals are the difference between the y values of the data points and the trend line.

For multilinear trend lines, we can check whether our assumptions about the confidence level are correct.

Experimental *t* factors

The solver function in EXCEL does not return standard errors of the optimized coefficients, contrary to the function `curve_fit` of `scipy.stats`. Therefore, in Sect. 10.7.6, we determine C-spec errors (“confidence-specified”) by a random experiment with tentative error ranges that are iteratively adapted to yield a hit rate that corresponds to the desired confidence level.

10.2 Linear Trend Line

We generate a set of points on a straight line (“generating line”), add normally distributed noise to their *y*-values, and then lay a straight trend line through these “data points”. The coefficients of such a line and their standard errors are determined with the spreadsheet function `LINEST`. In Python, this is achieved with the function `OLS` (“ordinary least squares”) of the library `Statsmodels`. The didactically most important message: *Ψ Two within and one out of*, meaning that in about 1/3 of the cases, the “true” coefficients of the generating line lie outside the *standard error range* of the estimated coefficients.

10.2.1 Creating Data Points and Evaluating Them

A linear trend line is to be laid through data points, minimizing the square deviation of the *y*-values. We generate the data points ourselves by randomly choosing *x*-values within a specific range, calculating the corresponding *y*-values on a straight-line (“true values”), and blurring them with normally distributed noise. Then, we estimate the coefficients of an optimum linear trend line through the data points and check whether the coefficients’ error ranges capture the true values.

Data points are generated

We generate a set of data points scattered around a straight line defined by

$$y = a + m \cdot x \quad (10.9)$$

with *m* being the slope and *a* the *y*-intercept. The independent variable *x* is randomly equi-distributed between 0 and x_{Max} . The values of *y* are blurred with a normally-distributed noise specified by a standard deviation *N_s*:

$$y_{N_s} = y + \textit{noise} \quad (10.10)$$

The formula for *noise* is:

- $N_s * \text{randn}(n)$ in Python, n being the size of the array generated, and
- $\text{NORM.INV}(\text{RAND}; 0; N_s)$ in EXCEL, to be written into a range comprising n cells.

In EXCEL, y_{N_s} could also be specified as a whole as

$$y_{N_s}(x) = \text{NORM}(\text{RAND}(); y; N_s) .$$

The set (x, y_{N_s}) is our data cloud.

Linear trend line through the data points

Our task is to determine a linear trend line

$$y_R = m_R \cdot x + a_R \tag{10.11}$$

through the data cloud, which minimizes the sum of the quadratic deviations of y_{N_s} to this line. This procedure is called *linear regression*.

Two examples are shown in Fig. 10.2, together with the linear trend line (by definition, going through the center (x_m, y_m) of the data cloud) and two straight lines with coefficients with + or - their standard errors. The equations and the coefficients of determination R^2 of the trend line are reported in the legend. The trend line is called the “linear regression line”. Its slope m_R is called the *regression coefficient*.

We start with the “true” function $y = 1x + 2$ to finally get the trend line for the noisy data:

$$y_r = 1.16x + 0.98 \text{ with } R^2 = 0.78 \text{ for a noise level } N_s = 1.4,$$

$$y_r = 0.89x + 2.42 \text{ with } R^2 = 0.28 \text{ for a noise level } N_s = 2.0.$$

Hit rates and degree of freedom *dof*

Statistical theory states that the “true” value is captured by the standard error range, plus-minus standard error from the estimated value, in 68.2% of the cases. This

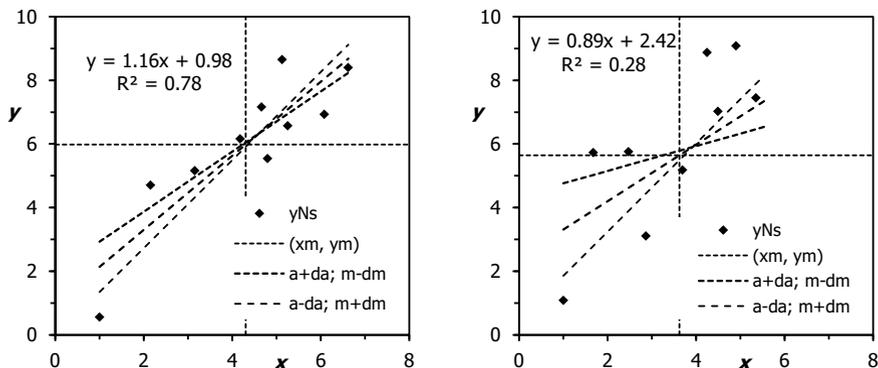


Fig. 10.2 Noisy data (diamonds) around the “true” line ($y = 2 + x$) with linear trend lines (solid) and lines for coefficients + or - their standard errors. **a** (left) Noise level 1.4. **b** (right) Noise level 2.0; noise level enters as standard deviation into a normal distribution

statement applies exactly only if the number of measurements goes to infinity, but our broom rule Ψ *Two within and one out of* applies as of 9 measuring points on a straight line.

► **Tim** Why now suddenly as of 9 measurement points? For measurement series, we have learned that the rule holds as of 8 measurements.

► **Mag** That's because of a different degree of freedom. For measurement series, the degree of freedom is the number of measurements -1, because only one parameter, the mean value, is estimated from the measurement series. For straight lines, however, two parameters are estimated, namely, slope m_R and y-axis intercept a_R . The degree of freedom $dof = 7$ is thus reached with 9 data points.

10.2.2 Data Structure and Nomenclature

x	set of independent variables starting at 1 and randomly increased by a value $< dx$
a, m	coefficients of a straight-line equation, "true values"
y	$y = a + m \cdot x$
N_s	noise level entering as the standard deviation in a normal distribution
y_{N_s}	y blurred with noise
x_m, y_m	average of the sets of x and y_{N_s}
a_R, m_R	coefficients estimated from a trend line through y_{N_s}
a_{Se}, m_{Se}	standard error of a_R and m_R
R^2	coefficient of determination
in_A, in_M	TRUE, if the error range captures the true value
$hitA, hitM$	number of hits, counting how often in_A, in_M are true.
dof	degrees of freedom, number of points - 2 for a linear trend line.

10.2.3 Spreadsheet Calculation with Linest

The program sketched in Sect. 10.2.1 is realized in the spreadsheet in Fig. 10.3 (S). The values of x start at 1.00 and progress with random interspaces $\leq dx$, as can be seen from the formula in A7 valid for A10. The 10 x -values in A9:A18 start at 1 and are randomly increased by $dx \cdot \text{RAND}()$. The corresponding y -values in column B are calculated with Eq. 10.9. A normally-distributed noise is finally added (y_{N_s} in column C, Eq. 10.10) to yield the "data points" presented in Fig. 10.2 with diamonds.

The coefficients m_R, a_R of the linear trend line through the data cloud are calculated with `LINEST` in G4:H6, together with their standard errors m_{Se}, a_{Se} , and the coefficient of determination R^2 . With in_M and in_A (F9:G9), we check whether the error range captures the true values of m and a . For in_M , we take the standard

	A	B	C	D	E	F	G	H	I	J
1	<i>y</i> intercept	a	2.00	y=1x+2			=LINEST(<i>yNs</i> ; <i>x</i> ; 1)			
2	<i>Slope</i>	m	1.00				mR	aR		
3	$0 \leq x \leq x_{Max}$	dx	1.40				mSe	aSe		
4	<i>Noise</i>	Ns	2.00			Estimated	0.864	3.362		
5						Standard error	0.194	0.883		
6						R²	0.713	1.304		
7	=A9+dx*RAND()									
8		x	y	yNs	xm	ym	in_m	in_a	hitM	hitA
9		1.00	3.00	4.55	4.03	6.85	TRUE	TRUE	6502	9474
10		1.67	3.67	5.98			t.95	2.31	=T.INV.2T(0.05;8)	
18		8.17	10.17	9.36						

Fig. 10.3 (S) A linear function $y = m \cdot x + a$ is blurred with noise. LINEST in G4:H6 determines the coefficients of a linear trend line through the data cloud. In F9:G9, we check whether the standard error range captures the true values of m or the t-extended error range the true values of a . *HitM* and *hitA* are returned from a VBA routine repeating the spreadsheet calculation 10,000 times

error m_{Se} whereas for in_A , we take the confidence interval for an error probability $E = 0.05$. *HitM* and *hitA* are returned from a VBA rep-log procedure running the spreadsheet calculation 10,000 times.

Question

Questions concerning Fig. 10.3 (S):

What is the formula for y_{Ns} , the y -values of the data points, in C9:C18?¹

Which parameter of the normal distribution is set by the parameter Ns in this formula?²

Trend line in the diagram

In Fig. 10.2, linear trend lines have been drawn through the “data points”, together with their equations and coefficients of determination R^2 . This is achieved by activating the data points and going through DESIGN/ADD CHART ELEMENT/TRENDLINE/LINEAR (EXCEL 2019). Proceed further with activating the trend line, going through FORMAT/SERIES 1 TRENDLINE 1/ and setting check-marks in DISPLAY EQUATION ON CHART and DISPLAY R- SQUARED VALUE ON CHART.

Each time the spreadsheet is modified (for example, by “clearing” the contents of an already empty cell), the noise is recalculated, and, accordingly, all data points in the graph change, yielding a new trend line.

¹ C9:D18 = [= y + NORM.INV (RAND(); 0; Ns)].

² Ns indicates the standard deviation of the normal distribution.

Linest as matrix function

What are the standard errors in the coefficients of the trend line? They are not reported in the legend for the trend line. To get them, we apply the `LINEST` spreadsheet function implemented in G4:H6 of Fig. 10.3 (S), performing linear regression. Its syntax is

$$\text{LINEST}(\text{known_ys}; [\text{known_xs}]; [\text{const}]; [\text{stats}]) \quad (10.12)$$

`LINEST` is a matrix function taking, as input, the Y -values (here, y_{Ns} from Fig. 10.3 (S)) and the X -values (here, x from Fig. 10.3 (S)) and outputting a matrix with 4 rows and 2 columns, of which only the first three rows are of interest to us here.

As a reminder: To insert a matrix function into a spreadsheet, first, mark the range into which the results are to be written; in Fig. 10.3 (S), it is the $(3R \times 2C)$ matrix G4:H6 (highlighted in grey). Then, write `= LINEST (`. After the parenthesis “(”, the function window opens, indicating the expected input, Eq. 10.12. After entering the ranges for the Y and X -values, you may enter control parameters that determine whether the trend line should go through the origin (`CONST = FALSE` or 0; default is `TRUE`) or whether statistical characteristics should be output (`STATS = TRUE` or 1; default is `FALSE`).

If `CONST = 0` or `FALSE`, then the straight line is drawn through the origin. If `STATS = TRUE`, additional regression characteristics are output, e.g., the standard errors of the coefficients and the coefficient of determination R^2 . In our case, the trend line should not be forced to go through zero; therefore, the corresponding site remains empty. Regression characteristics are to be output; therefore, we set a 1 as the last argument, thus the function is called as `LINEST (yNs; x;; 1)`.

► **Mag** How do you confirm the input of a matrix function?

► **Alac** With the magic chord!³

In the first row of the output matrix of `LINEST` are the estimates m_R and a_R for m and a (note the order!). In the second row are the standard errors m_{Se} and a_{Se} of these coefficients. In the third row on the left is the coefficient of determination R^2 , and on the right, the prediction error. The coefficients m_R and a_R , and R^2 correspond exactly to those of the trend line in a diagram with the same values for x and y_{Ns} .

The R^2 values reported in the figures and by `LINEST` are not adjusted for the reduced degree of freedom. To get R^2_{adj} , use Eq. 10.8!

Hit rates and degree of freedom

How often does the standard error range capture the true value?

With a rep-log procedure, we count how often in_m and in_a in F9:G9 are true. They are true if the respective error intervals $(m_R - m_E, m_R + m_E)$ and $(a_R - a_E,$

³ CTL + SHIFT + RETURN.

$a_R + a_E$) capture the true coefficients. We get hit rates of $hitM = 6502$ for 10,000 trials, closely corresponding to the theoretical value $C = 0.653$ for 8 degrees of freedom obtained with $1 - T.DIST.2T(1; 8) = 0.653$ ($2 * \text{scit.t.cdf}(1, 8) - 1$ in Python). The 8 degrees of freedom are given by 10, the number of data points, diminished by 2, because the 2 parameters m_R and a_R are estimated from the data set.

The hit rate for a is calculated with the standard error multiplied with 2.31, the t factor for $E = 0.05$. It corresponds closely to the expected 9500.

Rounding to a meaningful number of digits

In the diagrams, the coefficients of the trend line are given with 2 digits. By comparison with the standard errors from LINEST, we see that this assumes an accuracy that is greater than the uncertainty allows. The equation of the straight trend line is now, according to the results of LINEST in G4:H4 of Fig. 10.3 (S) and rounded to the number of significant digits,

$$y_R = (0.9 \pm 0.2)x + 3.4 \mp 0.9$$

We have rounded by inspection and written the result down by hand. However, this can also be achieved with mathematical formulas specified in the spreadsheet calculation in Exercise 9.3 or with the help of the user-defined function *FinRes* in Sects. 9.3.3 and 9.3.4. For more information about LINEST, see EXCEL HELP for this feature.

10.2.4 Python Program

The Python program in Table 10.1 generates 10 noisy data points (x, y_{Ns}) around a straight line (cell **a**) and evaluates them with `sm.OLS` (cell **b**). If OLS is called as `sm.OLS(yNs, x)`, a trend line $y_R = m_R x$ is fitted to the data. To get a fit to $y_R = a_R + m_R x$, we have to create a new set of independent variables `xx = sm.add_constant(x)`. The output in cell **c** is arranged so that it resembles the output of LINEST in EXCEL.

A figure corresponding to Fig. 10.2, as obtained with the program presented in Table 10.2.

Question

Where are the results in lines 13 and 14 in Table 10.1 needed? ⁴

In Fig. 10.4a, $R^2 = 0.74$ is reported. How big is R^2_{adj} ? ⁵

⁴ The coordinates x_m and y_m of the center of the data cloud are not needed in the presented program. They are necessary for a figure like Fig. 10.2.

⁵ With Eq. 10.8 and $N = 10$ and $\text{ddof} = 2$, we get $R^2_{adj} = 0.7075 \approx 0.71$.

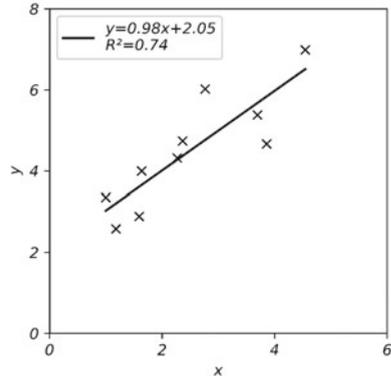
Table 10.1 **a** (top) Generation of noisy data points (x, y_{Ns}) around a straight line; **b** (bottom left) Linear regression line through the data points; **c** (bottom right) Results of the linear regression OLS

<pre> 1 import numpy.random as npr 2 import statsmodels.api as sm 3 a,m = 2.0,1.0 4 x=np.zeros(10) 5 dx=1.4 6 Ns=1.4 #Noise level 7 x[0]=1 8 for i in range(1,10): 9 x[i]=x[i-1]+dx*npr.rand() #Uneven spacing of x values 10 11 y=m*x+a 12 yNs=y+Ns*npr.randn(len(x)) </pre>																			
<pre> 13 xm=np.average(x) 14 ym=np.average(yNs) 15 xx=sm.add_constant(x) 16 model=sm.OLS(yNs,xx) 17 results=model.fit() 18 aR=results.params[0] #y=a+mx 19 mR=results.params[1] 20 aE=results.bse[0] #Errors 21 mE=results.bse[1] 22 r2=results.rsquared 23 r2_ad=results.rsquared_adj 24 in_a=(aR-aE < a) \ 25 and (a < aR+aE) 26 in_m=(mR-mE < m < mR+mE) </pre>	<pre> xx [[1.00 1.00] [1.00 1.18] . . . [1.00 3.86] [1.00 4.54]] </pre> <table> <thead> <tr> <th></th> <th>mR</th> <th>aR</th> </tr> </thead> <tbody> <tr> <td>est</td> <td>0.98</td> <td>2.05</td> </tr> <tr> <td>mE,aE</td> <td>0.21</td> <td>0.57</td> </tr> <tr> <td>r²</td> <td>0.74</td> <td></td> </tr> <tr> <td>r²_adj</td> <td>0.70</td> <td></td> </tr> <tr> <td>in_m,in_a</td> <td>True</td> <td>True</td> </tr> </tbody> </table>		mR	aR	est	0.98	2.05	mE,aE	0.21	0.57	r ²	0.74		r ² _adj	0.70		in_m,in_a	True	True
	mR	aR																	
est	0.98	2.05																	
mE,aE	0.21	0.57																	
r ²	0.74																		
r ² _adj	0.70																		
in_m,in_a	True	True																	

Table 10.2 Python program for plotting Fig. 10.4a, variables specified and results reported in Table 10.1

<pre> 1 lblT="y="+str(np.round(mR,2)) \ 2 +"x"+str(np.round(aR,2)) \ 3 +"\n"+"R²="+str(np.round(r2,2)) 4 5 FigStd('x',0,6,2,'y',0,8,2) 6 plt.plot(x,yNs,'kx') 7 ylin=x*mR+aR 8 plt.plot(x,ylin,'k-', label = lblT) 9 plt.legend() 10 plt.savefig("PhEx 9-2 trend line",dpi=1200) </pre>
--

Fig. 10.4 Chart with trend line and formula produced with the Python program in Table 10.2



10.3 Fitting a Polynomial Trend Line to Data Points with Multilinear Regression

We generate points on a parabola and add normally distributed noise to their y-values. A polynomial of the second degree is fitted to these simulated data, first, as a trend line within a diagram, and then with the spreadsheet function LINEST in EXCEL and with OSL of the statsmodel library of Python. These functions return, in addition to the coefficients of the polynomial, their standard errors. - Confidence intervals and confidence levels are interrelated through *Student's t* statistics. - Experience teaches us that measurement points must extend beyond the parabola vertex for its coefficients to be estimated reliably.

