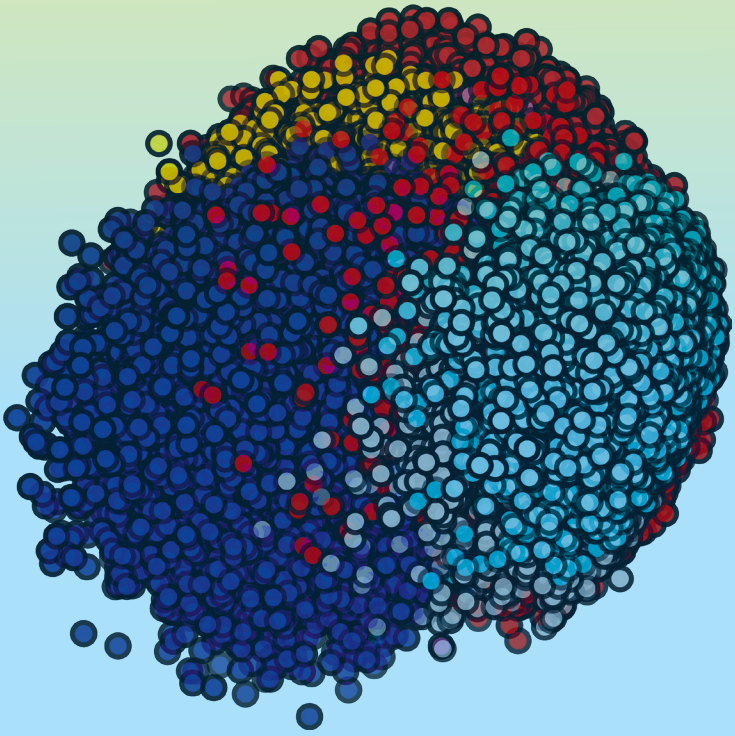
Omar Hijab

Math for Data Science



Springer

Math for Data Science

Omar Hijab

Math for Data Science

Springer

Omar Hijab
Department of Mathematics
Temple University
Philadelphia, PA, USA

ISBN 978-3-031-89706-1 ISBN 978-3-031-89707-8 (eBook)
https://doi.org/10.1007/978-3-031-89707-8

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

This work is subject to copyright. All rights are solely and exclusively licensed 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

If disposing of this product, please recycle the paper.

ਇਹ ਪੁਸਤਕ ਸਮਰਪਿਤ ਹੈ ਰਾਣੀ ਕੌਰ ਉਭੀ,
ਜਿਸ ਦੇ ਅਟੁੱਟ ਸਮਰਥਨ ਨੇ ਇਸ ਨੂੰ ਅਸਲੀਅਤ ਬਣਾ ਦਿੱਤਾ।

This book is dedicated to Rani Kaur Ubhi.

Preface

This text is a presentation of the mathematics underlying Data Science. The text assumes the math background typical of an Electrical Engineering undergraduate. In particular, we assume the reader has some prior calculus exposure.

By contrast, because we outsource computations to Python, and focus on conceptual understanding, Chapter 2, *Linear Geometry*, is developed in depth.

Depending on the emphasis and supplementary material, the text is appropriate for a course in the following programs

- Applied Mathematics,
- Business Analytics,
- Computer Science,
- Data Science,
- Engineering.

The level and pace of the text varies from gentle, at the start, to advanced, at the end. Depending on the depth of coverage, the text is appropriate for a one-semester course, or a two-semester course.

Chapters 1-3, together with some of the appendices, form the basis for a leisurely one-semester course, and Chapters 4-7 form the basis for an advanced one-semester course.

The chapter ordering is chosen to allow for the two semesters being taught simultaneously. The text was written while being repeatedly taught as a two-semester course with the two semesters taught simultaneously.

The culmination of the text is Chapter 7, *Machine Learning*. Much of the mathematics developed in prior chapters is used here. While only an introduction to the subject, the material in Chapter 7 is carefully built up from first principles.

As a consequence, the presentation and some results are new: The proofs of heavy ball convergence and Nesterov convergence for accelerated gradient descent are simplifications of the proofs in [39], and the connection between

hyperplane inseparability of a multi-class dataset and the existence of LR hyperplanes is apparently not present in the literature.

Important principles or results are displayed in these boxes.

The ideas presented in the text are made concrete by interpreting them in Python code. The standard Python data science packages are used, and a Python index lists the functions used in the text. All code is available online as `Jupyter` notebooks (see Appendix B). Because Python is used to highlight concepts, the supporting code snippets are, as pedagogical tools, sometimes longer than necessary.

```
Python code is displayed in these boxes.
```

Because SQL is usually part of a data scientist's toolkit, an introduction to using SQL, from within Python, is included in an appendix. Also, in case the instructor wishes to de-emphasize it, integration is presented separately in an appendix. Other appendices cover combinations and permutations, the binomial theorem, the exponential function, complex numbers, asymptotics, and minimizers, to be used according to the instructor's emphasis and preferences.

The bibiliography at the end is a listing of the references accessed while writing the text. Throughout, we use *iff* to mean *if and only if*, and we use $\approx$ for asymptotic equality (§A.7).

To help navigate the text, in each section, to indicate a break, a new idea, or a change in direction, we use a ship's wheel .

Sections and figures are numbered sequentially within each chapter, and equations and exercises are numbered sequentially within each section, so §3.4 is the fourth section in the third chapter, Figure 4.14 is the fourteenth figure in the fourth chapter, (3.2.1) is the first equation in the second section of the third chapter, and Exercise **1.2.3** is the third exercise in the second section of the first chapter. Also, [1] cites the first entry in the references.

I would like to express my sincere gratitude for the support of E. L. Grinberg, my friend and colleague, throughout this project. This endeavor began at *Rawa*, and we are truly thankful for the generous hospitality they provided during that time.

Omar Hijab
Spring 2025

Contents

Preface vii

List of Figures xi

1 Datasets 1
- 1.1 Introduction 1
- 1.2 The MNIST Dataset 5
- 1.3 Averages and Vector Spaces 10
- 1.4 Mean and Variance 18
- 1.5 High Dimensions 35

2 Linear Geometry 41
- 2.1 Vectors and Matrices 41
- 2.2 Products 48
- 2.3 Matrix Inverse 60
- 2.4 Span and Linear Independence 67
- 2.5 Zero Variance Directions 83
- 2.6 Pseudo-Inverse 88
- 2.7 Projections 97
- 2.8 Basis 107
- 2.9 Rank 114

3 Principal Components 121
- 3.1 Geometry of Matrices 121
- 3.2 Eigenvalue Decomposition 125
- 3.3 Graphs 151
- 3.4 Singular Value Decomposition 166
- 3.5 Principal Component Analysis 175
- 3.6 Cluster Analysis 184

4 Calculus 187
- 4.1 Single-Variable Calculus 187
- 4.2 Entropy and Information 211
- 4.3 Multi-Variable Calculus 218
- 4.4 Back Propagation 227