10.3.1 Introduction

Noisy data points around a parabola

The formula of a parabola is

$$\begin{aligned}
 y &= a + bx + cx^2 \\
 y &= ax^0 + bx^1 + cx^2
 \end{aligned}
 \tag{10.13}$$

The second variant emphasizes the powers of x .

Figure 10.5 shows 9 data points around a parabola generated with Eq. 10.13 and blurred with normally-distributed noise to yield values y_{Ns} , in **a** for a noise level (standard deviation of a normal distribution) of $N_s = 1$ and in **b** for $N_s = 2$.

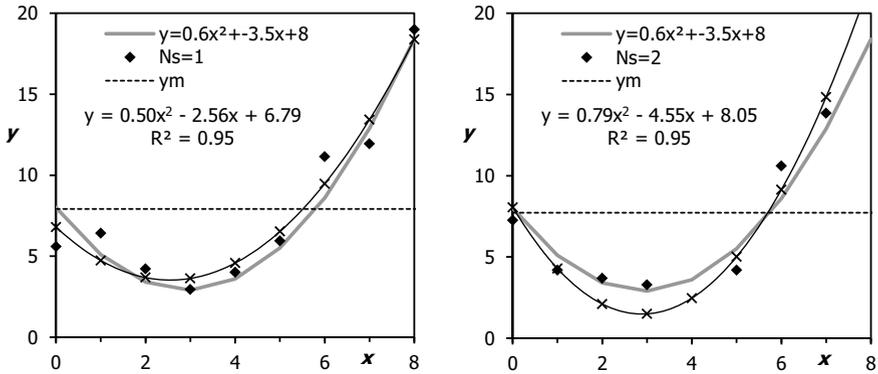


Fig. 10.5 Nine data points around a parabola (grey lines), together with polynomial trend lines (thin black lines) of degree 2. **a** (left) Noise level $N_s = 2$. **b** (right) Noise level $N_s = 4$

Trend parabola

Trend parabolas $y_R = a_R \cdot x^0 + b_R \cdot x^1 + c_R \cdot x^2$ are obtained by multilinear regression, in which the function is regarded as linearly depending on three variables x^0 , x^1 and x^2 that formally have to be stacked together in a matrix with three columns, e.g., in Python, with `xxx = np.array([1, x, x^2])`. The multilinear regression functions return the values a_R , b_R , c_R , together with their standard errors a_{Se} , b_{Se} , c_{Se} , and the coefficient of determination R^2 .

In EXCEL diagrams, polynomial trend lines can be fitted to the data, with their equations and R^2 being reported in its legend, as in Fig. 10.5. Attention: R^2 is unadjusted for a reduced degree of freedom.

Questions

Consider two data sets A and B, with the same number of points around the same parabola, but with the noise level of B being higher than that of A. What do you expect for the R^2 values for a fit with a parabola to A and B? Which one is higher?⁶

What is the degree of freedom for regression analysis of our parabola with 9 data points?⁷

The coefficient of determination R^2 is reported in the figures, together with the equation of the trend parabolas. Describe how R^2 is calculated with the data points (x, y_{N_s}) and the points (x, y_R) on the trend line! Consider the different variances, total, residual, explained!⁸

⁶ The R^2 for a fit to the data set with a lower noise level is expected to be higher, contrary to Fig. 10.5. Average values of 0.97 for **a** and 0.90 for **b** are to be expected.

⁷ Degree of freedom $dof = 9$ (number of data points) $- 3$ (parameters estimated from the data set) $= 6$.

⁸ Consult Eq. 10.6! Total variance $v_{Tot} = var(y_{N_s}) = (y_{N_s} - y_m) / dof$, residual variance $v_{Res} = sum(y_{N_s} - y_{Trend})^2 / dof$, $R^2 = (v_{Tot} - v_{Res}) / v_{Tot} = \text{explained variance} / \text{total variance}$.

What are the degrees of freedom to be set for adjusted R^2 ?⁹

Hit rates, with and without the t factor

The hit rates for, e.g., $(a_R - a_E < a < a_R + a_E)$ are around 0.64, as will be seen later in Fig. 10.7, and thus significantly smaller than suggested by our rule Ψ *Two within and one out of*. This is because the degree of freedom for the parabola with 9 (points) $- 3$ (coefficients of the parabola) $= 6$ is too small for this rule to apply. For a degree of freedom $f = 6$, a hit rate of 0.644 ($= 1 - \text{T.DIST.2 T}(1; 6)$ in EXCEL or $1 - (1 - 2 * \text{scf.t.cdf}(1, 6))$ in Python) is actually expected according to Student's t -distribution.

- ▶ **Tim** I'm confused. What should I state as the result?
- ▶ **Alac** Don't bother. Nobody knows about degrees of freedom and the t -value anyway.
- ▶ **Mag** If you specify the coefficients' standard error and the number of data points, an expert can derive statistically relevant statements. You can become such an expert yourself with our exercises.

The t -value for six degrees of freedom is $t = 1.091$ obtained with `T.INV.2S(0.318; 6)` (EXCEL) or `scf.t.ppf(1-0.318/2, 6)` (Python) for an error probability of 0.318 (confidence level 0.682) and $t = 2.45$ for an error probability of 0.05 (confidence level 0.95). Tests with an uncertainty given by the standard error multiplied by the t -values (" t -extended standard error") indeed give hit rates that correspond to the confidence levels.

- ▶ **Mag** What is the degree of freedom when calculating the variance of y_{Ns} or $(y_{Ns} - y_{Trend})$?
- ▶ **Alac** I would again say 6, but I'm not sure.
- ▶ **Tim** To calculate the variance of a set, we have to estimate the mean of this set. So, I conclude that the degree of freedom is $9 - 1 = 8$.
- ▶ **Mag** Tim is correct. Calculating $v_{Tot} = \sum (y_{Ns} - y_m)^2 / 8$ and $v_{Res} = \sum (y_{Ns} - y_R)^2 / 8$ yields the same value for $R^2 = (v_{Tot} - v_{Res}) / v_{Tot}$ as that reported in the figures or obtained with the multilinear regression functions.
- ▶ **Alac** OK, that's clear.

⁹ $Dof = N - 1$ for the total variance, $dof = N - 3$ for the residual variance, because 3 parameters are estimated from the data set.

► **Mag** But take care! For the adjusted R^2 value, you have to adjust the degrees of freedom, see Sect. 10.1.4!

Reliable fit of a parabola to the data points

In Fig. 10.6, trend parabolas are drawn through 9 data points. In Fig. 10.6a, the trend parabola differs more from the original parabola than in Fig. 10.6b, although $R^2 = 0.99$ is the same for both. The reason is that the characteristic feature of a parabola, the region around an extremum, is less well represented in the data points in **a**.

Questions

Why do the coefficients of the trend parabola fitted to the data points in Fig. 10.6a differ so much ($\approx 20\%$) from the true values (see the legend), although $R^2 = 0.99$ is achieved?¹⁰

Why does the trend curve in Fig. 10.6b capture the "true" parabola better than the one in Fig. 10.6a?¹¹

Higher-order polynomials

One can insert polynomials of higher degree as a trend line in diagrams. This is also possible with LINEST in the spreadsheet or with OLS in Python. You have to create column vectors x , x^2 , ..., and x^n , and enter the whole set as an argument for KNOWN_X'S in LINEST. For OLS, you have to extend the matrix with x^0 , i.e., a sequence of ones. This procedure can be generalized to other functions of x .

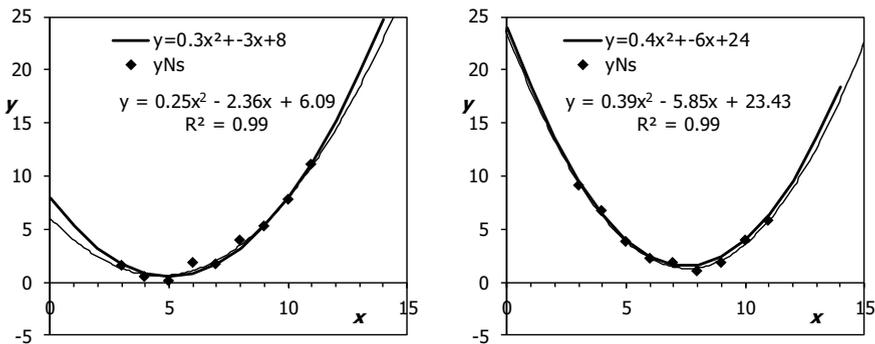


Fig. 10.6 **a** (left) Original parabola, simulated data points and trend line (second-degree polynomial) for unfavorably located data points. **b** (right) As in **a**, but for favorably located data points

¹⁰ The differences in the coefficients are so big because the data points do not capture the minimum well.

¹¹ In Fig. 10.6b, the characteristic property of the parabola, an extremum, is captured.

	A	B	C	D	E	F	G	H	I	J
1	a	8.00			=LINEST(yNs;xx;;1)					
2	b	-3.50			cR	bR	aR	Estimated		
3	c_	0.60			cSe	bSe	aSe	Stand. Error		
4		y=0.6x ² + -3.5x+8			0.66	-4.33	10.14	Estimated		
5	Ns	1.00			0.04	0.34	0.58	Stand. Error		
6		Ns=1		R ²	0.98	0.71	#N/A			
7	dx	1.00		N	NinC	NinB	NinA	t.32	1.09	=T.INV.2T(0.317;6)
8				1000	654	949	956	t.05	2.45	=T.INV.2T(0.05;6)
9	$=A11+dx$ $=x^2$ $=a+b*x+c_*x2p$ $=y+N_s*NORMINV(RAND();0;1)$ $=AND(cR-t.32*cSe<c_ ;c_<cR+t.32*cSe)$ $=AND(bR-t.05*bSe<b ;b<bR+t.05*bSe)$ $=AND(aR-t.05*aSe<a ;a<aR+t.05*aSe)$ $=AVERAGE(yNs)$									
10	x	x2p	y	yNs	inC	inB	inA	ym		
11	0.00	0.0	8.00	10.06	FALSE	FALSE	FALSE	7.86		
12	1.00	1.0	5.10	6.46						
19	8.00	64.0	18.40	18.30						

Fig. 10.7 (S) In columns A:D, data points around a parabola $y = a + b \cdot x + c \cdot x^2$ are created. In E4:G6, the noisy data $y_{Ns} = f(x)$ are fitted with a parabola. The results in E8:G8 are reported by a rep-log procedure that counts how often in_C , in_B , in_A are true for N repetitions of the experiment. The formulas for in_C , in_B , and in_A are different! N_{inC} is for $t = 1$, N_{inB} for $t = 1.09$, N_{inC} for $t = 2.45$. Data points and the trend line are shown in Fig. 10.5

10.3.2 Data Structure and Nomenclature

- a, b, c “true” coefficients of a parabola $y = a + b \cdot x + c \cdot x^2$
- x array of independent variables
- x_{2p} x to the second power
- xx $[x, x_{2p}]$ for EXCEL
- xxx $[1, x, x_{2p}]$ for Python
- y $y(x, x_{2p})$, function of two variables, values of the parabola for x and x_{2p}
- N_s noise level, entering as standard deviation in a normal distribution
- y_{Ns} y blurred with normally distributed noise
- a_R, b_R, c_R coefficients of a regression (trend) line through the data points (x, y_{Ns})
- m_{Se}, a_{Se}, c_{Se} standard error of m_R, a_R , and c_R
- in_A, in_B, in_C TRUE, if the error range captures the true value
- $N_{inA}, N_{inB}, N_{inC}$ number of hits, counting how often in_A, in_B, in_C are true
- dof degrees of freedom = number of points – 3 for a parabolic trend line
- t_{32}, t_{05} t values for confidence levels 0.68 and 0.95.

The argument structure is different for EXCEL and Python:

EXCEL `LINEST (yNs;xx;;1)`

Python `sm.OLS (yNs, xxxT)` with `xxxT` being the inverse of `xxx`.

10.3.3 Spreadsheet Solution

Generating the data set

In Fig. 10.7 (S), a spreadsheet layout for fitting a parabola trend line to data points is presented. The formula for y accesses the two variables $x_1 = x$ and $x_2 = x^2 (=x_{2p})$, as well as the constant a . The y -values are masked with Gaussian noise with standard deviation Ns and stored in y_{Ns} . Now, our data set is (x, y_{Ns}) .

Evaluating the data set

The noisy data set is evaluated in E4:G6 with

`LINEST(yNs;xx;;1)`.

to obtain the estimated coefficients c_R, b_R, a_R , together with their standard errors c_{Se}, b_{Se}, a_{Se} . Note the argument for the independent variables in the formula reported in E1! The independent variables x and x_{2p} have to be put together in one range, here, in a range A11:B19 named `xx`.

We calculate the error range of the coefficients for different confidence levels, i.e., we have to multiply the standard error with an appropriate t -value, and count, with a rep-log procedure, how often the error ranges capture the true values. The counts are stored in variables named $N_{inC}, N_{inB}, N_{inA}$.

The t -value for a confidence level of 0.95 is obtained with `T.INV(0.05; dof)`, where `dof` is the degree of freedom; see the formula in J8, valid for I8.

Question

concerning Fig. 10.7 (S):

Which are the independent variables in `LINEST`?¹²

Where are the t -values calculated, and what is their value for `inC, inB, inA`?

What is the confidence level of `inC, inB, inA`? Are the numbers of TRUE in $N_{inC}, N_{inB}, N_{inA}$ in agreement with expectation?¹³

¹² The independent variables are x in A11:A19 and x^2 in B11:B19, put together as argument `xx = A11:B19` for `LINEST`.

¹³ N_{inC} is calculated with `t.32`, valid for a confidence level of $1-0.317 = 0.683$ close to the empirical $N_{inC}/N = 0.654$; N_{inA} with `t.05` for a confidence level of $1-0.05 = 0.95$ close to $N_{inA}/N = 0.956$.

How do you use `LINEST` to determine the coefficients of a trend line for the function $y = a \cdot x_1^5 + b \cdot \log(x_2)$?¹⁴

Trend line in the diagram

To lay a parabola through the nine data points, `DESIGN/ADD CHART ELEMENT/TRENDLINE/ FORMAT TRENDLINE/TRENDLINE OPTIONS/` offers, among others, `POLYNOMIAL/ ORDER 2`. With appropriate clicks, the obtained regression equation and the coefficient of determination R^2 of the fit are reported in the diagram.

Hit rates

We apply a rep-log procedure to the spreadsheet calculation reading the values `TRUE` or `FALSE` in `E11:G11` N times and returning the counts into N_{inC} , N_{inB} , N_{inA} (`E8:G8`). The hit rates $6422/10000 = 0.642$, $6837/10000 = 0.684$, $9501/10000 = 0.95$ correspond to the confidence levels expected for the different t -values applied for in_C , in_B , in_A (see Footnote 13).

10.3.4 Python Solution

Generation of noisy data

In Table 10.3, noisy data y_{Ns} are generated around a parabola, the coefficients of which are specified all together in a list abc , not separately as a , b , c . The variables are specified as three row vectors x_0 , x , and x_{2P} , with x_0 being a list of ones and x_{2P} the squares of x ; they are gathered in a matrix xxx that is transposed to $xxxT$ to become a list of column vectors (see lines 7–11) required as entry into the OLS function, together with y_{Ns} .

Polynomial through the noisy data

In Table 10.4, the data set ($xxxT$, y_{Ns}) is evaluated with the OLS (ordinary least squares) function of `scipy.stats`, returning the results into $est = \text{sm.OLS}(\dots)$ consisting of

- a list of the estimated parameters (`est.params`),
- a list of their standard errors (`est.bse`),
- some characteristics of the fit (e.g., `est.rsquared`).

The results for a run are shown in the bottom cell of Table 10.4.

¹⁴ A two-column range is calculated for x_1^5 and $\log(x_2)$ and entered as `KNOWN_X'S` in `LINEST`.

Table 10.3 Python program, generating the data set $(xxxT, yNs)$ from (x, y)

```

1 import numpy as np
2 import numpy.random as npr
3
4 abc=[8.0,-3.5,0.6]           #Coeffs. of parabola
5 Ns=1
6 dx=1
7 x=np.linspace(0,8,9)
8 x2p=x**2
9 x0=np.ones(len(x))
10 xxx=np.array([x0,x,x2p])
11 xxxT=xxx.transpose(1,0)
12
13 y=abc[0]*x0+abc[1]*x+abc[2]*x2p #True parabola
14 yNs=y+Ns*npr.randn(len(y))     #Blurred with noise

```

yNs:

```
[ 7.44  6.15  1.76  5.14  3.42  5.13  8.74 12.32 16.80]
```

xxxT:

```
[[ 1.00  0.00  0.00]
 [ 1.00  1.00  1.00]
 [ 1.00  2.00  4.00]
 ...
 [ 1.00  6.00 36.00]
 [ 1.00  7.00 49.00]
 [ 1.00  8.00 64.00]]
```

Table 10.4 Evaluation of the data set $(xxxT, yNs)$ with OLS (ordinary least squares) of `scipy.stats`

```

15 import statsmodels.api as sm
16 import scipy.stats as sct
17
18 est=sm.OLS(yNs,xxxT).fit()
19 t05 = sct.t.ppf(0.975,6) #t value for E = 0.05
20 inABC=np.zeros(3)      #For the three coeffs.
21 for i in range(3):
22     inABC[i]=(est.params[i]-t05*est.bse[i] <
23              abc[i] < est.params[i]+t05*est.bse[i])
24              #In error range?
24 par=list(est.params)
25 err=list(est.bse)
26 rSq=est.rsquared

```

```

abc      8.00  -3.50  0.60
par      7.38  -3.39  0.60
err      0.87   0.51  0.06
rSq      0.97
t05      2.447
in        1      1      1

```

Table 10.5 Program lines of Tables 10.3 and 10.4 are assembled into a function returning the logical list in `ABC`

```

1  t05 = sct.t.ppf(0.975,6)           #t value for E=0.05
2  t32 = sct.t.ppf(0.8415,6)        #t value for E=0.318
3  t=[1.0,t32,t05]
4  def FitToPol():
5      yNs=y+Ns*npr.randn(len(y))
6      est=sm.OLS(yNs,xx).fit()      #Linear regression
7      inABC=np.zeros(3)
8      for i in range(3):            #Coeffs. in error range?
9          inABC[i]=(est.params[i]-t[i]*est.bse[i]
10             < abc[i] <
11             est.params[i]+t[i]*est.bse[i])
12      return inABC

```

Questions

How do you calculate Student's t value for a parabola's coefficients for a confidence level of $2/3$ and 12 data points?¹⁵

How is the regression line in Table 10.4 (coefficients in bottom cell) sensibly reported with uncertainties in the coefficients?¹⁶

Which lines in Tables 10.3 and 10.4 have to be gathered in a function, to be called in a loop to get hit rates for $E = 0.05$? Which variable has to be returned and then summed up in the main program?¹⁷

Hit rates

In Table 10.5, generation of y_{Ns} and evaluation of the data set are transferred into a function that returns the logical values `inABC`. This function is called in the main program of Table 10.6 in a loop in order to determine the hit rates. We have chosen three different t values for the three coefficients, 1, t_{32} , and t_{05} , corresponding to confidence levels of 0.644, 0.683 ($= 1 - (1 - 0.8415) * 2$), and 0.95, respectively.

In a loop in Table 10.6, we check whether the error ranges capture the true values. To do this efficiently, parameters concerning the coefficients a , b , c are put together in lists of shape comparable to the list in `est`:

- a , b , c in `abc`, see Table 10.3,
- the logical values `inA`, `inB`, `inC` in `inABC`, see Table 10.4,

¹⁵ `sct.t.ppf(1-1/6, 12-3) = 1.02`.

¹⁶ $y_R = 7.4(9) - 3.4(5)x + 0.60(6) \cdot x^2$.

¹⁷ Lines 14, 18–23 have to be gathered in a function; `inABC` has to be returned and summed up in the main program, see Table 10.5.

Table 10.6 Main program for getting hit rates for a, b, c ; $NinA = NinABC[0]$, etc.

13	<code>nRep=10000</code>				
14	<code>NinABC=np.zeros(3)</code>				
15	<code>for n in range(nRep):</code>				
16	<code>inABC=FitToPol()</code>	t	1.00	1.09	2.45
17	<code>NinABC[0]+=inABC[0]</code>	N	$NinA$	$NinB$	$NinC$
18	<code>NinABC[1]+=inABC[1]</code>	10000	6438	6888	9524
19	<code>NinABC[2]+=inABC[2]</code>				

and are addressed in Tables 10.4, 10.5 and 10.6 with their indices in these lists, e.g., `NinABC[0]+ = inABC[0]`.

The t -value for a confidence level of 0.95 is calculated according to

$$t_{05} = \text{sct.t.ppf}(0.975, 6).$$

`t.ppf` returns the value for a one-sided test. As we are checking whether the true values are larger than the lower error limit and smaller than the upper error limit, we have to enter an error probability $0.05/2 = 0.025$ instead of 0.05. The above formula returns $t_{05} = 2.45$.

The hit rates in the bottom cell of Table 10.6 are, with 0.64, 0.69, and 0.95, close to the theoretical confidence levels 0.642, 0.69, and 0.950 (see also Footnote 13).

10.4 Exponential Trend Line

We generate noisy data points (x, y_{Ns}) around an exponential and fit a trend line to the data, (1) as an exponential trend line in an EXCEL diagram, (2) with `LINEST`, and `OLS` of the `statsmodel` library of Python as a fit of a straight line through the logarithmized y_{Ns} data, and (3) with the spreadsheet matrix function `LOGEST` with a corrected standard error for the amplitude. For the exponential function to be clearly distinguishable from a parabola, the data points must cover a sufficiently large x -range.

10.4.1 Exponential and Logarithm

An exponential curve is defined by

$$y = g \cdot \exp(h \cdot x) \tag{10.14}$$

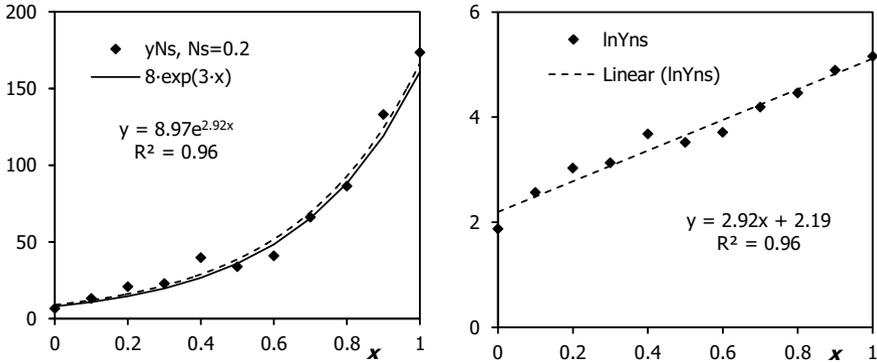


Fig. 10.8 **a** (left) Data points obtained from an exponential function blurred with noise on the y values, together with an exponential trend line. **b** (right) Natural logarithm of the values in **a**, together with a linear trend line

with g being the amplitude and h a characteristic growth parameter.

To simulate measured data, we add normally distributed noise to the function values y . A fit to the noisy data is most easily obtained by adding an exponential trend line in an EXCEL diagram, with the formula and coefficient of determination R^2 displayed as a legend. An example is shown in Fig. 10.8a.

Linear trend of the logarithmized data

By logarithmizing y , Eq. 10.14 can be converted into a straight-line equation of the form $z = a + m \cdot x$:

$$\ln(y) = \ln(g) + h \cdot x \tag{10.15}$$

with $z = \ln(y)$, $a = \ln(g)$, and $m = h$.

The logarithmized noisy y_{Ns} values of Fig. 10.8a are shown in Fig. 10.8b, together with a linear trend line. The output in the legend is essentially the same as in **a**, considering that $a = \exp(2.19) = 8.97$. This indicates that the exponential trend line is indeed obtained by a least-squares fit to the logarithmized data.

Question

When fitting an exponential trend line in a diagram, sometimes, the following error message appears: TRENDLINE CAN NOT BE CALCULATED FOR NEGATIVE VALUES OR FOR ZERO. When and why is that the case?¹⁸

¹⁸ The y_{Ns} values are logarithmized within the function that computes the exponential trend line. An error occurs when a y -value is negative or zero what occurs if the noise value is negative and its absolute value bigger than the signal.

Two aspects of this approach are problematic:

- The noise added to the exponential function is independent of the y -value, so that the logarithmized data scatter more around the trend line for smaller y -values, indicating that the noise is no longer evenly distributed along the straight line.
- The coefficient of determination is the same for both scales. This is problematic, because R^2 may be obtained as $(\text{Explained variance})/(\text{Total variance})$, and the explained variance should be different for different curves.

Uncertainty in the coefficients

We get the coefficients a and m of the linear trend, together with their standard errors, as usual, with `LINEST` and `OLS` and have to transform them into the coefficients g and h of the exponential function with $h = m$ and $g = \exp(a)$.

As the coefficient m is identical to h , so is the standard error $h_{Se} = m_{Se}$. The standard error g_{Se} , however, must be calculated according to the error-propagation law. We have $g = \exp(b)$ and $dg_E/db = \exp(b)$ so that

$$g_{Se} = \exp(b_R) \cdot b_{Se} \tag{10.16}$$

Due to the problems arising from logarithmizing the noisy data and deriving the uncertainty in the coefficient of the exponential from those of the linear trend of the logarithmized data, we have to be careful with the confidence level.

Influence of noise on the confidence level

For reasonable estimates of the standard error, the hit rate should be about 0.657 (Ψ *Two within and one out of*). With a rep-log procedure repeating the random experiment, “Eleven points around an exponential” 1000 times, we check how often this is actually the case for our experiment. The result can be seen in Fig. 10.9, below the label “const. noise”.

For the fitting of an exponential trend to 11 data points, the degree of freedom is $dof = 11 - 2 = 9$. This results in a confidence level for the standard error range of 0.657, calculated as `1-T.VERT.2S(1; 9)` (EXCEL) or `2 * sct.t.cdf(1, 9) - 1`

	const. noise				prop. noise	
	h=3	h=1.7	h=0.5	h=0.1	h=3	h=1.7
Total	1000	1000	1000	1000	1000	1000
Hits h	460	547	651	652	650	651
Hits g	383	463	601	641	643	659

Fig. 10.9 How often are the true values of h and g captured by the error range, for signals with constant noise level (*const. noise*) and noise proportional to the y -values of the exponential (*prop. noise*)?

(Python). With 1000 repetitions of the random experiment, one, therefore, expects that, in 657 cases, the error range of the estimated coefficients will capture the true value. However, in Fig. 10.9, the greater the coefficient h in the exponent, the less the theoretical value of 657 is reached, independent of the noise level.

Figure 10.8b indicates the reason for this. The deviations from the “true” curve are unevenly distributed on the log-scale. For large values of x , they are smaller, so that the trend line is closer to the “true” curve than for small values of x . This is a consequence of logarithmization. However, the unequal distribution of the deviations from the “true” values does not correspond to the mathematical model on which linear regression is based.

If we apply a noise proportional to the y -value, e.g., with

```
YNS = Y + NORM.INV(RAND();0;NS) * Y/4(EXCEL)
or yNs = y + Ns * randn(len(y)) * y/4 (Python),
```

then the deviations in the logarithmic values are more evenly distributed over the range of t (see Fig. 10.10b). Now, the deviations in the linear representation, Fig. 10.10a, are distributed unevenly. The number of hits in Fig. 10.9 (under “prop. noise”) deviates only slightly (less than 4%) from 657, also independent of the noise level.

► **Tim** When fitting an exponential trend, probably nothing fits at all; not even the noise is reliable.

► **Alac** I don’t see it that way. You can always give reasonable values for the coefficients.

► **Mag** In principle, that’s true. For low noise, the uncertainties and the differences among the various types of noise are not so big.

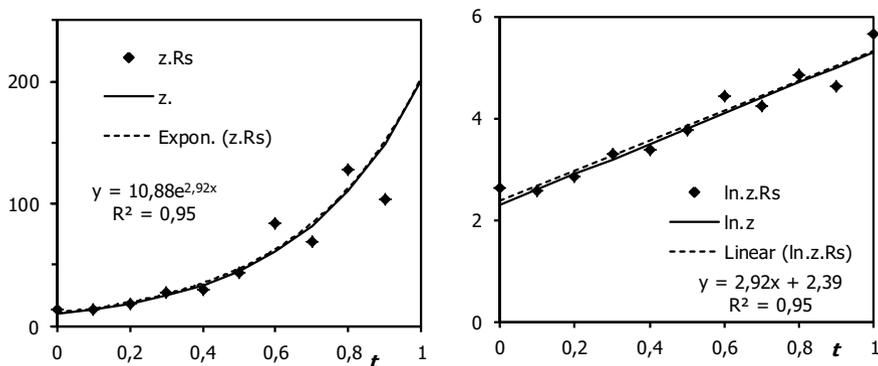


Fig. 10.10 a (left) Exponential curve with noise proportional to the y -value. b (right) Logarithmized data of a, together with a linear trend line

► **Tim** What exactly does “not so big” mean?

► **Alac** In a given case, we just figure it out with a simulation based on our method Ψ *We know everything and play stupid* to get the hit rate.

C-spec error by simulation-based t adaptation

We take the regression curve as the “true” curve, estimate the noise level from the data as well, and perform a statistical simulation with these parameters, similar to Sect. 10.7.6.

10.4.2 Exponential or Polynomial?

We fit both an exponential and a polynomial to the same set of data points generated around an exponential function. Figures 10.11a, b show two examples.

In Fig. 10.11a, both trend lines describe the experimental data equally well, with coefficients of determination R^2 of 0.9942 and 0.9953. When repeating the random experiment (new measurement points are randomly generated), the parabola sometimes gets an even higher R^2 value, even though the “true” original curve is exponential. There are not enough measurement points to clearly identify the exponential character. In Fig. 10.11b, a larger argument range was chosen. Even now, the parabolic trend line does not differ significantly from the measurement points; but it would predict completely wrong y -values for x greater than 35.

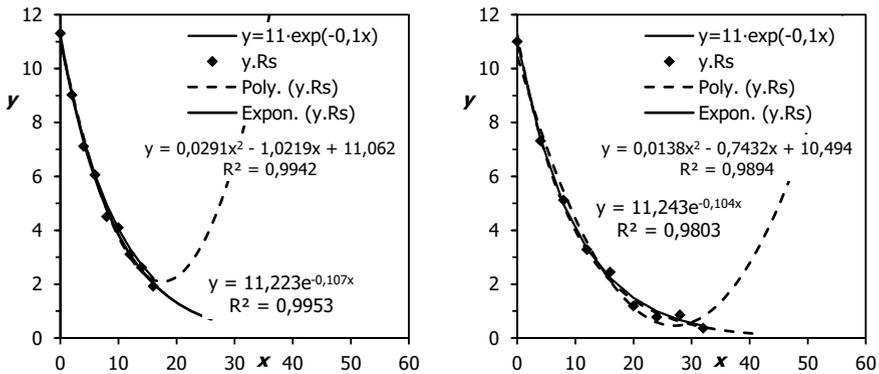


Fig. 10.11 **a** (left) Nine points near an exponential function; exponential and parabolic trend line. **b** (right) As in **a**, but with larger argument range

Question

Why is a parabola not an appropriate choice for the data in Fig. 10.11?¹⁹

- ▶ **Alac** In any case, the exponential trend line in Fig. 10.11a yields a good value of the decay constant.
- ▶ **Mag** If you are convinced that the underlying physical process is actually exponential, you may determine the decay constant from Fig. 10.11a.
- ▶ **Tim** What does it mean to be “convinced”?
- ▶ **Mag** That there is a theory of the physical process that predicts an exponential.
- ▶ **Alac** ...and then we remember “not disproved but also not proven” ...
- ▶ **Tim** ...but preliminarily accepted. This time, that makes sense to me.
- ▶ **Mag** That’s the best one can get out of the experiment. Discuss!

10.4.3 Data Structure and Nomenclature

x	series of independent variables
y	exponential values of x ; $y = g \cdot \exp(h \cdot x)$
N_s	noise level entering as a standard deviation into a normal distribution
y_{N_s}	y blurred with normally distributed noise
$\ln Y_{N_s}$	y_{N_s} logarithmized
m_R, a_R	coefficient of a straight trend line through $(x, \ln Y_{N_s})$
m_{Se}, a_{Se}	standard error of m_R, a_R
h_R, g_R	coefficients of the exponential trend line (regression line)
h_{Se}, g_{Se}	standard error of h_R, g_R , to be calculated from the results of the linear trend.

10.4.4 Python Program

The noisy data points around an exponential are generated in the top cell of Table 10.7 and logarithmized in the middle cell. The logarithmized data are fitted with a linear trendline, the coefficients thereof being presented in the bottom cell.

In Table 10.8, the coefficients of the linear trend fitted to the logarithmized

¹⁹ The characteristic of a parabola, its vertex, is not captured (see Sect. 10.2.2 on parabolas).

Table 10.7 **a** (top) Generation of noisy data points around an exponential; **b** (middle) Linear fit to the logarithmized data with OLS; **c** (bottom) Results of OLS

```

1  g,h= 10.0, 3.0           #y = g*exp(h*x)
2  Ns=4                    #Noise level
3  x=np.linspace(0,1,11,endpoint=True)
4  y=g*np.exp(h*x)
5  yNs=y+Ns*np.random.randn(len(x))
6  import statsmodels.api as sm
7
8  lnYns=np.log(yNs)       #Natural logarithm
9  xx=sm.add_constant(x)  #y intercept allowed
10 model=sm.OLS(lnYns,xx) #Linear regression
11 results=model.fit()
12 mR=results.params[1]    #Optimized coeffs of y=a+mx
13 aR=results.params[0]
14 mE=results.bse[1]      #Standard error
15 aE=results.bse[0]
16 r2=results.rsquared

      m      a
est.   2.78, 2.45
error  0.22, 0.13
r2=      0.95
    
```

Table 10.8 Transformation of the coefficients of the linear trend into those of an exponential trend

17	hR=mR		
18	hE=mE		
19	gR=np.exp(aR)	true	h g 3.00, 10.00
20	gE=aE*gR	est.	2.78, 11.61
21	inG=(gR-gE<g<gR+gR)	error	0.22, 1.49
22	inH=(hR-hE<h<hR+hE)	inG, inH	False True

data are transformed into coefficients of the exponential trend. The Boolean variables *inH* and *inG* check whether the estimated standard error ranges of the estimated coefficients capture the true values of *h* and *g*.

To get the hit rates, we have to integrate lines 5, 8–16 and 17–22 into a function that returns in_G and in_H and have a loop in the main program running over that function and counting the Trues in in_G and in_H .

10.4.5 Spreadsheet Solution

LINEST with the Logarithmized Data

In Fig. 10.12 (S), noisy data (x, y_{Ns}) are created, logarithmized, and fitted with a linear trend line with LINEST in E3:F5. The coefficients m_R and a_R of the linear trend and their standard errors are transformed in H3:I5 into the values of the exponential trend. H5:I5 contains the usual check as to whether the error ranges capture the true values. The numerical values of the coefficients h_R, g_R correspond to those of an exponential trend line in a figure.

We get the hit rates by applying a rep-log procedure that repeats the statistical experiment by writing a number into a cell to refresh the spreadsheet calculation and counts the TRUES in H5 and I5.