4.5 Convexity . . . 240

5 Probability . . . 261
5.1 Probability . . . 261
5.2 Binomial Probability . . . 276
5.3 Random Variables . . . 291
5.4 Normal Distribution . . . 317
5.5 Chi-squared Distribution . . . 333
5.6 Multinomial Probability . . . 345

6 Statistics . . . 359
6.1 Estimation . . . 359
6.2 Z-test . . . 364
6.3 T-test . . . 374
6.4 Chi-Squared Tests . . . 380

7 Machine Learning . . . 391
7.1 Overview . . . 392
7.2 Neural Networks . . . 393
7.3 Gradient Descent . . . 413
7.4 Network Training . . . 425
7.5 Linear Regression . . . 434
7.6 Logistic Regression . . . 440
7.7 Regression Examples . . . 450
7.8 Strong Convexity . . . 461
7.9 Accelerated Gradient Descent . . . 468

A Auxiliary Material . . . 475
A.1 Permutations and Combinations . . . 475
A.2 The Binomial Theorem . . . 482
A.3 The Exponential Function . . . 488
A.4 Two Dimensions . . . 498
A.5 Complex Numbers . . . 518
A.6 Integration . . . 528
A.7 Asymptotics and Convergence . . . 535
A.8 Existence of Minimizers . . . 541
A.9 SQL . . . 545

B Auxiliary Files . . . 557

References . . . 561

Python Index . . . 563

Index . . . 567

List of Figures

1.1 Iris dataset [27]. 2
1.2 Images in the MNIST dataset. 3
1.3 A portion of the MNIST dataset. 5
1.4 Original and projections: $n = 784$, 600, 350, 150, 50, 10, 1. 6
1.5 The MNIST dataset (Three-dimensional projection). 7
1.6 A crude copy of the image. 8
1.7 HTML colors. 11
1.8 The vector v joining the points μ and x. 12
1.9 Datasets of points versus datasets of vectors. 13
1.10 A dataset with its mean. 15
1.11 Vectorization of samples. 17
1.12 MSD for the mean (green) versus MSD for a random point (red). 19
1.13 Projecting a vector b onto the line through u. 25
1.14 Unit variance ellipses (blue) and unit inverse variance ellipses (red) with $\mu = 0$. 27
1.15 Variance ellipses (blue) and inverse variance ellipses (red) for a dataset. 30
1.16 Unit variance ellipse and unit inverse variance ellipse with standard Q. 31
1.17 Positively and negatively correlated datasets (inverse ellipses). 32
1.18 Ellipsoid and axes in 3d. 33
1.19 Disks inside the square. 35
1.20 Balls inside the cube. 36
1.21 Suspensions of interval $[a, b]$ and disk D. 39

2.1 Numpy column space array. 71
2.2 The points 0, x, Ax, and b. 89
2.3 The points x, Ax, the points x^*, Ax^*, and the point x^+. 89
2.4 Projecting onto a line. 98
2.5 Projecting onto a plane, $Pb = ru + sv$. 99

2.6 Relations between vector classes. 108
2.7 First defect for MNIST. 110
2.8 The dimension staircase with defects. 111
2.9 The dimension staircase for the MNIST dataset. 112
2.10 A 5×3 matrix A is a linear transformation from $\mathbf{R}^3$ to $\mathbf{R}^5$. . . . 114

3.1 Image of unit circle. 123
3.2 SVD decomposition $A = USV$. 124
3.3 Relations between matrix classes. 125
3.4 Inverse variance ellipse and centered dataset. 135
3.5 $S = span(v_1)$ and $T = S^{\perp}$. 138
3.6 Three springs at rest and perturbed. 143
3.7 Six springs at rest and perturbed. 143
3.8 Two springs along a circle leading to $Q(2)$. 144
3.9 Five springs along a circle leading to $Q(5)$. 144
3.10 Plot of eigenvalues of $Q(50)$. 148
3.11 Density of eigenvalues of $Q(d)$ for d large. 149
3.12 Directed and undirected graphs. 151
3.13 A weighed directed graph. 152
3.14 A double edge and a loop. 152
3.15 The complete graph K_6, the cycle graph C_6, and the wheel graph W_6. 153
3.16 The triangle $K_3 = C_3$. 154
3.17 An Eulerian graph. 157
3.18 An eighteenth-century map of Königsberg showing the seven bridges. 157
3.19 Non-isomorphic graphs with degree sequence $(3, 2, 2, 1, 1, 1)$. . . 162
3.20 Complete bipartite graph $K_{5,3}$. 163
3.21 A graph. 166
3.22 MNIST eigenvalues as a percentage of the total variance. 177
3.23 MNIST eigenvalue percentage plot. 177
3.24 Original and projections: $n = 784$, 600, 350, 150, 50, 10, 3. 181
3.25 The MNIST dataset (Two-dimensional projection). 182
3.26 The Iris dataset (Two-dimensional projection). 184

4.1 $f'(a)$ is the slope of the tangent line at a. 188
4.2 Composition of two functions. 190
4.3 Increasing or decreasing? 193
4.4 Increasing or decreasing? 194
4.5 The logarithm function $\log x$. 197
4.6 Tangent parabolas $p_m(x)$ (green), $p_L(x)$ (red), $L > m > 0$. 201
4.7 The sine function. 206
4.8 The sine function with $\pi/2$ tick marks. 207
4.9 Angle θ in the plane, $P = (x, y)$. 208
4.10 The absolute entropy function $H(p)$. 212

4.11 The absolute information $I(p)$. 213
4.12 The relative information $I(p, q)$ with $q = .7$. 216
4.13 Surface plot of $I(p, q)$ over the square $0 \leq p \leq 1$, $0 \leq q \leq 1$. . . . 217
4.14 Composition of multiple functions. 222
4.15 Composition of three functions in a chain. 227
4.16 A network composition [33]. 231
4.17 The function $g = \max(y, z)$. 232
4.18 Forward and backward propagation [33]. 232
4.19 Graph, directed graph, weighed directed graph, network. 236
4.20 A network with outgoing signals. 239
4.21 Another network. 239
4.22 Level sets and sublevel sets in two dimensions. 240
4.23 Level curves in two dimensions. 241
4.24 Line segment $[x_0, x_1]$. 242
4.25 Convex hull of x_1, x_2, x_3, x_4, x_5, x_6, x_7. 243
4.26 A convex hull with one facet highlighted. 244
4.27 A convex set K has a unique nearest point to any x_0. 245
4.28 Hyperplanes in two and three dimensions. 245
4.29 Separating hyperplane I. 247
4.30 tK lies in the interior of K: $x_t = (1 - t)\mu + tx$ is in the interior of K. 250
4.31 Hyperplane separation theorem. 251
4.32 $y = 1/x$ is a convex function. 259