Exponential trend with the spreadsheet function logest

EXCEL provides a spreadsheet function LOGEST that determines the parameters of an exponential trend and their uncertainties, much like LINEST does for a linear trend. The equation of the curve to be fitted is

$$y = a \cdot o^x = a \cdot \exp(h)^x = a \cdot \exp(hx) \tag{10.17}$$

Comparing with Eq. 10.14, the following equations are valid:

$$h = \ln(o) \text{ and } g = a.$$

	A	B	C	D	E	F	G	H	I	J
1	g	8			mR	aR		hR	gR	
2	h	3			mSe	aSe		hSe	gSe	
3	Ns	0.2	Estimated		3.08	2.10		3.08	8.19	=EXP(aR)
4			Standard error		0.20	0.12		0.20	0.96	=aSe*EXP(aR)
5			R ²		0.96	0.21		TRUE	TRUE	
6					$=g*EXP(h*x)$ $=y+NORMINV(RAND();0;Ns)$ $=LN(YNs)$ \uparrow {=LINEST(LnYns;x;1)}					
7	x	y	yNs	lnYns						
8	0	8.00	7.25	1.98						0.657 =1-T.DIST.2T(1;9)
9	0.1	10.80	12.78	2.55						
18	1	160.68	207.88	5.34						

Fig. 10.12 (S) The noisy data y_{Ns} are created in columns A:D and logarithmized in D. A linear trend line through (x, y_{Ns}) is calculated with LINEST (formula in E6). The coefficients of the linear trend are transformed into those of the exponential trend; formula for H5 in H6

Fig. 10.13 (S) Use of the spreadsheet function LOGEST to determine the coefficients of an exponential trend and their standard errors; the matrix formula in E21 applies to E24:F26

	E	F	G	H	I
21	{=LOGEST(yNs;x;;1)}			=LN(oR)	
22	oR	gR		hR	gR
23	oSe	gSe		hSe	gSe
24	21.86	8.19		3.08	8.19
25	0.20	0.12		0.20	0.12
26	0.96	0.21			

For multiple independent variables, the function to be fitted is extended to

$$y = a \cdot o_1^{x_1} \cdot o_2^{x_2} \cdot \dots \quad (10.18)$$

In Fig. 10.13 (S), LOGEST is applied to the data of Fig. 10.12 (S). The same value $R^2 = 0.96$ is obtained for the coefficient of determination and the same values for $h_R = \ln(o_R) = 3.08$ and $g_R = 8.19$ (calculated in I24) are returned. However, LOGEST outputs a wrong standard error for g_R . Comparison with E4 in Fig. 10.12 (S) shows that the error of the ordinate intercept of the linear trend is reported as untransformed. The same transformation as in I4:J4 of Fig. 10.12, based on Eq. 10.16, should be done.

10.5 Solving Nonlinear Equations

This exercise introduces the nonlinear optimization algorithms SOLVER of EXCEL and minimize of the Python library `scipy.optimize`, determining, as an example, the intersections of a polynomial with a straight line. This technique is applied in nonlinear regression.

10.5.1 Intersection of Straight Lines with a Parabola

SOLVER of EXCEL is an analysis tool that varies up to 200 independent variables (“adjustable parameters”) so that the value of a target variable (must be a scalar) as a function of these parameters becomes optimum, maximum, minimum, or close to a given value, depending on the setting. The function minimize of the `scipy.optimize` library solves the same tasks. We apply it to determine the intersections of a parabola with straight lines. For the task in this exercise we apply the function `fsolve`, also of the `scipy.optimize` library, that finds the roots of an equation. Minimize of this library is applied in Exercise 10.6.

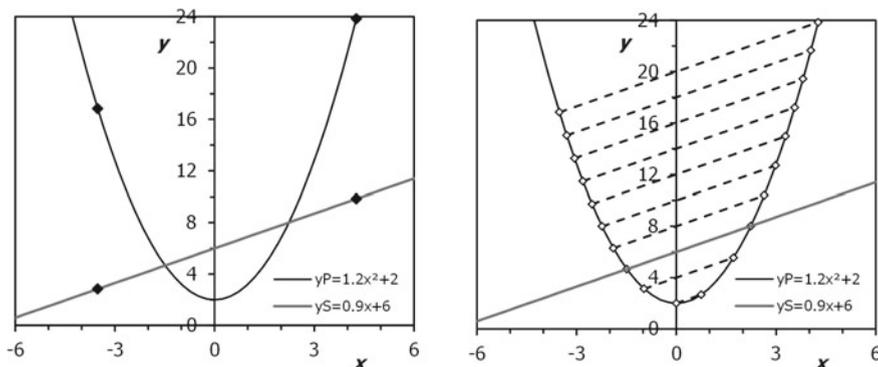


Fig. 10.14 **a** (left) The intersections of the straight line with the parabola are to be found. The currently selected x -values are not yet the solution. **b** (right) The intersections of ten straight lines with the parabola were determined with a solver algorithm

In Fig. 10.14a, a parabola and a straight line are shown whose intersections are to be found. This problem leads to a quadratic equation that can be solved analytically (pq solutions of the reduced quadratic equation). Therefore, the numerical method with a solver algorithm is not necessary here, but is very convenient, and also works for more complicated problems that cannot be solved analytically, e.g., for the intersections of a cosine function with a third-degree polynomial.

For the currently selected x -values, x_L (left of $x = 0$) and x_R (right of $x = 0$), the y -values on the straight line $y_L S$ and $y_R S$ are different from the corresponding $y_P L$ and $y_P R$ on the parabola. The x -values are to be modified so that the y -values become equal.

The equations for a straight line, and a parabola symmetric to $x = 0$ are

$$y_S = c_S + m_S \cdot x \tag{10.19}$$

$$y_P = b_P + a_P \cdot x^2 \tag{10.20}$$

We can manually adjust the x -values of the two points, e.g., with sliders, so that the y -values on the straight line and the parabola match. The same can be done with SOLVER by minimizing the quadratic deviations of the two y -values. Figure 10.14b shows the solutions for 10 straight lines with different ordinate intercept c_S . The two intersections for a particular straight line are found in one run by minimizing the sum of the two quadratic deviations of the y values. The 10 solutions are obtained in a rep-log procedure for the spreadsheet and in a for-loop in the main Python program.

10.5.2 Data Structure and Nomenclature

a_P, b_P coefficients of a parabola $y_P = a_P \cdot x^2 + b_P$

- c_S, m_S coefficients of a straight line $y_S = c_S + m_S \cdot x$
- x sequence of x -values (here 51 values between—6 and 6)
- y_P, y_S values of the parabola and the straight line for x
- x_L, x_R left and right horizontal positions of the intersections
- y_{LP}, y_{LS} y -values at x_L , P for parabola, S for straight line
- y_{RP}, y_{RS} y -values at x_R .

10.5.3 Spreadsheet Calculation

Layout

A spreadsheet layout for the task is presented in Fig. 10.15 (S). The graphs of the parabola $y_P(x)$ and the straight line $y_S(x)$ are calculated in D11:F61 for 51 points. The intersections are calculated in D7:F9, first, by adjusting x_L and x_R with sliders, and then with SOLVER. Later, we will use a rep-log procedure, triggered with the button “Intersections”, to get the intersections for the ten straight lines in Fig. 10.14b.

Activating Solver in Excel

The solver function must be activated with FILE/ OPTIONS/ ADD- INS/ SOLVER ADD-IN, then we find GO and click (see Fig. 10.16a). When we call DATA/SOLVER, the table in Fig. 10.16b pops up. The solver function optimizes the target cell’s value (SET OBJECTIVE) by varying the parameter cells’ values (BY CHANGING VARIABLE CELLS). The target cell is linked to parameter cells through a set of formulas. If the value of a parameter cell is changed, the value of the target cell also changes.

	A	B	C	D	E	F	G	H	I	J	K
1	aP	1.20	$y_P=1.2x^2+2$		mS	0.90	$y_S=0.9x+6$				
2	bP	2.00			cS	6.00					
3	dx	0.24									
4							3.9E+02				
5											
6											
7				xL	yLP	yLS					
8				xR	yRP	yRS					
9				x	yP	yS					
10											
11											
61											

Fig. 10.15 (S) The parameters for the parabola are in A1:B2, those for the straight line in E1:F2; arguments and function values in D11:F61; in the range D7:G9, the two intersection points are to be determined; in columns I to K, the intersection points for ten different straight lines with various y -axis intercepts c_S are stored. The button “Intersections” triggers the procedure *Intersectio* in Fig. 10.20 (P)

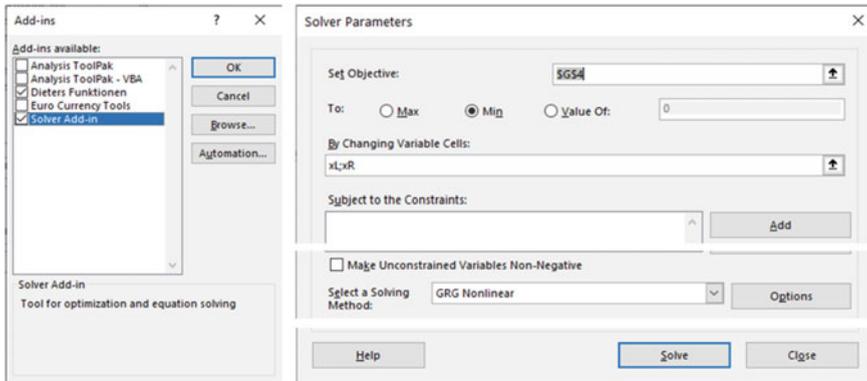


Fig. 10.16 a (left) Possible add-ins that can be activated in the EXCEL options. “Dieters Funktionen” are user-defined functions that have been saved as add-ins (see Sect. 4.9.1). b (right) Window after calling the solver function with DATA/SOLVER (in the Analysis Group); SET OBJECTIVE: the target value in the target cell is maximized (MAX), minimized (MIN), or adjusted to a given value (VALUE OF) by varying the values in the changeable cells (BY CHANGING VARIABLE CELLS)

Often, the option make unconstrained variables non-negative is activated as default. This check-mark must be removed, because the variable x to be optimized can also be negative for our task.

Solver determines intersections

The intersections are now determined with the SOLVER function. To do this, open the DATA tab (see Fig. 1.1 in Sect. 1.7) and click on the SOLVER button at the far right of the Analysis group, DATA/ SOLVER. A window opens, as in Fig. 10.16b. First, we take on only one intersection point and enter G7 (of Fig. 10.15 (S)), containing the left point’s square deviation of the y values of the parabola and the straight line, as the target cell in the SOLVER tab, and the x -value in cell D7 as the changeable cell. After pressing the SOLVE button, the two points on the parabola and the straight-line slide together.

We could now determine the right intersection in the same way; but we choose a different solution, with both intersections determined simultaneously, by summing up the square deviations of the two points in G4. We enter G4 as the target cell to be minimized, and the x -values in D7 and D9 as variable cells showing up with their names x_L and x_R . When we press SOLVE, the two intersections are determined together in one run.

1	Private Sub Intersect_Click()	Private Sub ScrollBar2_Change()	8
2	Call Intersectio	Cells(7, 4) = "(C7-500)/100" 'xL	9
3	End Sub	End Sub	10
4			11
5		Private Sub ScrollBar3_Change()	12
6		Cells(9, 4) = "(C9-500)/100" 'xR	13
7		End Sub	14

Fig. 10.17 (P) These procedures are triggered when, in Fig. 10.15 (S), the button “Intersections” is clicked (..._CLICK) or when one of the sliders is changed (..._CHANGE), respectively

Question

F4 of Fig. 10.15 (S) contains a function that calculates the quadratic deviation of the two intersections without accessing column G. Which function can that be?²⁰ See Section 5.10 (Mathematical and Trigonometric Functions) for advice!

► If the option MAKE UNCONSTRAINED VARIABLES NON- NEGATIVE is activated, SOLVER cannot find any negative x -values. This option must be unchecked if this is inadequate for the problem under consideration, as it is in this exercise. There are two ways to select the changeable cells:

- The solver algorithm varies the values in cells D7 and D9 and overwrites the original formula. However, throughout this exercise, we want to change these values with the slider again. To be able to do so, we reinsert the formula with a macro, that is always triggered when the slider is operated; see SUB SCROLLBAR2_CHANGE und SUB SCROLLBAR3_CHANGE in Fig. 10.17 (P).
- We can have SOLVER vary the values in C7 and C9. Then, the formulas in D7:D9 remain unchanged.

► **Mag** Let’s choose the first variant!

► **Alac** Why would we start there? The second variant is easier, because no macro is needed. Are we following the motto: Why be straightforward when we can complicate things?

► **Tim** Well, the first one teaches us how to link a macro to a control element.

► **Mag** Exactly, sometimes we learn via detours.

Sometimes, SOLVER declares the same x -value as the solution for the two intersections. In such cases, the initial x -values have been unfavorably chosen, e.g., both

²⁰ =SUMXMY2(E7:E9; F7:F9) “Sum x minus y squared”

1	Sub Macro1()	1
2	Sheets("calc").Select	2
3	SolverOk SetCell:="G4", MaxMinVal:=2, ValueOf:=0, ByChange:="D7,D9", _	3
4	Engine:=1, EngineDesc:="GRG Nonlinear"	4
5	SolverSolve	5
6	End Sub	6

Fig. 10.18 (P) Recorded macro when initializing the SOLVER function as in Fig. 10.16b

to be greater than zero. Such fallacies are possible in all optimization programs, because, primarily, only local optima are found.

- ▶ Use the sliders to change the changeable cells' initial values so that an approximate solution is reached before you start SOLVER!

Questions

Why does the formula in D7 or D9 in Fig. 10.15 (S) need to be re-entered after SOLVER has run?²¹

How do you re-enter the formula in D7 or D9?²²

According to Fig. 10.20, what are the initial values of x for a y -axis intercept of 20?²³

Calling Solver from a VBA procedure

The SOLVER function can be called from a macro. To get the corresponding commands, we turn DEVELOPER/RECORD MACRO on before calling SOLVER. The result for our example is presented in Fig. 10.18 (P).

Before starting the program, we must activate the reference to the solver function in the VBA editor with TOOLS/REFERENCES/SOLVER (see Fig. 10.19).

We insert the commands recorded in Fig. 10.18 (P) into the intended procedure as in Fig. 10.20 (P).

In a loop in SUB *Intersectio*, the ordinate intercept of the straight line is incremented from 2 to 20 in steps of 2 (line 11), and each time, SOLVERSOLVE calls the SOLVER function (line 16). The addition USERFINISH:=TRUE causes the solution proposed by SOLVER to be accepted immediately. If this addition is missing, a window pops up after each proposal of the SOLVER function in which the user can click OK.

²¹ Because, in the current variant of the optimization process, the formulas in these cells are overwritten by the SOLVER function.

²² By introducing macros like SUB SCROLLBAR2_CHANGE in Fig. 10.17 (P).

²³ The initial values are $x = -3$ and $+3$. They are the same for all y -axis intercepts, because they are always reset within the loop "cg=" in lines 13 and 14 in Fig. 10.20 (P). A better solution could be to take the previously optimized x -values as the start, because they are closer to the expected optimized value.

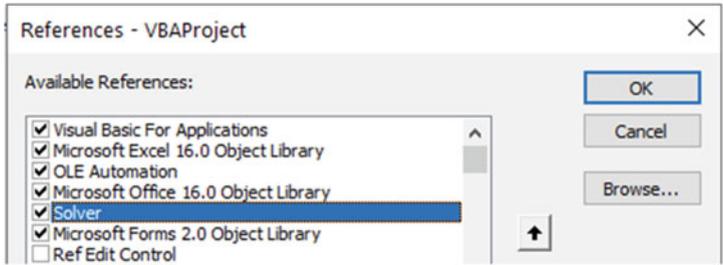


Fig. 10.19 Activating the SOLVER function in the VBA editor via TOOLS/REFERENCES

5	Sub Intersectio()	SolverSolve Userfinish:=True	16
6	r2 = 7 'Row index	Cells(r2, sp2) = cS	17
7	sp2 = 9 'Column index	'ordinate section of the straight line	18
8	Range("I7:K400").Clear	Cells(r2, sp2 + 1) = Cells(7, 4) 'xL	19
9	SolverOk SetCell:="\$G\$4", MaxMinVal:=2, _	Cells(r2, sp2 + 2) = Cells(7, 6) 'yLS	20
10	ValueOf:="0", ByChange:="\$D\$7;\$D\$9"	r2 = r2 + 1	21
11	For cS = 2 To 20 Step 2	Cells(r2, sp2 + 1) = Cells(9, 4) 'xR	22
12	Cells(2, 6) = cS	Cells(r2, sp2 + 2) = Cells(9, 6) 'yRS	23
13	Cells(7, 4) = -3 'left x to start	r2 = r2 + 2	24
14	Cells(9, 4) = 3 'right x to start	Next cS	25
15		End Sub	26

Fig. 10.20 (P) *SUB Intersectio*, working on Fig. 10.15 (S) specifies the parameters for the SOLVER function in lines 9 and 10 and selects the initial value c_S of the ordinate intercept of the straight line in the loop (FOR $c_S =$), calls the SOLVER function in line 16, and saves the coordinates of the intersections in the spreadsheet in the range below I6:K6, starting with $r_2 = 7$ (line 6). The specifications in lines 9 and 10 have to be defined only once, after which they apply to all subsequent calls of SOLVER

The initial values are $x = -3$ and $+3$. They are the same for all y -axis intercepts, as they are always reset within the loop (FOR $c_S =$) in Fig. 10.20 (P). If they were set before the loop, the previously optimized x -values would be the next y -axis intercept's initial values. This could be better, because the values are closer to the x -values expected for the next y -axis intercept.

Question

What effect does the instruction $r2 = r2 + 2$ in line 24 of Fig. 10.20 (P) have on range I7:K11 of Fig. 10.15 (S)?²⁴

²⁴ Empty cells are introduced between the coordinates of the points, e.g., J9:K9, so that the data are displayed in the diagram as separate points.

10.5.4 Python Program

Lambda functions

In Table 10.9, function tables of the parabola (x, y_{Px}) and the straight line (x, y_{Sx}) are set up using the lambda functions *Prb* and *Str*. A lambda function is a small function associated with a variable. It may take any number of arguments, but must only contain a single executable expression. Another example is *make_0* in Table 10.10.

Furthermore, in Table 10.9, two x values, x_L and x_R , are specified, together with their y -values y_{LP} and y_{RS} , serving later as initial values for the intersections of the parabola with the straight line.