5.1 Uniform probability density function. 265
5.2 100,000 sessions, with 5, 15, 50, and 500 tosses per session. . . . 269
5.3 The histogram of Iris petal lengths. 271
5.4 Iris petal lengths sampled 100,000 times. 272
5.5 Iris petal lengths batch means sampled 100,000 times, batch sizes 3, 5, 20. 273
5.6 Asymptotics of binomial coefficients. 281
5.7 The posterior density of p given 7 heads in 10 tosses. 285
5.8 The logistic function takes real numbers to probabilities. 287
5.9 The logistic function. 288
5.10 Decision boundary in $\mathbf{R}$. 289
5.11 Decision boundary in $\mathbf{R}^3$. 290
5.12 When we sample X, we get x. 291
5.13 Probability mass function $p(x)$ of a Bernoulli random variable. 298
5.14 Cumulative distribution function $F(x)$ of a Bernoulli random variable. 298
5.15 Confidence that X lies in interval $[a, b]$. 307
5.16 Continuous cumulative distribution function. 310
5.17 When we sample X_1, X_2, . . . , X_n, we get x_1, x_2, . . . , x_n. . . . 313
5.18 The `pdf` of the standard normal distribution. 317
5.19 The binomial `cdf` and its CLT normal approximation. 322

5.20 `z = Z.ppf(p)` and `p = Z.cdf(z)` . . . 324
5.21 Confidence (green) or significance (red) (lower-tail, two-tail, upper-tail). . . . 325
5.22 68%, 95%, 99% confidence cutoffs for standard normal. . . . 326
5.23 68%, 95%, 99% cutoffs for non-standard normal. . . . 328
5.24 (X, Y) inside the square and inside the disk. . . . 333
5.25 Chi-squared distribution with different degrees. . . . 335
5.26 With degree $d \geq 2$, the chi-squared density peaks at $d - 2$. . . . 337
5.27 Normal probability density on $\mathbf{R}^2$. . . . 340
5.28 The softmax function takes vectors to probability vectors. . . . 347

6.1 Statistics flowchart: p-value p and significance α. . . . 360
6.2 Histogram of sampling $n = 25$ students, repeated $N = 1000$ times. . . . 364
6.3 Student distribution, against normal (dashed). . . . 375

7.1 Perceptrons in parallel (R in the figure is the retina) [21]. . . . 395
7.2 A perceptron: weights, incoming signals, and outgoing signals. 396
7.3 A perceptron with bias: weights and outgoing signals. . . . 396
7.4 Neural network: weights. . . . 398
7.5 Neural network: incoming signals. . . . 399
7.6 Neural network: outgoing signals. . . . 399
7.7 Downstream, local, and upstream derivatives at node i. . . . 406
7.8 Neural network: downstream derivatives. . . . 407
7.9 A shallow dense layer with a bias input. . . . 410
7.10 Layered neural network [11]. . . . 411
7.11 Neural network with biases . . . 412
7.12 A network with six neurons, two outputs, and one input. . . . 412
7.13 Downstream, local, and upstream derivatives at termination. . . . 413
7.14 Double well newton descent. . . . 415
7.15 Double well function and sublevel sets at w_0 and at w_1. . . . 418
7.16 Double well gradient descent. . . . 420
7.17 Neural network: weight gradients. . . . 426
7.18 Loss decay as learning rate varies: single sample training. . . . 428
7.19 Loss decay as learning rate varies: batch sample training. . . . 431
7.20 Linear regression neural network. . . . 436
7.21 Logistic regression neural network. . . . 442
7.22 Population versus employed: linear regression. . . . 451
7.23 Polynomial regression: Degrees 2, 4, 6, 8, 10, 12. . . . 455
7.24 Hikers dataset: x. . . . 457
7.25 Hikers dataset: (x, q) [37]. . . . 457
7.26 Hikers dataset: (x, x_0). . . . 458
7.27 Neural network for hiker outcomes. . . . 459
7.28 Equivalent neural network for hiker outcomes. . . . 460
7.29 Hiker dataset: (x, x_0, q). . . . 460

7.30 Convex hulls of Iris classes in $\mathbf{R}^2$. 461
7.31 Convex hulls of MNIST classes in $\mathbf{R}^2$. 461

A.1 $6 = 3!$ arrangements of 3 balls. 476
A.2 The exponential function $\exp x$. 492
A.3 Convexity of the exponential function. 496
A.4 A vector v. 498
A.5 Vectors v_1 and v_2 and their shadows in the plane. 498
A.6 Adding v_1 and v_2 499
A.7 Scaling with $t = 2$ and $t = -2/3$ 500
A.8 The polar representation of $v = (x, y)$. 501
A.9 v and its antipode $-v$. 502
A.10 Two vectors v_1 and v_2. 503
A.11 Pythagoras for general triangles. 505
A.12 Proof of Pythagoras for general triangles. 506
A.13 P and $P^\perp$ and v and $v^\perp$. 507
A.14 Multiplying and dividing points on the unit circle. 519
A.15 Complex numbers 521
A.16 The second, third, and fourth roots of unity 524
A.17 The fifth, sixth, and fifteenth roots of unity 525
A.18 Complex conjugate roots $\rho_\pm$. 527
A.19 Areas under the graph. 529
A.20 Area under the parabola. 531
A.21 The graph and area under $\sin x$. 532
A.22 Integral of $\sin x/x$. 533
A.23 Dataframe from list-of-dicts. 548
A.24 Menu dataframe and SQL table. 549
A.25 Rawa restaurant. 551
A.26 OrdersIn dataframe and SQL table. 552
A.27 OrdersOut dataframe and SQL table. 553

Chapter 1
Datasets

In this chapter we explore examples of datasets and some simple Python code. We introduce the mean and variance of a dataset, then present a first taste of what higher dimensions might look like.

1.1 Introduction

A *sample*, an *example*, or an *observation* is a list of numbers, and a *dataset* is a collection of samples. If x is a sample, $x = (t_1, t_2, \ldots, t_d)$, the numbers t_1, t_2, $\ldots$, t_d are its *features* or *attributes*.

When manipulating datasets as collections of vectors, they are usually arranged into $d \times N$ arrays. When displaying datasets, as in spreadsheets or SQL tables, they are usually arranged into $N \times d$ arrays.

Practically speaking, the following are all representations of datasets

$$matrix = CSV\ file = spreadsheet = SQL\ table = array = dataframe$$

Geometrically, a dataset is a collection of N points x_1, x_2, $\ldots$, x_N in d-dimensional *sample space* $\mathbf{R}^d$.

Often one or several of the features are separated out as the *label* or *target*. In this case, the remaining features are the *source* or *input*, and each sample is a source-target pair. Such a dataset is a *labeled* or *targeted* dataset.

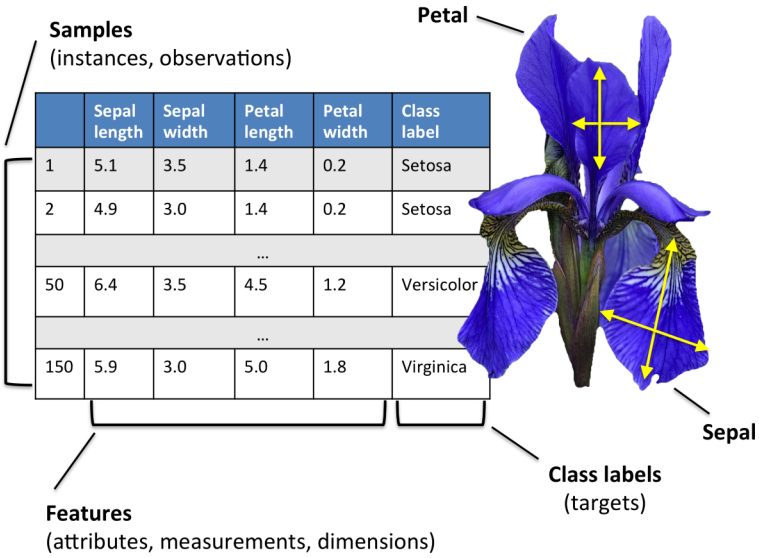


The *Iris dataset* contains $N = 150$ examples of $d = 4$ features of Iris flowers, and there are three classes of Irises, *Setosa*, *Versicolor*, and *Virginica*, with 50 samples from each class. For each example, the class is the label corresponding to that example, so the Iris dataset is labeled.

The four features are sepal length and width, and petal length and width. In Figure 1.1, the dataset is displayed as an $N \times d$ array.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

O. Hijab, *Math for Data Science*, https://doi.org/10.1007/978-3-031-89707-8_1



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica

Fig. 1.1 Iris dataset [27].

The Iris dataset is loaded using the code[1]

```
from sklearn import datasets

iris = datasets.load_iris()
iris["feature_names"]
```

This returns

`['sepal length','sepal width','petal length','petal width']`.

To return the data and the classes, the code is

```
dataset = iris["data"]
labels = iris["target"]

dataset, labels
```

The above code returns `dataset` as an $N \times d$ array. To return a $d \times N$ array, take the transpose `dataset = iris["data"].T`.



The *MNIST dataset* consists of images of hand-written digits (Figure 1.2). There are 10 classes of images, corresponding to each digit 0, 1, ..., 9. We

[1] `sklearn` must be installed prior to running this code, see the end of this section.

seek to compress the images while preserving as much as possible of the images' characteristics.

Each image is a grayscale 28x28 pixel image. Since $28^2 = 784$, each image is a point in $d = 784$ dimensions. Here there are $N = 60000$ samples and $d = 784$ features.

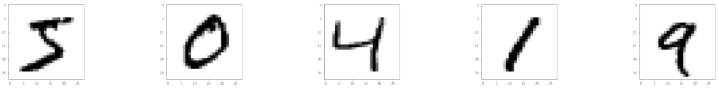


Fig. 1.2 Images in the MNIST dataset.



This subsection is included just to give a flavor. *All unfamiliar words are explained in detail in Chapter 2.* If preferred, just skip to the next subsection.

Suppose we have a dataset of N points

$$x_1, x_2, \ldots, x_N$$

in d-dimensional sample space. We seek to find a lower-dimensional space $U \subset \mathbf{R}^d$ so that the projections of these points onto U retain as much information as possible about the data.

In other words, we are looking for an n-dimensional subspace U for some $n < d$. Among all n-dimensional subspaces, which one should we pick? The answer is to select the subspace U among all n-dimensional subspaces that retains as much information about the dataset as possible.

Another issue is the choice of n, which is an integer satisfying $0 \leq n \leq d$. On the one hand, we want n to be as small as possible, to maximize data compression. On the other hand, we want n to be big enough to capture most of the features of the data. At one extreme, if we pick $n = d$, then we have no compression and complete information. At the other extreme, if we pick $n = 0$, then we have full compression and no information.

Projecting the data from $\mathbf{R}^d$ to a lower-dimensional space U is *dimensional reduction.* The best alignment, the best fit, or the best choice of U is *principal component analysis.* These issues will be taken up in §3.5.



If this is your first exposure to data science, there will be a learning curve, because here there are three kinds of thinking: Data science (datasets, PCA, descent, networks), math (linear algebra, probability, statistics, calculus), and Python (`numpy`, `pandas`, `scipy`, `sympy`, `matplotlib`). It may help to read the

`code examples`, and the `important math principles` first, then dive into details as needed.

To illustrate and make concrete concepts as they are introduced, we use Python code throughout. We run Python code in a `Jupyter` notebook.

All code in this text is available online as `Jupyter` notebooks, one per section, at

https://mathdatasciencebook.com.

The code is arranged (see Appendix B) so that each notebook runs on its own in the order presented. The only exception is this section, where the `pip install` code below must be run first.

`Jupyter` is an IDE, an integrated development environment. `Jupyter` supports many languages, including Python, Sage, Julia, and R. A useful `Jupyter` feature is the ability to measure the amount of execution time of a `Jupyter` cell by including at the start of the cell

```
%%time
```

It's simplest to first install Python, then `Jupyter`. To minimize overhead, it's best to install Python using a package built for your laptop's OS, and avoid extra packages or frameworks. If Python is installed from

https://www.python.org/downloads,

then the Python package installer `pip` is also installed.

To install `Jupyter`, *from within an OS shell,* enter

```
pip install jupyter
```

After installing `Jupyter`, all other packages are installed from within `Jupyter`. For this text, *from within a* `Jupyter` *cell,* we ran

```
pip install numpy sympy scipy scikit-learn pandas matplotlib
    ↪ ipympl sqlalchemy pymysql import_ipynb
```

After installing these packages, restart `Jupyter` to activate the packages. The above is a complete listing of the packages used in this text.

Optional step: Because one often has to repeatedly install different versions of Python, it's best to isolate your installations from whatever Python your laptop's OS uses. This is achieved by installing a copy of Python within a `venv`, a *virtual environment*. Then several `venv`s may be set up side-by-side, and, at any time, any `venv` may be deleted without impacting any others, or the OS. For more on this step, see Appendix B.

Exercises

Exercise 1.1.1 What is `dataset.shape` and `labels.shape`?

Exercise 1.1.2 What does `sum(dataset[0])` return and why?

Exercise 1.1.3 What does `sum(dataset)` return and why?

Exercise 1.1.4 Let `a` be a list. What does `list(enumerate(a))` return? What does the code below return?

```
def uniq(a):
   return [ x for i, x in enumerate(a) if x not in a[:i] ]
```

1.2 The MNIST Dataset

The MNIST dataset consists of 60,000 training images. Since this dataset is for demonstration purposes, these images are coarse.



Fig. 1.3 A portion of the MNIST dataset.