Table 10.9 Function tables of the parabola and the straight line, and two points on each of them

```

1  x=np.linspace(-6,6,51,endpoint = True)
2  aP,bP=1.2, 2.0          #Parabola
3  Prb=lambda x: aP*x**2+bP
4  yPx=Prb(x)
5  mS,cS=0.9, 20.0        #Straight line
6  Str=lambda x: mS*x+cS
7  ySx=Str(x)
8
9  xL=-5                  #Initial x left
10 yLP,yLS=Prb(xL),Str(xL)
11 xR=5                   #Initial x right
12 yRP,yRS=Prb(xR),Str(xR)

```

Table 10.10 Plotting initial points and optimized intersections

```

1  FigStd('x',-6,6,2,'y',0,50,10)
2  plt.plot(x,yPx,'k--',label="parabola")
3  plt.plot(x,ySx,'k-',label="straight line, cS="+str(cS))
4  plt.plot([xL,xL,xR,xR],[yLS,yLP,yRS,yRP],'ks',
5           ms=5, fillstyle='none')          #Initial points
6
7  from scipy.optimize import fsolve
8  make_0 = lambda x : Prb(x)-Str(x)
9
10 xL=fsolve(make_0, xL)                    #Optimized left x
11 yLS=Str(xL)
12 xR=fsolve(make_0, xR)                    #Optimized right x
13 yRS=Str(xR)
14 plt.plot([xL,xR],[yLS,yRS],'ks',ms=5)  #ms, Marker size
15 plt.legend()

```

Function `fsolve`

In Table 10.10, first, the curves and the initial points are plotted, after which the intersections are determined by `fsolve` and then plotted in the same figure. The function `scipy.optimize.fsolve (func, x0, ...)` finds the roots of a function of the (in general non-linear) equations defined by $\text{func}(x) = 0$ given a starting estimate x_0 . In our case, the lambda function `make_0` is the root-defining function.

10.6 Temperature Dependence of the Saturation Magnetization of a Ferromagnet

The nonlinear Langevin equation relates the saturation magnetization of a ferromagnet with the temperature. We solve it with `SOLVER`, called from a macro, and with the function `minimize` of the `scipy.optimize` library of Python.

10.6.1 Langevin Function

Langevin equation

The Langevin equation describes the temperature dependence of the saturation magnetization M of a ferromagnet:

$$M = N\mu \cdot \tanh\left(\frac{\mu\lambda M}{k_B T}\right) \quad (10.21)$$

with:

- μ magnetic moment of a magnetic element, e.g., an electron,
- N density of magnetic elements,
- λ $\mu\lambda N =$ molecular field,
- k_B Boltzmann's constant,
- T absolute temperature.

The saturation magnetization M thus appears on both sides of the equation. This is physically justified because a magnetic dipole aligns itself in the entire field and also contributes to the entire field.

Equation 10.21 can be simplified by introducing reduced variables

$$m := \frac{M}{N} \mu \text{ and } t := \frac{k_B T}{N \mu^2 \lambda} \tag{10.22}$$

to

$$m = \tanh\left(\frac{m}{t}\right) \tag{10.23}$$

The reduced magnetization m is proportional to the magnetization M . The reduced temperature t is proportional to the temperature T . The solution $m = m(t)$ is called the *Langevin function*.

Graphical and numerical solution

The Langevin equation, Eq. 10.23, cannot be solved analytically. Still, it can be solved graphically by plotting $y = \tanh(m/t)$ for a given value of t as a function of m and by determining the point of intersection with the straight line $y = m$ (see Fig. 10.21a). It is solved numerically by keeping t fixed and varying m with a solver function so that the square deviation $(m - \tanh(m/t))^2$ is minimized.

We determine 18 points on the curve $m(t)$ by implementing two peculiarities: (a) all points are optimized simultaneously, and (b) all points are initially on a quarter circle.

As to (a): The Langevin equation can be solved for each reduced temperature t independently of the other temperatures. However, we solve it simultaneously for all points on the curve by adding up the quadratic deviations for the 18 cases and minimizing this sum by varying the 18 values of m .

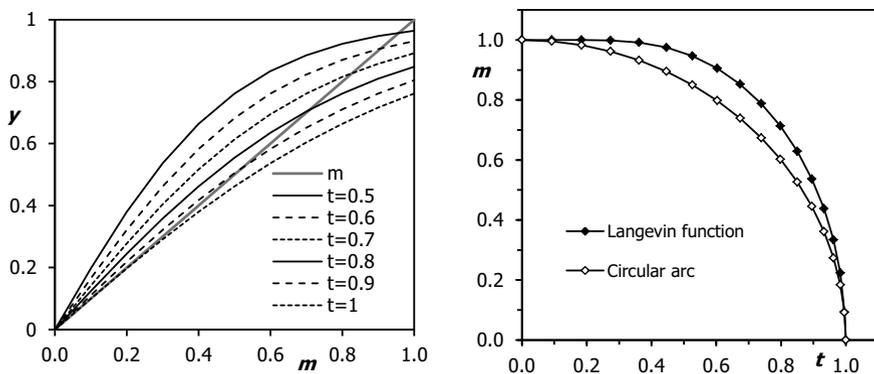


Fig. 10.21 a (left) Graphical solution of the Langevin equation; the y-axis is valid for m and $\tanh(m/t)$. b (right) Langevin function as a varied circular arc

As to (b): Since we expect a strongly bent curve, steeply going towards zero at $t = 1$, we select starting points (before minimizing) on a circular arc (see Fig. 10.21b).

Physical interpretation

The Langevin function in Fig. 10.21b is, with respect to the circular arc, flatter for small t and steeper for t approaching 1. The magnetic moments first stabilize each other in parallel alignment, but above a specific temperature, the order collapses.

10.6.2 Data Structure and Nomenclature

- ϕ array of 18 angles of a quarter circle
- t $\sin(\phi)$
- m reduced magnetization to be optimized
- m_C $\cos(\phi)$, initial values of m
- th $\text{tanhyp}(m/t)$, tangens hyperbolicus.

The entries $m(0) = 1$ and $m(1) = 0$ are held fixed; $th(0)$ is not defined.

10.6.3 Spreadsheet Layout

A spreadsheet solution of the Langevin equation is presented in Fig. 10.22 (S). The initial values m_C are on a circular arc and copied (with PASTE/PASTE VALUES) into m (E5:E22) before optimization. The solver algorithm is then used to vary these m -values so that they equal tanh according to the Langevin equation. If something goes wrong with the optimization, we can start again by copying m_C into m .

	A	B	C	D	E	F	G	H
1	0.0924	=PI()/17/2	Langevin function					
2	Circular arc							
3	=A5+\$A\$1	=SIN(phi)	=COS(phi)	=TANHYP(m/t)				=SUMXMY2(m;th)
4	phi	t	mC	m	th	devi		
5	0.00	0.000	1.000	1.000				
6	0.09	0.092	0.996	1.000	1.00	2.2E-08		
22	1.57	1.000	0.000	0.000	0.00			

Fig. 10.22 (S) Solving the Langevin equation; reduced magnetization: initial positions m_C on a circular arc, m after optimization; m is defined as [E6:E22]

Table 10.11 Optimizing the Langevin function

```

1  from scipy.optimize import minimize
2
3  phi=np.linspace(0.01,np.pi/2,18)
4  t=np.sin(phi)           #Values on quarter arc (not varied)
5  mC=np.cos(phi)         #Initial values on quarter arc
6
7  def objective(m):
8      th=np.tanh(m/t)
9      return sum((m-th)**2)
10
11 m=minimize(objective,mC,method='SLSQP')
12
13 FigStd('m',0,1.0,0.2,'y',0,1,0.2)
14 plt.plot(mC,t,'k+-')
15 plt.plot(m.x,t,'kx-')  #m.x, optimized values
16 plt.axis('scaled')
```

Question

What are the target (objective) and the adjustable variables for SOLVER in Fig. 10.22 (S)?²⁵

10.6.4 Python

In Table 10.11, the Langevin equation is solved with the scipy function minimize. The syntax is

```
m = minimize (objective,mC, method = 'SLSQP')
```

where

m list of variables to be varied; shape determined by *mC*

mC list of initial values for *m*

objective name of a function with *m* as the argument and a scalar as the output to be minimized

SLSQP *Sequential Least-Squares Programming*, type of solver to be applied.

The optimization works although *m*[0] and *m*[-1] are not excluded from the optimization because $m_C = \tanh(m_C/t)$ is already fulfilled.

²⁵ The target is *devi* (G6) and the variable cells are *m*, except its first value (E6:E22), because the magnetization *m* must be 1 at *t* = 0 and be kept fixed during optimization.

10.7 Fitting Gaussians to Spectral Lines with Nonlinear Regression

We fit a sum of two Gaussians to two overlapping EDX spectral lines, taking advantage of the additional knowledge that, for physical reasons, both lines must have the same width. The EXCEL tool is SOLVER; the Python function is `curve_fit` from the `scipy.stats` library. The C-spec errors of the SOLVER solution are obtained through simulation-based t adaptation (Student's t) using the parameters obtained from the fit.

10.7.1 Fitting the Sum of Two Gaussians to Data Points

In this exercise, we use the SOLVER tool of EXCEL and the function `curve_fit` from the `scipy.stats` library to fit functions with several fit parameters to measurement data, generated artificially in the by now well-proven manner.

Generation of two spectrally overlapping EDX signals

In Fig. 10.23a, a spectrum of an EDX analysis is shown. EDX is the abbreviation for “energy dispersive X-ray analysis”. It means the energy-resolved analysis of characteristic X-ray radiation with silicon detectors after excitation of the sample with an electron beam. The count rate y of a detector is recorded as a function of the energy x of the photons arriving at the detector.

Here, the mapped spectra are artificially generated, but are very similar to the actual data for a SrTiO₃ coated Si wafer. The generation is done by specifying two Gaussian bell curves with the parameters maximum value, centre, and standard deviation as $A = 600$, $x_A = 1.75$, $x_{Ad} = 0.05$ and $B = 150$, $x_B = 1.85$, $x_{Bd} = x_{Ad} = 0.05$, respectively, adding them together with a constant background noise level y_C .

General bell curve

The normal distribution for medium x_m and standard deviation x_{Sd} is defined as

$$N(x, x_m, x_{Sd}) = \frac{1}{x_{Sd}\sqrt{2\pi}} \cdot \exp\left(-\frac{1}{2}\left(\frac{x - x_m}{x_{Sd}}\right)^2\right) \quad (10.24)$$

The pre-factor is chosen so that the integral over the function is 1, as it must be for a probability distribution. The functions for normal distributions are used to model the spectra, and later to provide fits to the noisy data. To this end, it is advantageous that maximum A and width x_d be chosen independently of each other. The fit function

is now $f(x, x_m, x_{sd}) = A_N \cdot N(x, x_m, x_{sd})$. Consequently, A_N for $N(x, x_m, x_{sd})$ becomes

$$A_N = A \cdot x_{sd} \cdot \sqrt{2\pi} \quad (10.25)$$

When A and x_{sd} are independent of each other, the initial parameters can be set with the eye more intuitively.

Question

What is the area under the curve $f(x, x_m, x_{sd}) = A_N \cdot N(x, x_m, x_{sd})$?²⁶

Two different fits to the spectrum

The sum of two bell curves, G_a and G_b , and a constant background y_{Ar} are fitted to a simulated spectrum by means of a solver algorithm.

Two different fits to the same measurement data can be seen in Fig. 10.24. For Fig. 10.24a, the standard deviations of the two bell curves have both been varied to obtain an optimum fit. For Fig. 10.24b, the two standard deviations have been forced to be equal. Both fits appear to be equally good, with a very high $R^2 = 0.99$. Therefore, the fit is not unique.

Due to physical considerations, a fit with equal spectral width is more likely, because the width of the lines is determined by the resolving power of the detector and should be approximately constant within the considered energy range. So, we decide to interpret the experimental spectrum with the fit in Fig. 10.24b.

Caution when fitting with many parameters!

The described method of fitting curves with a set of parameters to measurement data is useful, but also dangerous. If you have enough fit parameters, you can fit almost any data series without the parameters having a meaningful physical interpretation. Gain experience! Our exercises offer an excellent practice field for this, as we invent “true” data ourselves.

- **Alac** Sure, according to the motto: *Ψ We know everything and play stupid.*
- **Tim** And thereby learn statistics.

Uncertainty in the coefficients

The function `curve_fit` of the `scipy.optimize` library returns, in addition to the parameters of the regression curve, the covariance matrix of the optimized coefficients from which the standard errors can be obtained as the square root of the diagonal elements.

²⁶ The area under this curve is A_N .

The solver function of EXCEL returns only the optimized parameters. The standard errors in the fitted curves' parameters can also be determined in a mathematically exact way in spreadsheets; see, e.g., E. Joseph Billo, *EXCEL for Chemists*, Wiley-VCH (1977) ISBN 0-471-18,896-4, Chap. 17 or John Wiley (2011) ISBN 978-0470-38,123-6, Chap. 15.

In practice, a coefficient can be changed individually by hand until there is no longer a good fit upon visual inspection. The deviation from the optimum value of the coefficient may then be reported as the uncertainty. An example can be seen in Fig. 10.23b, where the center of the left bell curve has been changed from 1.75 to 1.755. Evidently, the total curve G_{abc} no longer fits the data points. So, 0.005 is a rough estimate of the error in x_{Ar} .

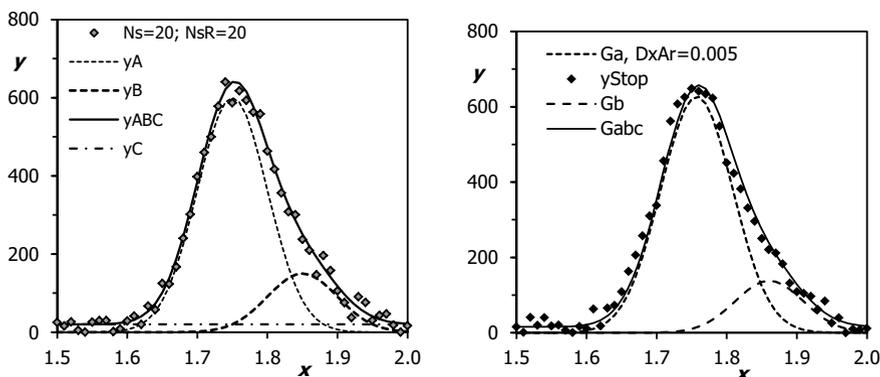


Fig. 10.23 **a** (left) Simulated EDX spectrum of a Si wafer coated with SrTiO₃; G1 and G2 are the underlying “true” spectral lines. **b** (right) A fit to noisy data with a sum of two bell curves and a constant; the left bell curve has been shifted by changing x_{Ar} by $\Delta x_{Ar} = 0.005$ to estimate the error in x_{Ar}

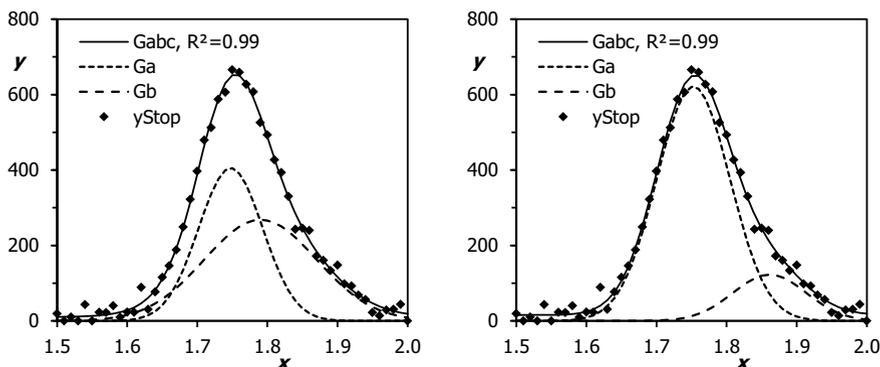


Fig. 10.24 Fit to an EDX spectrum with two Gaussian curves. **a** (left) The two standard deviations are varied independent of each other. **b** (right) The two standard deviations are forced to be equal

► **Tim** Is that allowed, and is it accurate enough?

► **Alac** Better a rough estimate of the uncertainties of the coefficients than none at all. In my experience, many labs engage in that practice. The researchers feel that a report of the optimum coefficients without uncertainty is, in any case, useless. Nevertheless, even this is done in many laboratories, as I have also observed.

► **Mag** Alac is right. However, we will do better with a simulation described in the next section by determining the hit rates for the assumed errors in the coefficients and obtaining the hit rates to the C-spec errors for a specified confidence level by simulation-based *t* adaptation.

10.7.2 C-spec Errors of the Coefficients by a Statistical Simulation

Figure 10.25 (S) lists standard errors of the parameters of the two bell curves fitted to noisy data, obtained with various methods. The values for Python from row 10 on are obtained as the square root of the diagonal elements of the covariance matrix that is returned by the function `curve_fit` of the `scipy.optimize` library.

The C-spec errors (“confidence-specified”) for EXCEL are obtained with a statistical simulation in six steps:

- (1) Starting from a fit to the spectrum and taking the resulting parameters of the optimized bell curves (row 2) as values of a “true” noise-less spectrum, as well as the residual noise of the fit as the “true” noise level. The residual noise is

	A	B	C	D	E	F	G
1	Ar	xAr	xAsdr	Br	xBr		
2	625	1.752	0.050	151	1.86		
3	DyAr	DxAr	DAsdr	DyBr	DxBr		
4	20	0.0050	0.0020	20	0.002	by eye	
5	10	0.0019	0.0013	16	0.008	D from row 4	
6	11	0.0019	0.0015	16	0.006		
7	10	0.0021	0.0015	16	0.007		
8	11	0.0016	0.0014	16	0.007	new D from row 7	
9							
10	Python						
11	10	0.0014	0.0014	13	0.006		
12	11	0.0013	0.0016	15	0.007		
13	12	0.0015	0.0015	19	0.005		

Fig. 10.25 (S) [A1:E2]: Results of a fit of two bell curves to experimental data; standard errors of the coefficients—obtained in EXCEL by eye (row 4) or by adjusting the values from row 4 with a *t*-value (rows 5–8)—in rows 11 to 13 with Python as the square root of the diagonal elements of the covariance matrix

calculated as the standard deviation of the difference between the noisy data and the fitted curve.