Each image consists of $28 \times 28 = 784$ pixels, and each pixel shading is a byte, an integer between 0 and 255 inclusive. Therefore each image is a point x in $\mathbf{R}^d = \mathbf{R}^{784}$. Attached to each image is its label, a digit 0, 1, ..., 9.

We assume the dataset is loaded onto your laptop as a CSV file `mnist.csv` (see Appendix B), with each row in the file consisting of the pixels for a single image, together with the image label: Each row starts with the image's label, followed by 784 bytes. The code

```
from pandas import *
from numpy import *

mnist = read_csv("mnist.csv").to_numpy()

# separate rows into data and labels
# first column is the labels
labels = mnist[:,0]
# all other columns are the pixels
dataset = mnist[:,1:]

mnist.shape, dataset.shape, labels.shape
```

returns `(60000, 785)`, `(60000, 784)`, `(60000,)`.

To *compress* the image means to reduce the number of dimensions in the point x while keeping maximum information. We can think of a single image as a dataset itself, and compress the image, or we can design a compression algorithm based on a collection of images. It is then reasonable to expect that the procedure applies well to any image that is similar to the images in the collection.

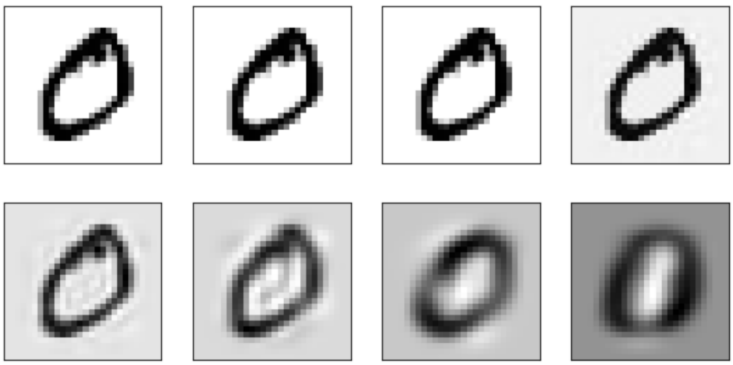


Fig. 1.4 Original and projections: $n = 784$, 600, 350, 150, 50, 10, 1.

For the second image in Figure 1.2, reducing dimension from $d = 784$ to n equal 600, 350, 150, 50, 10, and 1, we have the images in Figure 1.4.

Compressing each image to a point in $n = 3$ dimensions and plotting all $N = 60000$ points yields Figure 1.5. All this is discussed in §3.5.

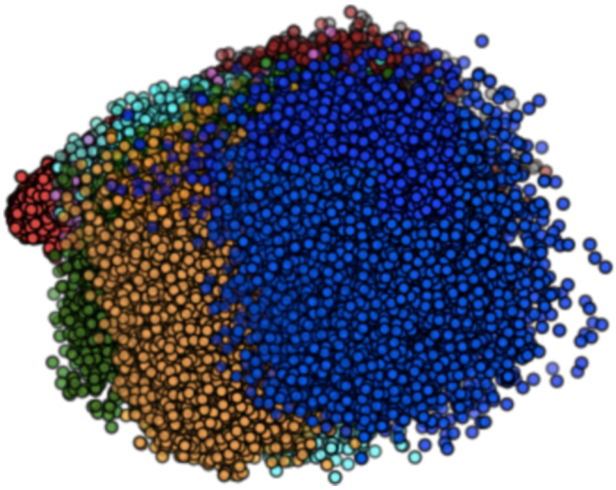


Fig. 1.5 The MNIST dataset (Three-dimensional projection).



Here is an exercise. The top left image in Figure 1.4 is given by a 784-dimensional point which is imported as an array `pixels`.

```
pixels = dataset[1].reshape((28,28))
```

Here `dataset[1]` is an array of shape `(784,)`, and `pixels` is an array of shape `(28,28)`.

1. In `Jupyter`, return a two-dimensional plot of the point $(2, 3)$ at size 50 using the code

```
from matplotlib.pyplot import *

grid()
scatter(2,3,s = 50)
show()
```

2. Do `for` loops over `i` and `j` in `range(28)` and use `scatter` to plot points at location `(i,j)` with size given by `pixels[i,j]`, then `show`.

Here is one possible code, returning Figure 1.6.

```
from matplotlib.pyplot import *
from numpy import *

pixels = dataset[1].reshape((28,28))

for i in range(28):
   for j in range(28):
      scatter(i,j, s = pixels[i,j])

grid()
show()
```

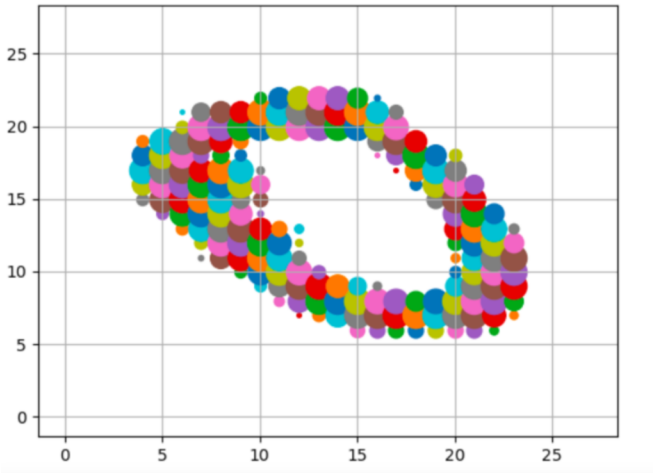


Fig. 1.6 A crude copy of the image.

The top left image in Figure 1.4 is returned by the code

```
from matplotlib.pyplot import *

imshow(pixels, cmap = "gray_r")

grid()
show()
```



In recent versions of `numpy`, floats are displayed as follows

```
np.float64(5.843333333333335)
```

To display floats without their type, as follows,

```
5.843333333333335
```

insert

```
from numpy import *
set_printoptions(legacy = "1.25")
```

at the top of your `Jupyter` notebook.



We end the section by discussing the Python `import` command. The code snippet above can be rewritten

```
import matplotlib.pyplot as plt

plt.imshow(pixels, cmap = "gray_r")

plt.grid()
plt.show()
```

or as

```
from matplotlib.pyplot import imshow, grid, show

imshow(pixels, cmap = "gray_r")
grid()
show()
```

So we have three versions of this code snippet.

In the second version, it is explicit that `imshow` is imported from the *module* `pyplot` of the *package* `matplotlib`. Moreoever, `matplotlib.pyplot` is referenced by a short nickname `plt`. A collection of Python modules is called a Python package.

In the first version `import from *`, many commands, maybe not all, are imported from the module `matplotlib.pyplot`.

In the third version, only the command `imshow` is imported. Which import style is used depends on the situation.

In this text, we usually use the first style, as it is visually lightest. To help with online searches, in the Python index, Python commands are listed under their full package path.

Exercises

Exercise 1.2.1 Run the code in this section on your laptop (all code is run within `Jupyter`).