- (2) Assuming *initial errors* in the coefficients by changing them by hand until the deviation from the data becomes visible (Figs. 10.24 and 10.25 (S) row 4).
- (3) Simulating noisy data with the parameters from (1) and (2) to obtain a *fit to these data* with our model “two bell curves plus constant background noise”.
- (4) Repeating the statistical experiment (3) 100 times or more and so determining the *hit rates* for the initial error ranges from (2).
- (5) Estimating the confidence level from the hit rates in (4), and from that, the *t values* for the desired confidence level.
- (6) Adjusting the *C-spec errors* by dividing the errors assumed in (2) by the *t values* from (5).

Row 4 of Fig. 10.25 contains the initial values of the errors obtained by visual inspection. The next three rows report the results of statistical simulations with these errors. After that, the initial values of row 4 were replaced with those of the fit in row 7 to obtain the values in row 8. They are within the error range obtained with Python.

So, our method Ψ *We know everything and play stupid* is also successful for a task occurring in real laboratory life.

10.7.3 Data Structure and Nomenclature

Generation of a noisy spectrum

A, x_A, x_{Asd}	amplitude, center, standard deviation
y_A	bell curve with the above parameters
B, x_B, x_{Bsd}, y_B	second bell curve
y_C	background signal level
Ns	noise level
y_{ABCns}	$y_A + y_B + y_C + noise$, noisy spectrum.

Best fit to noisy spectrum

A_r, x_{Ar}, x_{Asdr}	amplitude, center, standard deviation of the fit curve
y_{Ar}	bell curve with the above parameters
$B_r, x_{Br}, x_{Bsdr}, y_{Br}$	bell curve with the above parameters
y_{Cr}	background signal level
y_{ABCnew}	$= y_{Ar} + y_{Br} + y_{Cr}$
Ns_R	noise level estimated as standard deviation of $(y_{ABCns} - y_{ABCnew})$.

Visually estimated standard errors of the fit parameters

$Dy_{Ar}, Dx_{Ar}, D_{Adr}$ estimated standard errors of A_r, x_{Ar}, x_{Adr}
 $Dy_{Br}, Dx_{Br}, D_{Bdr}$ estimated standard errors of B_r, x_{Br}, x_{Bdr} .

C-spec errors of the fit parameters obtained from a statistical experiment

p_{out} error rate for the estimated standard errors
 t Student's t value calculated from p_{out}
 st_E adapted standard error calculated from p_{out} and t , to get a confidence level 0.68.

10.7.4 Python

For the Python program in Table 10.12, three user-defined functions are needed:

- a bell curve (`def gauss`) with parameters amplitude A , center x_m , and standard deviation x_{sd} ,
- a function (`def gauss2`) adding two bell curves with independent parameters,
- a function (`def gauss3`) adding two bell curves with the same standard deviation.

In Table 10.13, the function `gauss` is used to generate two bell curves $y_A(x)$ and $y_B(x)$ intended to mimic ideal spectral lines. A noisy spectrum is simulated by adding these two curves and a constant background level y_C to get y_{ABC} , which is then blurred with normally-distributed noise to become $y_{ABC}ns$. To avoid negative values that do not occur in reality, a *maximum* function is applied in line 13.

Table 10.12 Defining a Gaussian and the sums of two Gaussians

```

1 import scipy.stats as sct
2
3 def gauss(x, A, xm, xd):
4     f=sct.norm(0,1).pdf(0)
5     return A/f*xd*sct.norm(xm, xd).pdf(x)
6
7 def gauss2(x, A, xA, xAd, B, xB, xBd, yC):
8     G2=gauss(x, A, xA, xAd)+gauss(x, B, xB, xBd)+yC
9     return G2
10
11 def gauss3(x, A, xA, xAd, B, xB, yC): #Same width
12     G3=gauss(x, A, xA, xAd)+gauss(x, B, xB, xAd)+yC
13     return G3

```

Table 10.13 Generation of a noisy EDX signal with two spectral lines

```

1 import numpy.random as npr
2
3 A, xA, xAd =600,1.75,0.05 #Amplitude, position, width
4 B, xB, xBd=150,1.86,0.05
5 yC=20 #Background level
6 Ns=20 #Noise level
7 dx=0.01
8
9 x=np.arange(1.5,2+dx,step=dx)
10 yA=gauss(x,A,xA,xAd)
11 yB=gauss(x,B,xB,xBd)
12 yABC=yA+yB+yC
13 yABCns=np.maximum(yABC+Ns*npr.randn(len(x)),0)

```

In Table 10.14, a function of type `gauss2` is fitted to the noisy data by a nonlinear least-squares fit with `curve_fit`, returning the optimized coefficients (here, stored in p_{opt}) and the covariance matrix (here, stored in cov). The standard errors D of the coefficients are obtained as square roots of the diagonal elements of the covariance matrix. The current fit looks like Fig. 10.24a with two bell curves with different widths, $x_{\text{Asd}} = 0.047$ and $x_{\text{Bsd}} = 0.67$, as reported in the bottom cell of Table 10.14.

Table 10.14 Fitting `gauss2` (independent widths of the two bell curves) to noisy data; “error” means standard error

```

1 from scipy.optimize import curve_fit
2
3 p0=[400,1.7,0.05,200,1.9,0.05,130] #Initial guess
4 popt, cov = curve_fit(gauss2, x, yABCns,p0)
5 D=np.zeros(7)
6 for i in range(7):
7     D[i]=np.sqrt(cov[i][i]) #√Trace of covariance matrix

```

	A	xA	xAd	B	xB	xBd	yC	
true	600	1.750	0.050	true	150	1.860	0.050	20
estim.	534	1.745	0.047	estim.	180	1.828	0.067	21
error	189	0.006	0.004	error	100	0.062	0.027	6

Table 10.15 Fitting `gauss3` (equal widths of the two bell curves) to noisy data; “error” means standard error

```

8 coef=[A, xA, xAd, B, xB, yC]
9 inC=np.ndarray(6, dtype=bool)
10 yABCns=np.maximum(yABC+Ns*npr.randn(len(x)), 0)
11 ini=[400, 1.7, 0.05, 200, 1.85, 200] #Initial guess
12 popt3, cov = curve_fit(gauss3, x, yABCns, p0=ini)
13 D=np.zeros(6)
14 for i in range(6):
15     D[i]=np.sqrt(cov[i][i])
16     inC[i]=popt3[i]-D[i]<coef[i]<popt3[i]+D[i]

```

	A	xA	xAd		B	xB	xBd	yC
true	600	1.750	0.050	true	150	1.860	0.050	20
estim.	597	1.752	0.054	estim.	144	1.869	0.054	12
error	10	0.0015	0.0015	error	14	0.007	0.001	5
in	True	False	False	in	True	False	False	False

Question

How can lines 5-7 in Table 10.14 be formulated with list comprehension in one line?²⁷

A fit with `gauss3`, Table 10.15, supposing equal widths of the bell curves, results in coefficients close to the “true” ones. In order to determine the confidence levels of the calculated standard errors in the coefficients, we check whether the error ranges capture the true values. The variables “in are reported in the bottom cell. To do that efficiently in a loop, the coefficients are put together in a list `coef`, to be addressed with the loop index. The list is of the same size as the estimated coefficients `popt3` and their errors `D`, and the result of the logical check is stored in a list `inC` of the same size.

Hit rates

Based on the logical check in Table 10.15, we can estimate the hit rates of the error ranges (see Table 10.16). The statements for a fit with `gauss3` are assembled into a function `hitRates` that returns the Boolean array `inC`, stating, for every coefficient, whether the true value is captured by the standard error range. A loop over `rep = 1000` repetitions counts the number of Trues in `NinCC`, with the result being reported in the bottom cell of Table 10.16. The hit rates are close to 668, so that we have no reason to doubt that `curve_fit` returns the standard errors of the estimated coefficients.

²⁷ `D = [np.sqrt(cov[i][i]) for i in range(len(cov))].`

Table 10.16 Determining the hit rates for the standard errors obtained in Table 10.15; “function name” should be “variable namefit with gauss3, according”

```

17 ini=[400,1.7,0.05,200,1.85,200]
18
19 def hitRates():
20     inC=np.ndarray(6,dtype=bool)
21     yABCns=np.maximum(yABC+Ns*npr.randn(len(x)),0)
22     popt3, cov = curve_fit(gauss3, x, yABCns,p0=ini)
23     D=np.zeros(6)
24     for i in range(6):
25         D[i]=np.sqrt(cov[i][i])
26         inC[i]=popt3[i]-D[i]<coef[i]<popt3[i]+D[i]
27     return inC
28
29 coef=[A, xA, xAd, B, xB, yC]
30 NinCC=np.zeros(6)
31 rep = 1000
32 for r in range(rep):
33     inCC=hitRates()
34     #Without () only another variable name
35     NinCC+=inCC

```

```

inC:
[ 664.00  679.00  683.00  669.00  680.00  691.00]

```

Questions

What is the difference between the statements `inCC = hitRates` and `inCC = hitRates()`?²⁸

What is the degree of freedom for a fit with `gauss3`, according to Sect. 10.7.2?²⁹

10.7.5 Spreadsheet

Generation of a noisy spectrum

The ideal spectrum y_{ABC} without noise is generated in Fig. 10.26 (S) with the parameters specified in B1:E5. A standard normal distribution is used for the noise, addressed as `NORM.DIST(x; xA, xAsd;0)` where “0” stands for pdf. Its amplitude is calculated

²⁸ The statement `inCC = hitRates()` calls the function `hitRates()`, whereas `inCC = hitRates` simply assigns another name to this variable.

²⁹ From lines 11 of Table 10.12 and 9 of Table 10.13: $dof = 51 \text{ (points)} - 6 \text{ (parameters)} = 45$.

	A	B	C	D	E	F	G	H	I	J	K
1	Amplitude	A	600	B	150				varTot	46158	
2	Center	xA	1.75	xB	1.85				varRes		
3	StD	xAsd	0.05	xBsd	0.05				R ²		
4	Offset	yC	20								
5	Noise level	Ns	20	nD	0.399	=NORM.DIST(0;0;1;0)					
6		dx	0.01						Ns=20; NsR=22		
7											
8											
9		x	yA	yB	yABC	Noise	yABCns	NsR			
10		1.5	0.00	0.00	20.00	-7	13	22.01			
59		1.51	0.01	0.00	20.01	-22	0				
		2	0.00	1.67	21.67	-4	17				

Fig. 10.26 (S) Generation of a noisy spectrum y_{ABCns} ; step (1) of Sect. 10.7.2

in E5 as n_D , guaranteeing that the integral over the function is 1. As we want to handle the maxima A and B of the curves as parameters independent of the widths, the pre-factors for y_A and y_B are calculated corresponding to Eq. 10.25. With A and the width x_{Asd} (or B and B_{sd}), we have two parameters at hand that have an immediate visual meaning and can independently be adjusted by hand for the bell curves to fit the experimental curves.

The ideal “true” spectrum y_{ABC} is blurred with noise to become y_{ABCns} in column G. The signal, a count rate, is never negative in reality; this is guaranteed with $\text{MAX}(*, 0)$. The residual noise N_{sR} is calculated as the standard deviation of $(y_{ABCns} - y_{ABC})$. Its value is close to the pre-specified “true” noise level N_s .

Question

To generate the noisy spectrum in Fig. 10.26 (S), two functions are used: NORM.DIST (reported in C7) and NORM.INV (reported in F7). Which roles do they play?³⁰

Fitting bell curves to the spectrum

In Fig. 10.27 (S), the sum of two bell curves, G_a plus G_b plus an offset y_{Cr} , is fitted to the noisy spectrum created in Fig. 10.26 (S). This is done in a separate spreadsheet into which the vectors x and y_{ABCns} are copied from Fig. 10.26 (S). The variables for SOLVER are the 7 coefficients in C1:F4 with index r . They are changed in each iteration of SOLVER so that y_{ABCns} is calculated anew. To have the “experimental” spectrum fixed, the contents of y_{ABCns} have been value-copied (COPY, PASTEVALUES) into y_{Stop} . The variable y_{ABCns} is no longer needed in the current fit. The target cell for SOLVER is N_{sR} in G9, the residual noise level for $(y_{Stop} - G_{abc})$. The coefficient of determination r_{sq} of the fit is calculated in H1:H3 according to Eq. 10.6.

³⁰ NORM.DIST produces a smooth Gaussian curve as a function of energy. NORM.(RAND();0;NS) produces the normally distributed noise of the y-values.

	A	B	C	D	E	F	G	H	I	J
1		Amplitude	Ar	405.20	Br	268.20	varTot	45495	=VAR.S(yABCns)	
2		Center	xAr	1.75	xBr	1.79	varRes	316	=SUMXMY2(yABCns;Gabc)/50	
3		Std	xAsdr	0.046	xBsdr	0.080	rSq	0.99	=(varTot-varRes)/varTot	
4		Offset	yCr	10.14						
5		Noise level	NsR	16.3	=G9	Gabc, R ² =0.99				
6										
7	=x	=yABCns	427	=Ar/nD*xAsdr*NORM.DIST(x;xAr;xAsdr;0)		=Br/nD*xBsdr*NORM.DIST(x;xBr;xBsdr;0)		=Ga+Gb+yCr		
8	x	yABCns	yStop	Ga	Gb	Gabc	NsR	=SQRT(SUMXMY2(yStop;Gabc)/50)		
9	1.5	21	19	0.0	0.4	10.5	16.3			
59	2	26	0	0.0	8.7	18.8				

Fig. 10.27 (S) New spreadsheet calculation: fit of the sum of two bell curves to the noisy data of Fig. 10.26 (S); y_{Stop} is the “frozen” result of the fit procedure, reported in Fig. 10.24a

Questions

What is the adjusted coefficient of determination r_{SqAdj} for Fig. 10.27 (S)?³¹
 Is the procedure reported in Fig. 10.27 (S) a least-squares fit?³²

The results for two different fits with the sum of two bell curves, (a) with individual widths and (b) with identical widths, are shown in Fig. 10.28 (S). Both fits yield a very high coefficient of determination, $R^2 = 0.99$, so there is no good reason to prefer one particular fit. However, we know that, for physical reasons, the width of the spectral lines is determined by the resolution of the detector, and therefore should be the same for both lines; thus, (b) is preferred.

10.7.6 C-spec Error of the Optimized Coefficients by Simulation-Based t Adaptation

Fig. 10.29 (S), based on Fig. 10.28 (S), presents a spreadsheet calculation for obtaining the errors of the optimized coefficients.

Row 4 lists the optimized coefficients of a fit of the sum of two bell curves to noisy experimental data. They are now regarded as the true values for a *new statistical experiment* and used as parameters to generate a spectrum in the calculation model of Fig. 10.26 (S) that is then entered into Fig. 10.27 (S). The noise level NsR , calculated as the standard deviation of the difference between the noisy spectrum and the fitted curve in H9 of Fig. 10.26 (S), is value-copied into G9 of Fig. 10.27 (S).

³¹ $R^2_{adj} = 1 - (1 - R^2) \cdot \frac{N-1}{N-ddof}$ (Eq. 10.8), here, $N = 51$ (Fig. 10.26 (S)), $ddof = 7$ and $R^2 = 0.99$ (Fig. 10.27 (S)), so that $R^2_{adj} = 0.989$. In this example, the difference between R^2 and R^2_{adj} is unimportant.

³² Strictly speaking, not because the target is the square root of the squares of the deviations. However, *sqr*t is a strictly monotonously increasing function so that the minimum in *sqr*t is also the minimum in the squares.

	A	B	C	D	E	F	G	H	I	J
1	Amplitude	Ar	405.20	Br	268.20	varTot	45495	$=VAR.S(yABCns)$		
2	Center	xAr	1.75	xBr	1.79	varRes	316	$=SUMXMY2(yABCns;Gabc)/50$		
3	Std	xAsdr	0.046	xBsdr	0.080	rSq	0.99	$=(varTot-varRes)/varTot$		
4	Offset	yCr	10.14							
5	Noise level	NsR	16.3	$=G9$	Gabc,	$R^2=0.99$				

	A	B	C	D	E	F	G	H	I	J
1	Amplitude	Ar	621.70	Br	122.50	varTot	44929	$=VAR.S(yABCns)$		
2	Center	xAr	1.75	xBr	1.86	varRes	284	$=SUMXMY2(yABCns;Gabc)/50$		
3	Std	xAsdr	0.052	xBsdr	0.052	rSq	0.99	$=(varTot-varRes)/varTot$		
4	Offset	yCr	15.51							
5	Noise level	NsR	16.1	$=G9$	Gabc,	$R^2=0.99$				

Fig. 10.28 (S) Copy of instances from Fig. 10.27 (S); results of fitting with bell curves with **a** (top) individual widths, **b** (bottom) identical widths; F3 = [=xAsdr]

	K	L	M	N	O	P	Q	R	S
3		Ar	xAr	xAsdr	Br	xBr			
4		405	1.75	0.046	268	1.79			
5		DyAr	DxAr	DAsdr	DyBr	DxBr			
6		10	0.0021	0.0015	16	0.0071			
7		$=AND(Ar-DyAr<A;A<Ar+DyAr)$ $=AND(xAr-DxAr<xA;xA<xAr+DxAr)$ $=AND(xAsdr-DAsdr<xAsd;xAsd<xAsdr+DAsdr)$ $=AND(Br-DyBr<B;B<Br+DyBr)$ $=AND(xBr-DxBr<xB;xB<xBr+DxBr)$							
8		inA	inxA	inxAd	inB	inxB			
9		FALSE	FALSE	FALSE	FALSE	FALSE			
10	100	67	79	71	66	67			
11		0.67	0.79	0.71	0.66	0.67	$=P10/$K10		
12	pOut	0.33	0.21	0.29	0.34	0.33	$=1-P11$		
13	t	0.99	1.27	1.07	0.96	0.99	$=T.INV.2T(pOut;44)$		
14	stE	11	0.0016	0.0014	16	0.007	$=DxBr/t$		

Fig. 10.29 (S) Continuation of Fig. 10.28 (S); spreadsheet calculation for determining the standard errors of the optimized coefficients; L4:P4 copied from D1:F3

The deviation values reported in row 6 are obtained by changing the optimized parameters until the trend curve differs visibly from the noisy spectrum to be pre-supposed as errors for the coefficients obtained in *new statistical experiments* with spreadsheet calculations as in Fig. 10.26 (s) and Fig. 10.27 (S) in order to determine whether the supposed confidence intervals capture the true values.

With a rep-log procedure, the *new statistical experiment* is repeated 100 times to obtain the hit rates reported in row 10 of Fig. 10.29 (S). From that, we get the confidence levels in row 11 and the error probability p_{Out} into row 12; therefrom, the corresponding t -value in row 13; and finally, the standard errors stE for a confidence level of 0.68 estimated as initial errors (row 6) divided by t .

This simulation with 100 trials is repeated with the stE copied manually as standard errors in row 6 in the following run. The results of this iterative procedure are reported in Fig. 10.25 (S).

10.8 Questions and Tasks

1. How do you obtain a linear mapping of the function

$$y = A \cdot \exp\left(\left(\frac{x}{\Delta x}\right)^2\right)$$

and what meaning do the y-axis intercept and slope at $x = 0$ have?

- In Fig. 10.30a (S), a straight line is adapted to measurement points with LINEST. What is the meaning of the numbers in D1:E2 and D3?
- A parabola is fitted to 7 data points with LINEST. How many degrees of freedom does the fitted parabola have? Does our rule: Ψ *Two within and one out of*? apply to the coefficients of the parabola?
- In Fig. 10.30b, a polynomial $y_C(x)$ is fitted to data points (x, y_S) with SOLVER. Which is the target cell, and which are the adjustable cells? Which mathematical formula is behind the spreadsheet formula in S5?
- In Fig. 10.31 (S), a parabola is fitted with LINEST to 7 data points (x, y) . A rep-log procedure counts 10,000 times how often the true values of a, b , and c are outside of the error range. Which data are in B7:C13? What is the meaning

	D	E	F	G		N	O	P	Q	R	S	T	U
1	2.97	1.35	=LINEST(y;x;;1)										
2	0.19	1.20											
3	0.97	1.76			5								
					6								
					7	x				yC		yS	
					8	0	aS	1.11	1.74	2.49	4.06		
					9	1	bS	1.66	6.34	5.25			
					10	2	cS	2.49	10.84	10.24			
					13	6			52.38	52.41			

=aS*x^2+bS*x+cS
=SUMXMY2(yC;yS)

Fig. 10.30 a (left, S) Output of LINEST. b (right, S) A polynomial y_S is fitted to the data points (x, y_C) with SOLVER

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	a	1			1.11	1.66	2.49						
2	b	2			0.11	0.69	0.88	={=LINEST(A7:A13;B7:C13;;1)}					
3	c	3			1.00	1.01	#N/A						
4	Ns	1						10000					
5													
6													
7	y	x	x ²		a.1	b.1	c.1	a.2	b.2	c.2			
8	1.74	0	0		FALSE	TRUE	TRUE	TRUE	TRUE	TRUE			
9	6.34	1	1		0.378	0.379	0.383	0.374	0.119	0.120	0.119	0.116	
13	10.84	2	4										
	52.38	6	36										

=AND(E1-E2<a;a<E1+E2)

=T.DIST.2T(1;4)

=AND(F1-2*F2<b;b<F1+2*F2)

=T.DIST.2T(2;4)

Fig. 10.31 (S) A parabola is fitted to the 7 data points (x, y) with LINEST. The values in E8:G8 and I8:K8 are the result of a rep-log procedure

Table 10.17 Python code snippet for performing a linear least-square fit to data points (x, y_{Ns}) in arrays x and y_{Ns}

```

1  #Data points (x,y) in lists x and yNs
2  model=sm.OLS(yNs,xx)
3  results=model.fit()
4  aR=results.params[0]
5  aE=results.bse[0]
6  r2=results.rsquared
7  r2_adj=results.rsquared_adj
    
```

Fig. 10.32 (S) For determining the intersection of two straight lines

	A	B	C	D	E
1	a.1	1		a.2	3
2	m.1	2		m.2	-2
3					
4	x	y.1	y.2		
5	1	3	1	4	

of the values in E2:G2 and the arguments of the formula for T.DIST.2T in H8 and L8? How have the values in E8:G8 and in I8:K8 been obtained?

- Table 10.17 shows a Python code snippet for performing a linear least-square fit to data points (x, y_{Ns}) in arrays x and y_{Ns} . A linear trend line is fitted to the data points with the OLS function of the statsmodel.api library. What is the array xx in the argument of OLS? How do you get the coefficients of the linear trend line and their standard errors? What is the difference between the two quantities queried in lines 6 and 7?
- Figure 10.32 (S) shows the spreadsheet layout for determining the intersection of two straight lines with SOLVER. What are the formulas in B5, C5, and the target cell D5? What are the objective (target) cell and the variable cells in SOLVER?
- Figure 10.33a shows a spreadsheet layout for calculating the intersections of three straight lines displayed in **b**. What are the formulas for y_1 , y_2 , and y_3 ? The target cell to be minimized is E4. What are the formulas in E6, E7, and E8?
- Table 10.18 shows a Python program for solving the problem of Fig. 10.33, namely finding the intersections of three straight lines with a solver algorithm. It is incomplete. Introduce the 6 missing statements!

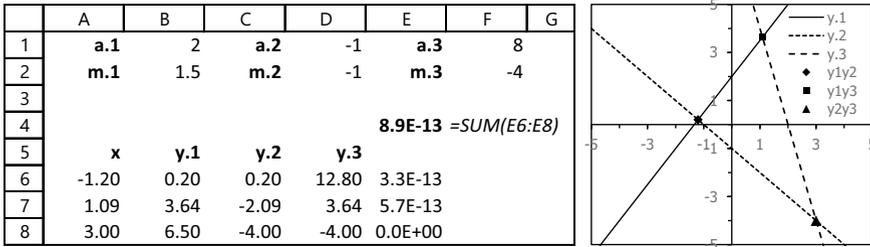


Fig. 10.33 **a** (S, left) spreadsheet layout for calculating the intersections of y_1 , y_2 , and y_3 using SOLVER, the column vector x contains the optimized x -values of the intersections. **b** (right) Straight lines of **a**, together with their intersections

Table 10.18 Python program for solving the problem in Fig. 10.33; 6 lines are omitted

```

8  a1, m1 = 2, 1.5
9  y1 = lambda x: a1+m1*x
10 def target(x):
11     y12=(y1(x[0])-y2(x[0]))**2
12     return y12+y13+y23
13 xI=[-1,1,2.5]
14 x=minimize(target,xI,method='SLSQP')
15 print(x.x)
    
```

Index

Symbols

- Γ structure, 9, 69, 261
- Ψ Always one more! Yes, but of what and than what?, 14
- Ψ Approximated average value instead of exact integral, 13
- Ψ Calculate with variances, report the C-spec error, 14, 368, 375, 391, 398
- Ψ Chance is blind and checkered, 14, 318, 361
- Ψ Come to a decision! You may be wrong, 14
- Ψ Cos plus Cos equals mean value times half the difference, 13
- Ψ Ctrl + Shift + Enter, 12, 30
- Ψ Doppler effect with plus and minus, 13, 92, 110
- Ψ Empty lines separate curves, 12, 174
- Ψ First, the tangent at $x = 0$, 13
- Ψ From variance to confidence with Student's t value, 14
- Ψ half, half, whole, the halves count twice, 7, 13, 281, 305
- Ψ If in doubt, count!, 7, 14, 307, 365
- Ψ Imaging equation for lenses with plus and minus!, 13
- Ψ Mostly, not always, 6, 14, 307, 417
- Ψ Plus 1 becomes times e , 13
- Ψ The dollar makes it absolute, 7, 11, 69
- Ψ Twice as good with four times the effort, 14
- Ψ Two within, one out of, 14, 367, 369, 376, 377, 380–382, 391, 417, 421, 424, 426, 433, 442, 472
- Ψ We know everything and play stupid, 14, 365, 385, 392, 421, 423, 444, 461, 464
- Ψ Worse makes good even better, 14, 390, 417

A

- Ab-initio constructors, 12, 37
- Absolute cell addressing, 16, 19
- Absolute temperature, 103, 420, 456
- ACTIVESHEET.SHAPES.ADDSHAPE, 126
- ActiveX control, 24
- ACTIVEX CONTROLS, 24, 25, 131
- Add-In, 168, 170, 450, 451
- Addition of cosines, 57, 65, 68
- ADDSHAPE, 128, 134
- Adjusted R^2 , 423, 433, 434
- AND, 322
- Angle of inclination, degrees, 209
- Angular frequency, 60, 242, 247
- Animation, 229, 234–239, 246, 251, 254, 257, 272, 350, 351
- Animation object, 237
- Antiderivative, 347, 352
- Append, 34, 176, 178
- Application as a matrix formula, 52
- Arcus tangens, 39, 42, 43, 199, 202, 223
- Area under a curve, 192, 195
- argmax, 56
- argmin, 56
- ARRAY in VBA, 157
- Arrays in Numpy, 15, 16, 31
- Arrays in Python, 70, 325
- Arrhenius plot, 420
- Arrow, 9, 25, 35, 88–90, 126, 132, 148, 185, 186, 198–203, 205, 209–215, 217, 229–234, 236, 237, 249–251, 266, 335
- Arrow heads, 185, 212, 231, 237
- as npr, 35, 37, 309, 324
- ATAN2, 39, 43, 199, 223
- AVERAGE, 320, 387
- Axis, 9, 10, 18, 24, 39, 40, 54–57, 65–70, 79–81, 83, 86, 88, 89, 93, 95–97,

- 99, 102–109, 112, 113, 137, 142, 175–178, 181, 183, 184, 188, 189, 192, 194–196, 198, 200–203, 206, 224, 225, 228, 237–249, 254, 258–261, 266, 297, 298, 335, 345, 406, 419, 426, 450, 449, 450, 453, 454, 457, 459, 472
 - Axis=1, 54, 67
 - ax1.twinx, 108
- B**
- Barometric formula, 283–285
 - Bayes' rule, 329, 331, 332
 - Beats, 13, 17, 57, 59, 60, 63, 65, 68
 - Bell curve, 60, 334, 335, 343, 345, 346, 460–467, 469–471
 - Bernoulli, 227, 228, 241
 - Boltzmann, 103, 286, 420, 456
 - Brachistochrone problem, 241
 - Broadcasting, 2, 12, 16, 17, 41, 42, 44–47, 66
- C**
- Cartesian coordinates, 13, 71, 81, 137, 198–202, 224, 227, 243, 252, 258, 264–266
 - Case-insensitive, 139, 148
 - Case-sensitive, 139, 331
 - Cdf, 309, 310, 333–335, 338–342, 345, 352, 360, 384
 - Cell addressing, 11, 16, 19, 22
 - Cell references, 21, 26–28, 69, 122, 232
 - Cell references, relative, absolute, indirect, 26
 - Center of gravity, 136, 212–214, 228, 251–259, 267
 - Central ray, 85–87
 - Centrifugal, 112, 207–209, 211
 - Chain rule, 189
 - CHART STYLES, 23
 - Checkerboard, 99, 132, 133, 135, 136, 139, 141, 142
 - Chemical react, 420
 - Chi², 337
 - Chi² distribution, 358
 - Chi² test, 14, 307–314, 316, 317, 333, 337–340, 343, 345, 347, 348, 358–361
 - CHISQ.DISTR.T, 310, 312
 - CHISQ.TEST, 307, 312, 313, 317, 338
 - Chisquare, 35, 308
 - Circle, 71, 72, 86, 123, 125, 128–133, 139, 140, 146, 177, 178, 194–197, 246, 252, 260–266, 457, 458
 - Close-packed plane, 143–145, 149, 150
 - Coefficient of determination, 422, 426, 428, 432, 437, 441, 442, 448, 469, 470
 - Color, 55, 90, 125, 126, 128, 129, 131–135, 137, 150
 - Column vector, 12, 16, 27, 41–44, 46, 47, 51–53, 56, 63, 65, 66, 106, 171–174, 184, 211, 214, 219, 221, 263, 264, 303, 322, 355, 434, 437, 474
 - COMMANDBUTTON, 128, 131, 132
 - Command button, 13, 128, 131, 132
 - Commutative, 46
 - Composite function, 188, 189, 191
 - Concatenation operator, 24, 53, 69
 - Confidence interval, 309, 365–367, 370, 376, 378, 380, 382, 388, 391, 397, 399, 407, 408, 410, 411, 414–416, 419, 427, 431
 - Confidence level, 7, 309, 365–368, 370, 378, 382, 383, 385, 388, 391, 399, 408, 409, 411, 414, 415, 421, 423, 424, 431, 433, 436, 437, 439, 440, 442, 463–465, 467, 471
 - Confidence-specified error, 368, 407, 409
 - Coordinate system, 76, 81, 83, 136, 142, 179, 198, 199, 242, 249, 251, 254, 263
 - Cosine, 9, 13, 15, 17, 57, 59–70, 73, 78, 121, 168, 187, 188, 193, 195, 223, 347, 449
 - COUNT(), 323, 325
 - COUNTIF, 322, 323
 - Covariance matrix, 461, 463, 466
 - Create a name, 77
 - Creation of diagrams, 16
 - C-spec error, 7, 14, 367, 368, 375, 383, 385–387, 389–392, 394, 395, 397, 398, 407–412, 414–417, 422–424, 444, 460, 463–465, 470
 - Cumulative density function, 309, 310, 352
 - Curie–Weiss law, 419
 - Curly brackets, 30, 44, 184, 200
 - curve_fit, 14, 35, 419, 421, 424, 460, 461, 463, 466, 467
 - Cycloid, 227, 228, 238, 241, 243, 246
- D**
- Damping, 270, 273
 - Data frame, 117, 165, 167, 169
 - DataFrame, 117, 165, 169
 - Data series into a chart, 22
 - Data types, 34, 36, 134
 - Ddof, 310, 315–317, 321, 380, 412, 423

Debug, 120, 127, 155
 DEBUG/ STEP INTO, 120, 127
 Decay constant, 445
 def ArrowP, 90, 185, 234
 Definite integral, 106, 192
 def StdFig, 108
 Degrees of freedom, 310, 311, 313, 315, 316,
 358, 359, 361, 380, 383, 391, 412,
 421, 423, 426, 429, 433–435, 472
 Delete, 34, 120, 126, 314
 Delta degrees of freedom, 310, 315, 380, 423
 DESIGN MODE, 25, 131
 Determinant, 42, 45, 47, 48, 216, 223
 DEVELOPER, 24, 25, 118, 124, 125, 127,
 131, 453
 Dictionaries { }, 34
 Difference equations, 270
 Diffraction pattern, 350, 351, 356
 DIM, 148, 153, 157, 171, 172
 direct cell addressing, 16
 Directrix, 17–19, 31
 Distance to track, 99
 Distribution function, 308–310, 312, 313,
 333–335, 338–343, 345, 346, 348,
 350, 352, 355, 362, 363
 DO ... LOOP UNTIL, 165
 Dollar sign, 106
 Dot product, 224, 253, 257

E

EDX, 460, 462, 466
 Electrical network, 182, 215, 217, 219, 221
 Ellipse, 124, 127–129, 132, 134, 204
 EMPTY, 164, 165
 Equi-distribution, 310, 313–316, 324
 Equilibrium of torques, 213
 Error propagation in powers, 400, 403
 Error propagation in products, 399, 403
 Error propagation in sums, 368, 398, 402
 Error range, 7, 14, 365–370, 374–380, 382,
 384–388, 393, 395, 411, 421,
 424–428, 435, 436, 439, 442, 443,
 446, 447, 464, 467, 472
 Euler, 13, 101, 269, 271, 272, 274, 276–278,
 304
 Explained variance, 442
 Exponential, 13, 75, 78, 99, 101–107, 113,
 308, 347, 372, 420, 421, 423,
 440–448
 Exponential distribution, 347
 Exponential growth, 75, 100
 Externally consistent, 390, 393, 395, 396

F

fcc, 143–145, 147, 148, 150, 151
 Fill handle, 21
 fillstyle, 108, 214, 258, 259, 326, 331, 455
 FinRes, 373, 374, 415, 429
 First derivative, 186, 187, 190, 224
 Focal ray, 86
 Focus, 17, 18, 31, 80, 81, 83, 85
 For-loop, 91, 118, 120, 122, 123, 130, 131,
 139, 143, 146, 163, 164, 167, 206,
 233, 286, 315–317, 360, 389, 396,
 404, 415, 449
 FORMAT/ CURRENT SELECTION, 23
 FORMAT DATA SERIES, 212, 231
 FORMAT SELECTION, 23, 107
 Formatted output with %, 41
 Formatting the chart, 23
 Formula network, 76, 77, 244, 263, 372
 Foucault's pendulum, 228
 Frames, 136, 142, 167, 234, 236–238, 251,
 257
 Frequencies of occurrence, 308, 310, 311,
 315–317, 336–339, 341, 345, 350,
 358–360, 362, 384
 FREQUENCY, 14, 57–64, 66, 67, 73, 92–99,
 307, 308, 310–317, 323, 324, 326,
 333, 336–339, 341, 343–345, 351,
 353, 356–358, 360–362, 384
 Friction, 112, 227, 269–271, 273, 274,
 278–280, 282–285, 287–289, 291,
 295, 298–303, 305
 Frictional force, 208, 280, 287, 299, 306
 Friction force of the rope, 299, 300, 302
 fsolve, 35, 448, 456
 Fundamental frequency, 57–59, 63
 Fundamental rule of statistical reasoning, 14,
 307, 369, 392
 Fundamental theorem of calculus, 194, 197

G

Global parameters, 98, 139, 148–150, 276,
 292, 297, 404
 Global variables, 132–135, 244
 GOAL SEEK, 421
 Gosset, 167
 Gravitational force, 112, 207
 Gravity, 227, 229, 230, 253–255, 257,
 278–280, 298, 303, 382, 390
 Grid scale in the spreadsheet, 129