Exercise 1.2.2 The first image in the MNIST dataset is an image of the digit 5. What is the 43,120th image?

Exercise 1.2.3 Figure 1.6 is not oriented the same way as the top-left image in Figure 1.4. Modify the code returning Figure 1.6 to match the top-left image in Figure 1.4.

1.3 Averages and Vector Spaces

Suppose we have a population of things (people, tables, numbers, vectors, images, etc.) and we have a sample of size N from this population:

```
# L = [x_1,x_2,...,x_N]
```

The total population is the *population* or the *sample space*. For example, the sample space consists of all real numbers and we take $N = 5$ samples from this population

```
L_1 = [3.95, 3.20, 3.10, 5.55, 6.93]
```

Or, the sample space consists of all integers and we take $N = 5$ samples from this population

```
L_2 = [35, -32, -8, 45, -8]
```

Or, the sample space consists of all rational numbers and we take $N = 5$ samples from this population

```
L_3 = [13/31, 8/9, 7/8, 41/22, 32/27]
```

Or, the sample space consists of all Python strings and we take $N = 5$ samples from this population

```
L_4 = ['a2e?',r'#%T','7y5,','kkk>><</','[[)*+']
```

Or, the sample space consists of all HTML colors and we take $N = 5$ samples from this population



Fig. 1.7 HTML colors.

Here's the code generating the colors

```
# HTML color codes are #rrggbb (6 hexes)
from matplotlib.pyplot import *

from random import choice

def hexcolor():
   chars = '0123456789abcdef'
   return "#" + ''.join([choice(chars) for _ in range(6)])

for i in range(5): scatter(i,0, c = hexcolor())
show()
```



Let L be a list as above. The goal is to compute the sample *average* or *mean* of the list, which is

$$\mu = \frac{x_1 + x_2 + \cdots + x_N}{N}. \tag{1.3.1}$$

In the first example, for real numbers, the average is

$$\frac{3.95 + 3.20 + 3.10 + 5.55 + 6.93}{5} = 4.546.$$

In the second case, for integers, the average is 32/5. In the third case, the average is 385373/73656. In the fourth case, while we can add strings, we can't divide them by 5, so the average is undefined. Similarly for colors: the average is undefined.

This leads to an important definition. A sample space or population V is called a *vector space* if, roughly speaking, one can compute means or averages in V. In this case, we call the members of the population "vectors", even though the members may be anything, as long as they satisfy the basic rules of a vector space.

In a vector space V, the rules are:

1. vectors v, w can be added, yielding sum $v + w$,
2. vector addition is commutative, $v + w = w + v$,
3. vector addition is associative, $u + (v + w) = (u + v) + w$,

4. there is a zero vector 0,
5. vectors v have negatives $-v$, $v + (-v) = 0$,
6. vectors v can be scaled to rv by real numbers r,
7. double-scaling is the same as multiplying scalars, $r(sv) = (rs)v$,
8. scaling is distributive over addition both ways,

$$(r + s)v = rv + sv, \qquad r(u + v) = ru + rv,$$

9. scaling v by 1 and 0 returns the vector v and the zero vector, $1v = v$ and $0v = 0$.



Let $x_1, x_2, \ldots, x_N$ be a dataset. Is the dataset a collection of points, or is the dataset a collection of vectors? In other words, what geometric picture of datasets should we have in our heads? Here's how it works.

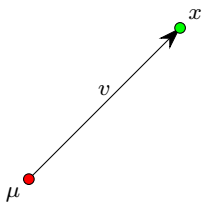


Fig. 1.8 The vector v joining the points μ and x.

A *vector* is an arrow joining two points (Figure 1.8). Given two points $\mu = (a, b)$ and $x = (c, d)$, the vector joining them is

$$v = x - \mu = (c - a, d - b).$$

Then μ is the *tail* of v, and x is the *head* of v. For example, the vector joining $\mu = (1, 2)$ to $x = (3, 4)$ is $v = (2, 2)$.

Given a point x, we would like to associate to it a vector v in a uniform manner. However, this cannot be done without a second point, a *reference point*. Given a dataset of *points* $x_1, x_2, \ldots, x_N$, the most convenient choice for the reference point is the mean μ of the dataset. This results in a dataset of *vectors* $v_1, v_2, \ldots, v_N$, where $v_k = x_k - \mu$, $k = 1, 2, \ldots, N$.

The dataset $v_1, v_2, \ldots, v_N$ is *centered*, its mean is zero,

$$\frac{v_1 + v_2 + \cdots + v_N}{N} = 0.$$

So datasets can be points $x_1, x_2, \ldots, x_N$ with mean μ, or vectors $v_1, v_2, \ldots, v_N$ with mean zero (Figure 1.9).

Centered Versus Non-Centered

If $x_1, x_2, \ldots, x_N$ is a dataset of points with mean μ and

$$v_1 = x_1 - \mu, v_2 = x_2 - \mu, \ldots, v_N = x_N - \mu,$$

then $v_1, v_2, \ldots, v_N$ is a centered dataset of vectors.

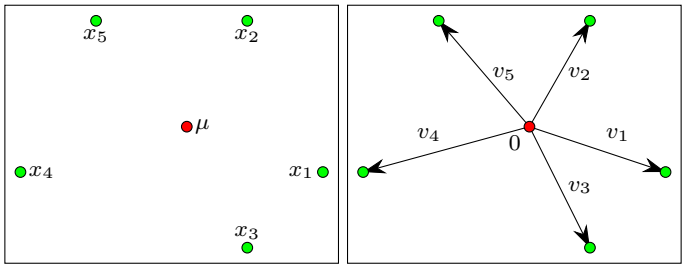


Fig. 1.9 Datasets of points versus datasets of vectors.



Let us go back to vector spaces. When we work with vector spaces, numbers are referred to as *scalars*, because $2v$, $3v$, $-v$, ... are scaled versions of v. When we multiply a vector v by a scalar r to get the scaled vector rv, we call this vector *scaling*. This is to distinguish this multiplication from the inner and outer products we see below.

For example, the samples in the list L_1 form a vector space, the set of all real numbers $\mathbf{R}$. Even though one can add integers, the set $\mathbf{Z}$ of all integers does not form a vector space because multiplying an integer by $1/2$ does not result in an integer. The set $\mathbf{Q}$ of all rational numbers (fractions) is a vector space, so L_3 is a sampling from a vector space. The set of strings is not a vector space because even though one can add strings, addition is not commutative:

```
'alpha' + 'romeo' == 'romeo' + 'alpha'
```

returns `False`.



For the scalar dataset

$$x_1 = 1.23, x_2 = 4.29, x_3 = -3.3, x_4 = 555,$$

the average is

$$\mu = \frac{1.23 + 4.29 - 3.3 + 555}{4} = 139.305.$$

In Python, averages are computed using `numpy.mean`. For a scalar dataset, the code

```
from numpy import *
set_printoptions(legacy = "1.25")

dataset = array([1.23,4.29,-3.3,555])
mu = mean(dataset)
mu
```

returns the average.