H

Half-step procedure, 271, 272, 305

- Harmonics, 57–59, 62, 65, 67
 Height-dependent friction, 284
 Hit rate, 7, 368, 374, 377–380, 383, 385,
 391–394, 396, 408, 409, 411, 414,
 415, 421, 424, 425, 428, 429, 433,
 437, 439, 440, 442, 444, 447, 463,
 464, 467, 468, 471
- I**
- Identifier, 21, 30, 33, 34, 37, 38, 46, 55, 64,
 76, 162, 170, 184
 If, 10, 13, 14, 115, 117, 123, 167, 224, 303
 If ... then ... else, 349
 IF(), 10, 294, 303, 322, 323
 Imaging equation for lenses, 13, 82, 83
 import numpy as np, 35, 36
 Indentation, 139, 167, 283
 Independent variable, 9, 10, 20, 27, 53, 63, 76,
 77, 97, 102, 106, 117, 188, 196,
 261, 262, 285, 291, 349, 352, 399,
 400, 420, 424, 426, 429, 435, 436,
 445, 448
 INDIRECT, 11, 16, 26, 27, 70, 249, 386, 387
 Indirect cell addressing, 11, 16
 Integer, 25, 26, 34, 101, 131, 132, 139, 148,
 157, 176, 222, 223, 236, 309, 339,
 372
 Integral function, 192–195
 Integration along a line, 182
 Internally consistent, 391, 393, 395, 396
 Interval search with five queries, 353
 Inverse matrix, 45, 182, 219, 222
- J**
- Join, 151–153, 155, 158, 176
 Jupyter, 3, 11, 16, 31, 32, 64
- K**
- Karl Popper, 339
 Keyword argument, 55, 90, 94, 98, 138, 139,
 166, 167, 185, 233
 Kirchhoff's rules, 182, 215, 217, 219, 220
- L**
- Laboratory system, 228, 238, 243, 246, 247,
 249–251, 266
 Langevin, 456–459
 Length l of a vector, 43
 Length of a curve, 181, 194
 linalg, 47
- Linear algebra, 17, 35, 41–43, 45, 47, 49, 181,
 223
 Linear regression, 8, 14, 419, 420, 425, 428,
 430, 443
 Linear trend line, 405, 421–427, 441, 443,
 447, 473
 Line breaks, 250
 LINDEX, 419, 421, 424, 426–429, 431, 434,
 436, 437, 440, 442, 447, 472
 Line vector, 181–184, 209, 211
 Linked cell, 25, 26, 71, 79, 110, 184, 232, 255
 List comprehension, 12, 16, 109, 224, 324,
 388, 467
 List processing, 15, 16, 181
 Logarithm, 13, 105, 223, 372, 440, 441
 Logarithmic scale, 105, 107
 Logarithmized data, 405, 406, 441–443,
 445–447
 LOGEST, 440, 447, 448
 Logical branches, 117, 125
 Logical IF queries, 133
 Longitudinal wave, 260
 LookAhead, 273–279, 285, 304
 Loop2i, 13, 118, 121, 123, 163, 176, 207
- M**
- Mach cone, 258, 259, 264, 265
 Macro recorder, 115, 117, 124, 125, 174–176
 Magic chord, 30, 44, 52, 172, 184, 311, 312,
 428
 Mass-spring system, 187, 270, 272, 275, 278,
 298, 306
 Mathematical functions, 8, 36, 78, 93, 182,
 222–224, 346
 Mathematical pendulum, 306
 Matlab, 35
 Matplotlib, 35, 39, 55, 65, 90, 175, 176, 185,
 229, 234, 235
 matplotlib.pyplot, 175
 Matrices, 12, 13, 16, 27, 38, 39, 41–49, 57,
 184, 219, 223, 225, 261, 296
 Matrix equation, 217–220
 Matrix formula, 2, 17, 29, 30, 50, 52, 184,
 200, 201, 263, 448
 Matrix function, 12, 30, 172, 219, 222, 307,
 309, 311, 312, 316, 428, 440
 Matrix of powers, 216, 218, 221
 Matrix operations, 2, 15, 17, 41, 45, 182, 225
 MDETERM, 45, 223
 Mean, 13, 14, 27, 38, 57, 61, 62, 73, 79, 85,
 100, 101, 104, 119, 131, 137, 174,
 239, 240, 243, 260, 269–271, 276,
 281, 293, 303, 305, 307, 310, 312,

- 313, 316, 318, 320–322, 324, 333, 334, 339, 340, 342, 343, 356, 358–360, 362, 366–369, 371, 372, 374–378, 380–382, 384–387, 390–395, 397–402, 404, 407, 408, 410–413, 417, 422, 423, 426, 433, 444, 445, 460, 461
 - Microphone signal, 57–59
 - Mid-perpendicular, 181–183, 349
 - Midpoint, 137, 182
 - Minimize, 35, 421, 425, 448, 456, 459
 - MINVERSE, 45, 219, 223, 225
 - MMULT, 29, 45, 219, 223, 225, 254
 - Mobile, 182, 212–214
 - MOD, 222, 292, 293
 - Module in VBA, 164
 - Modulo, 222, 293
 - Moment of inertia, 227, 228, 251, 253–256, 258
 - Motion on a straight line, 93
 - Multilinear regression, 420, 431–433
 - Multi-line strings, 158
 - Multiple statements on one line in VBA, 122
 - Multiple tests for error probability, 309
 - Multiple tests for hit rates, 366, 368, 387
- N**
- N/A, 322, 323
 - NameError, 32
 - Name Manager of EXCEL, 27
 - Naming cell ranges, 11, 51
 - Naming cells, 21
 - Ndarrays, 36
 - Nested loop, 122, 132, 141, 176, 178, 282, 344
 - Network of formulas, 75
 - Newton's equation of motion, 7, 13, 269, 270, 272, 275, 305
 - Newton's law, 188, 269
 - Newtonian friction, 271, 279
 - Nodes, 58, 60
 - Noise, 307, 344, 365, 366, 368–370, 384, 391, 398, 400–403, 407, 410–414, 419, 421, 423–427, 431, 432, 435, 436, 441–445, 460, 463–465, 468–470
 - Nonlinear equations, 448
 - Nonlinear regression, 419, 421, 422, 460
 - Normal distribution, 7, 308, 309, 333–336, 340–343, 345, 346, 359, 360, 362, 381, 425–427, 431, 435, 445, 460, 468
 - NORM.DIST, 309, 333, 335, 343, 469
 - NORM.INV, 293, 308, 309, 333, 336, 337, 343, 345, 363, 381, 408, 412, 413, 425, 427, 443, 469
 - np.append, 109, 176, 196, 225, 388, 432
 - np.arange, 12, 16, 37, 70, 107, 108, 295
 - np.arctan2, 39, 137
 - np.array, 12, 36, 47, 53, 65, 70
 - np.dot, 39, 46, 224, 253
 - np.exp, 101, 335
 - np.extract, 325, 332
 - np.flipud, 90
 - np.histogram, 315–317, 341, 362
 - np.hstack, 38, 90
 - np.inf, 317
 - np.linspace, 12, 16, 37, 53, 65, 70, 108, 109, 224, 225
 - np.linalg.det, 43, 47, 49
 - np.log, 37, 308
 - np.logical_and, 324, 325, 389
 - np.logical_not, 325
 - np.logical_or, 325
 - np.matmul, 219, 223
 - np.mean, 320
 - npr.choice, 307–309
 - npr.rand, 35, 37, 141, 308, 315, 317, 343
 - npr.randn, 35, 37, 336, 341, 412
 - npr.random, 324, 332, 336, 341, 345
 - np.shape, 38, 47
 - np.std, 321, 380, 412
 - np.tan, 39, 308
 - np.transpose, 46, 47, 49
 - npl.linalg.inv, 43
 - numpy.linalg, 43, 182, 221, 222, 225
 - Numpy.random, 35, 37, 135, 307–309, 324, 336, 363
- O**
- Object-side, 81, 86
 - ON ERROR GOTO, 155
 - openpyxl, 166
 - Ordinary Least Squares (OLS), 421, 424, 437, 438
 - Orthogonal vector, 209
 - Oscillation of a mass-spring-system, 187
 - out.to_excel, 169
 - OverflowError, 101
 - Overtones, 13, 17, 57–59, 62, 73
- P**
- Pandas, 35, 117, 160, 162, 165, 167
 - Parabola, 15, 17–20, 22, 24, 40, 49–53, 55, 56, 178, 189, 229–231, 420, 431–437,

- 439, 440, 444, 445, 448–451, 455, 472
 - Parabola from vertex, 17
 - Parallelogram of forces, 203
 - Path velocity, 240, 243
 - Pendulum, 9, 227, 228, 246–251, 382, 390
 - Percent point function, 309, 310, 334, 340, 352
 - Perpendicular, 110, 112, 141, 182–184, 204–208, 224, 237, 308
 - Perpendicular bisectors, 198
 - Photons, 308, 348–350, 355, 356, 358, 460
 - Physical unit, 10, 76, 199, 200, 208, 232, 247, 273, 290, 320, 334, 367, 368, 375, 398
 - Plain-text file, 161
 - Plot objects, 234–236, 238
 - plt.arrow, 185, 186, 233
 - plt.axis('scaled'), 203
 - plt.legend, 55
 - plt.rcParams, 237
 - plt.savefig, 98
 - plt.xscale(value"log"), 108
 - plt.yscale, 40, 108
 - Points, 6, 8, 18, 19, 27, 29, 39, 43, 55, 60, 62, 63, 65, 78–81, 85, 86, 88–91, 95, 110, 124, 130, 137, 139, 177–179, 183–185, 187, 199–202, 209–211, 213, 228, 230, 233, 236, 238–249, 251–256, 260, 265–267, 269, 270, 294, 298–301, 318–320, 324–330, 369, 375, 410, 451, 457
 - Polar coordinates, 13, 71, 78, 137, 182, 195, 200–202, 204, 227, 242, 258, 260, 261, 265
 - Polynomial, 19, 178, 182, 192, 193, 195, 196, 204–207, 215, 216, 218–221, 421, 431, 432, 434, 437, 444, 448, 449, 472
 - Polynomial from defining points, 220
 - Power fluctuations, 289
 - Power-law trend line, 405, 406
 - Power operator, 20, 211
 - Powers of 2, 100
 - ppf, 308, 310, 333–336, 340, 341, 343, 345–347, 350–355, 357, 383, 384, 388, 439, 440
 - Printoptions, 45
 - Printoptions for arrays, 45, 46
 - Probability, 7, 308–310, 313, 314, 323, 324, 326–330, 333–336, 338–340, 342–346, 348, 350–352, 358, 359, 361–363, 365, 367, 368, 375, 376, 378, 382–387, 389, 394, 396, 433, 440, 460, 471
 - Probability Density Function (PDF), 310, 333–336, 339, 340, 342, 343, 345, 347, 350–352, 358–360, 362, 384
 - Procedure header, 129, 130, 133–135, 138, 139, 146
 - Product rule of probability, 326, 327
 - Program flow in Jupyter, 64
 - Progress loop, 272, 277, 282, 283, 287, 295, 297
 - Progress with look-ahead, 13, 269, 271, 272, 274, 291, 302, 305
 - PROJECT EXPLORER, 126, 127, 168, 170
 - Projectile motion, 227
 - Projectile trajectory, 227, 229, 230, 233
 - Pythagoras, 96
- Q**
- Quantile function, 334
- R**
- RIC0 in Pandas, 165
 - Radius of curve, 209, 211
 - Ray constructions, 76, 83, 85, 90
 - RBS, 161, 163
 - rcParams, 39, 235, 237
 - RECORD MACRO, 124, 125, 131, 174, 453
 - Rectangle, 25, 76, 77, 118, 123, 125, 126, 131–135, 139, 148, 318–327, 330
 - Referring to names in a cell, 29
 - Referring to names in a matrix function, 30
 - Regression coefficient, 425
 - Relative addressing, 15, 20
 - Rep-log procedure, 13, 117, 121, 177, 240, 243, 293, 314, 344, 347, 348, 356, 379–381, 387, 389, 392, 395, 401, 403, 427, 428, 435–437, 442, 447, 449, 450, 471, 472
 - Residual variance, 422, 423, 432, 433
 - RGB, 128
 - Rolling curve, 228, 238, 242, 243
 - Rope friction, 300
 - Rotation, 183, 227, 228, 238, 239, 241–243, 246–248, 251–256, 266
 - Rotational matrix, 184, 228, 247, 252–256
 - RotMat, 254
 - Round, 222
 - Round a measurement result, 372
 - Row vector, 12, 16, 27, 41–44, 46, 47, 50, 51, 53, 63, 65, 66, 171–174, 221, 253, 261, 263, 264, 437

- Runge-Kutta, 6, 13, 269, 271–281, 283, 295, 305
- S**
- Scalar product, 46, 168, 170–173, 182, 198, 200–202, 208, 209, 211, 224
- Scan-log procedures, 117
- Scipy, 35, 333, 459
- scipy.optimize, 421, 448, 456, 461, 463
- Scipy.stats, 35, 307–309, 341, 358, 421, 424, 437, 438, 460
- scipy.stats as sct, 309, 341
- SCROLLBAR, 15, 19, 24–26, 71, 179, 200
- sct.chi2, 358
- sct.chisquare, 316, 317
- sct.norm, 308
- sct.norm.ppf, 332, 336, 341, 345
- sct.t, 383–385, 388, 396, 415, 416, 429, 433, 438–440, 442
- sct.t.ppf, 385, 433, 440
- Secondary y-axis, 40
- Second derivative, 181, 185–187, 189–192, 224
- SET OBJECTIVE, 450, 451
- Sets { }, 34
- Shape, 17, 36–38, 41–45, 47, 49, 53, 65, 70, 80, 123, 129, 133, 135, 137, 139, 144, 187, 283, 296, 315, 357, 364, 439, 459
- signum, 93, 279
- Sine function, 27, 57, 69, 72, 187, 188, 190, 193, 195, 197, 257
- SI units, 273
- Slicing, 2, 12, 34, 97, 151, 191, 205, 206, 225, 244
- Slider, 12, 13, 15, 16, 19, 20, 24–26, 60, 71, 77, 79, 80, 110, 111, 117, 179, 184, 185, 200, 201, 211, 227, 231, 232, 239, 244, 249, 255, 260, 263, 449, 450, 452, 453
- SLSQP, 459
- sm.add_constant, 429
- sm.OLS, 429
- SOLVER, 14, 419, 421, 448–454, 456, 459, 460, 469, 472–474
- Solver algorithm, 449, 452, 458, 461, 473
- Sound, 4, 57, 58, 92, 95–97, 112, 116, 228, 258–261, 263–265, 285, 309
- Sound barrier, 258–260, 265
- Sound wave, 258–260, 263
- stackoverflow.com, 35, 158
- Standard deviation, 14, 293, 294, 318–322, 324, 333, 334, 340, 342–344, 358–360, 365, 367, 375–378, 382, 385, 386, 394, 397–405, 407, 408, 410, 411, 413, 417, 423–426, 431, 435, 436, 445, 460–462, 464, 465, 469, 470
- Standard error, 14, 325, 365–367, 369–372, 374–380, 382–384, 386, 387, 390, 391, 393, 399, 408, 410–412, 417, 421, 424–429, 431–433, 435–437, 440, 442, 445–448, 461–463, 465–468, 471, 473
- Standard error range, 367, 369, 370, 376–380, 382, 421, 424, 425, 427, 428, 442, 446
- Static friction, 112, 287, 291
- STDEV.S, 320
- Steiner's, 228, 251, 254, 256, 267
- Straight line, 9, 43, 70, 76, 78–80, 87, 92–95, 102–104, 110, 112, 113, 179, 208, 228, 238, 260, 265, 269, 350, 384, 403, 406, 419, 420, 423–426, 428–430, 440, 442, 448–451, 453–455, 457, 472–474
- Straight-line segment, 176, 177, 181–184
- Student's t, 14, 382, 384, 390, 408, 409, 411, 431, 433, 439, 465
- Subplots, 108, 234
- SUB SCROLLBAR1_CHANGE(), 26
- SUMPRODUCT, 211, 223–225, 253, 254, 256
- SUMSQ, 223
- SUMXMY2, 184, 223
- Support points, 269, 271
- Swinging anchor, 228, 256
- Symmetric difference, 34, 329, 330
- System of equations, 225
- Systems of linear equations, 181, 182, 215
- T**
- t adaptation, 368, 423, 444, 460, 463, 470
- Tangent, 13, 99, 102, 106, 108, 182, 186, 204–207, 223, 264, 265
- Tangential vector, 230
- T.DIST.2T, 383, 473
- Text processing, 151, 152, 157, 158, 176, 355
- T.INV.2T, 383
- Trace to the predecessor, 106
- Translation, 227, 228, 238, 239, 243
- TRANSPOSE, 45
- Trapezoidal rule of integration, 224
- TRENDLINE, 367, 407, 422, 427, 437, 441, 445
- TRENDLINE OPTIONS, 437

- Triangle, 60, 63, 112, 123, 125, 132–134, 136, 137, 139, 140, 142
- t statistics, 365, 367, 377, 382, 384, 388, 410, 419, 431
- Tuple, 41, 142, 330, 332
- Turtle, 115, 135–137, 139–143, 148, 150
- TypeError, 94
- U**
- Uncertainty relation, 59, 61
- Uniform motion, 230
- Union, 34, 327, 328, 330, 394
- Unit matrix, 45, 48
- Unsupported operand type, 325
- Unwrapping a list, 94
- Upper envelope, 17, 19, 49–51, 56
- V**
- ValueError, 33, 36
- Variance, 14, 320, 334, 366, 367, 375, 380, 390, 391, 393, 395–398, 400, 407, 408, 412, 422, 423, 433, 442
- VAR.S, 320
- VBA macro to a control element, 26
- VBA terms, 155
- Vector addition, 199
- Vector product, 117, 170–173
- Vectors, 10, 12, 13, 16, 27, 29, 41–48, 50, 51, 53, 63, 65–67, 110, 113, 117, 168, 170–174, 181, 182, 185, 198–202, 207–211, 232, 237
- Vectors in polar and Cartesian coordinates, 198
- Vectors in the plane, 13, 182, 198
- Vectors with Python, 185
- Vertex form, 17, 19
- W**
- Wave crests, 258–264
- Wavefronts, 228, 258, 259, 261
- Wave-particle dualism, 348
- Weighted average, 182, 212, 213, 271, 276, 281, 389
- Weighted sum, 13, 253, 392, 397
- X**
- xy-plane, 27, 43, 80, 90, 198–202, 209, 211–213, 227, 230, 231, 241, 254, 258, 262, 266
- Z**
- zip, 330, 332