For the two-dimensional dataset

$$x_1 = (1, 2), x_2 = (3, 4), x_3 = (-2, 11), x_4 = (0, 66),$$

the average is

$$\mu = \frac{(1, 2) + (3, 4) + (-2, 11) + (0, 66)}{4} = (0.5, 20.75).$$

Note the features are summed separately: the x-components are summed, and the y-components are summed, leading to a two-dimensional mean. (This is *vector addition*, taken up in §A.4.)

In Python, a dataset of four points in $\mathbf{R}^2$ may be presented as a 2×4 array or as a 4×2 array. As a 2×4 array, the code is

```
from numpy import *

dataset = array([[1,3,-2,0],[2,4,11,66]])
mu = mean(dataset, axis = 1)
```

returning the mean $(0.5, 20.75)$. Here the option `axis = 1` indicates we sum over the column index. This means summing the entries of the first row, then summing the entries of the second row, resulting in a mean with two components.

The default is to consider the row index i as index zero, and to consider the column index j as index one. With this understood, presenting the dataset as a 4×2 array, the code is

```
from numpy import *

dataset = array([[1,2],[3,4],[-2,11],[0,66]])
mu = mean(dataset, axis = 0)
```

returns the same mean.

To *transpose* a `numpy` array is to switch rows with columns, so the transpose `dataset.T` of a 2×4 array `dataset` is a 4×2 array.

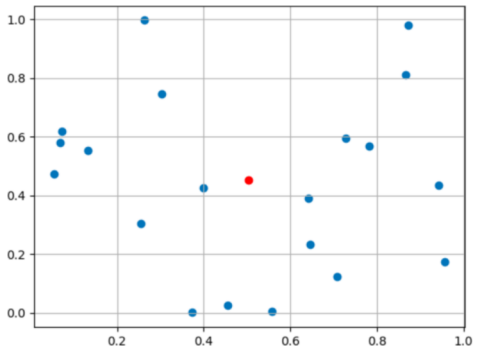


Fig. 1.10 A dataset with its mean.

Here is a more involved example of a dataset of random points and their mean:

```
from numpy import *
from matplotlib.pyplot import scatter, grid, show

from numpy.random import default_rng
samples = default_rng().random

N = 20

# 2xN array
dataset = samples((2,N))
mu = mean(dataset,axis = 1)

scatter(*mu)
scatter(*dataset)

grid()
show()
```

This returns Figure 1.10.

In this code, `scatter(x,y)` expects two positional arguments, the x and the y features, arranged as two scalars, for a single point, or two arrays of x

and y components separately, for several points. Similarly, `plot(x,y)` expects the x and y features as two separate arrays.

The *unpacking operator* `*` unpacks `mu` and `dataset` from one pair each into their separate x and y features `*mu` and `*dataset`. So while `mu` is one Python object, a `tuple`, `*mu` are two Python objects, two `float`s. For this to work, `dataset` must be $2 \times N$. When `dataset` is $N \times 2$, we unpack its transpose `*dataset.T`.

`default_rng` is the default random number generator in `numpy`. It is used throughout the text.

The code `random(tuple)` returns a random array with shape `tuple`, so `samples((2,N))` above returns a random $2 \times N$ array.



Sometimes, a population is not a vector space, so we can't take sample means from it. Instead, we take the sample mean of a scalar or vector computed from the samples. This computed quantity is a *statistic* associated to the population.

A statistic is an assignment of a scalar or vector $f(x)$ to each sample x from the population, and the sample mean is then

$$\frac{f(x_1) + f(x_2) + \cdots + f(x_N)}{N}. \tag{1.3.2}$$

Since scalars and vectors do form vector spaces, this mean is well-defined. For example, a population of cats is not a vector space (they can't be added), but their heights is a vector space (heights can be added). This process is *vectorization* of the samples.

Vectorization is frequently used to count proportions: Samples are drawn from finitely many categories, and we wish to count the *proportion* of samples belonging to a particular category.

If we toss a coin N times, we obtain a list of heads and tails,

$$H, H, T, T, T, H, T, \ldots$$

To count the proportion of heads, we define

$$f(x) = \begin{cases} (1,0), & \text{if } x \text{ is heads,} \\ (0,1), & \text{if } x \text{ is tails.} \end{cases}$$

If we add the vectorized samples $f(x)$ using vector addition in the plane (§A.4), the first component of the mean (1.3.2) is an average of ones and zeroes, with ones matching heads, resulting in the proportion $\hat{p}$ of heads. Similarly, the second component is the proportion of tails. Hence (1.3.2) is the pair $(\hat{p}, 1 - \hat{p})$, where $\hat{p}$ is the proportion of heads in N tosses.

More generally, if the label of a sample falls into d categories, we may let $f(x)$ be a vector with d components consisting of zeros and ones, according to the category of the sample. This is *one-hot encoding* (see §2.4 and §7.6).

For example, suppose we take a sampling of size N from the Iris dataset, and we look at the classes of the resulting samples. Since there are three classes, in this case, we can define $f(x)$ to equal

$$(1,0,0), \qquad (0,1,0), \qquad (0,0,1),$$

according to which class x belongs to. Then the mean (1.3.2) is a triple $\hat{p} = (\hat{p}_1, \hat{p}_2, \hat{p}_3)$ of proportions of each class in the sampling. Of course, $\hat{p}_1 + \hat{p}_2 + \hat{p}_3 = 1$, so $\hat{p}$ is a *probability vector* (§5.6).

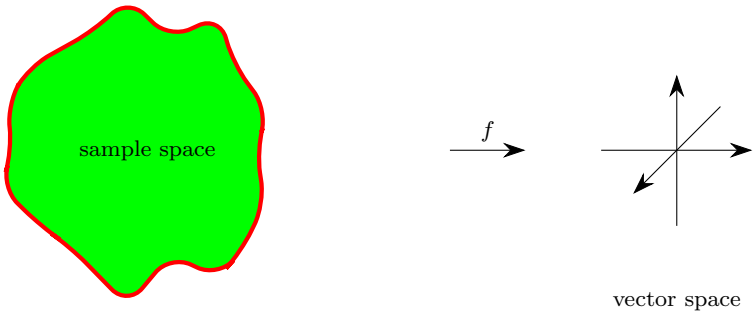


Fig. 1.11 Vectorization of samples.

When there are only two possibilities, two classes, it's simpler to encode the classes as follows,

$$f(x) = \begin{cases} 1, & \text{if } x \text{ is heads,} \\ 0, & \text{if } x \text{ is tails.} \end{cases}$$

Then the mean (1.3.2) is the proportion $\hat{p}$ of heads.



Even when the samples are already scalars or vectors, we may still want to vectorize them. For example, suppose $x_1, x_2, \ldots, x_N$ are the prices of a sample of printers from across the country. Then the average price (1.3.1) is well-defined. Nevertheless, we may set

$$f(x) = \begin{cases} 1, & \text{if } x \text{ is greater than \$100,} \\ 0, & \text{if } x \text{ is} \leq \$100. \end{cases}$$

Then the mean (1.3.2) is the sample proportion $\hat{p}$ of printers that cost more then $100.

In §6.4, we use vectorization to derive the chi-squared tests.

Exercises

Exercise 1.3.1 For the `dataset = array([[1,3,-2,0],[2,4,11,66]])`, the commands `mean(dataset,axis = 1)` and `mean(dataset,axis = 0)` return means in $\mathbf{R}^2$ and in $\mathbf{R}^4$. What does `mean(dataset)` return and why?

Exercise 1.3.2 What is the average petal length in the Iris dataset?

Exercise 1.3.3 What is the average shading of the pixels in the first image in the MNIST dataset?

Exercise 1.3.4 What's the difference between `plot` and `scatter` in

```
from numpy import *
from matplotlib.pyplot import scatter, plot

def f(x): return x**2

x = arange(0,1,.2)

plot(x,f(x))
scatter(x,f(x))

grid()
show()
```

1.4 Mean and Variance

Let x_1, x_2, …, x_N be a dataset in $\mathbf{R}^d$, and let x be any point in $\mathbf{R}^d$. The *mean-square distance* of x to D is

$$MSD(x) = \frac{1}{N}\sum_{k=1}^{N} |x_k - x|^2.$$

Above $|x|$ is the length of the vector x, so $|x_k - x|$ is the distance between x_k and x. In this section we focus on two dimensions $d = 2$. Background on vectors and matrices in two dimensions is in §A.4.

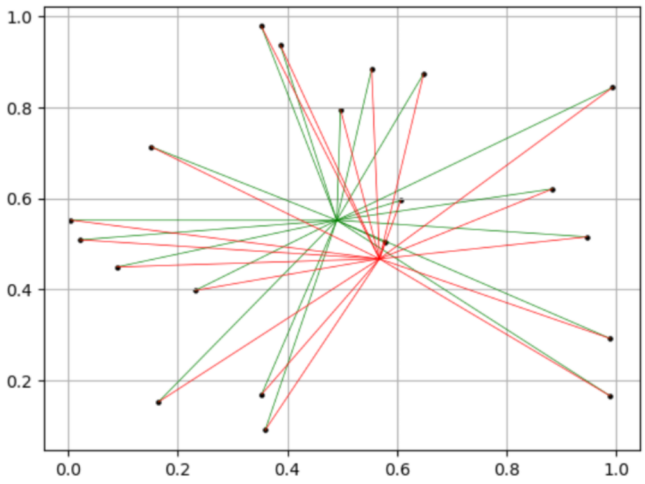


Fig. 1.12 MSD for the mean (green) versus MSD for a random point (red).

The *mean* or *sample mean* is

$$\mu = \frac{1}{N}\sum_{k=1}^{N} x_k = \frac{x_1 + x_2 + \cdots + x_N}{N}. \tag{1.4.1}$$

The mean μ is a point in $\mathbf{R}^d$. The first result is

Point of Best-fit

The mean is the point of best-fit: The mean minimizes the mean-square distance to the dataset (Figure 1.12).

Using (A.4.6),

$$|a+b|^2 = |a|^2 + 2a \cdot b + |b|^2$$

for vectors a and b, it is easy to derive the above result. Insert $a = x_k - \mu$ and $b = \mu - x$, then sum over $k = 1, 2, \ldots, N$, to get

$$MSD(x) = MSD(\mu) + \frac{2}{N}\sum_{k=1}^{N}(x_k - \mu)\cdot(\mu - x) + |\mu - x|^2.$$

Now the middle term vanishes

$$\begin{aligned}\frac{2}{N}\sum_{k=1}^{N}(x_k - \mu)\cdot(\mu - x) &= \frac{2}{N}\left(\left(\sum_{k=1}^{N} x_k\right) - N\mu\right)\cdot(\mu - x)\\ &= 2(\mu - \mu)\cdot(\mu - x) = 0,\end{aligned}$$

so we have

$$MSD(x) = MSD(\mu) + |x - \mu|^2.$$

Since $|x - \mu|^2 \geq 0$, the result follows. This result and its proof remain valid in any dimension.

Here is the code for Figure 1.12.

```
from matplotlib.pyplot import *
from numpy import *
set_printoptions(legacy = "1.25")

from numpy.random import default_rng
samples = default_rng().random
d, N = 2, 20

# d x N array
dataset = samples((d,N))

mu = mean(dataset,axis = 1)
p = samples((2,))

for v in dataset.T:
   points = array([mu,v])
   plot(*points.T,c = 'green')
   points = array([p,v])
   plot(*points.T,c = 'red')

scatter(*mu)
scatter(*dataset)

grid()
show()
```



The variance of a dataset is defined in any dimension d. When $d = 1$, the dataset consists of scalars $x_1, x_2, \ldots, x_N$. If the dataset has mean μ, we can center the dataset,

$$v_1 = x_1 - \mu, v_2 = x_2 - \mu, \ldots, v_N = x_N - \mu.$$

Then, for a scalar dataset, the *variance* q is $MSD(\mu)$,

$$q = \frac{v_1^2 + v_2^2 + \cdots + v_N^2}{N} = \frac{1}{N} \sum_{k=1}^{N} v_k^2. \tag{1.4.2}$$

Since the variance is nonnegative, we may take its square root: The square root of the variance is the *standard deviation* $\sigma = \sqrt{q}$.

If a scalar dataset has mean zero and variance one, it is *standard*. If the variance q of a scalar dataset is not one, it may be standardized by centering and dividing by the standard deviation,

$$x'_k = \frac{x_k - \mu}{\sigma}, \qquad k = 1, 2, \ldots, N. \tag{1.4.3}$$

The resulting dataset x'_1, x'_2, …, x'_N is then standard,

$$\mu' = \frac{1}{N}\sum_{k=1}^{N} x'_k = 0, \qquad q' = \frac{1}{N}\sum_{k=1}^{N} {x'_k}^2 = 1.$$



Now suppose the dataset consists of points x_1, x_2, …, x_N in $\mathbf{R}^d$. If the dataset has mean μ, we can center the dataset,

$$v_1 = x_1 - \mu, v_2 = x_2 - \mu, \ldots, v_N = x_N - \mu.$$

Then the *variance* is the matrix (see §A.4 for tensor product)

$$Q = \frac{v_1 \otimes v_1 + v_2 \otimes v_2 + \cdots + v_N \otimes v_N}{N} = \frac{1}{N}\sum_{k=1}^{N} v_k \otimes v_k. \tag{1.4.4}$$

Since $v \otimes v$ is a symmetric matrix, Q is a symmetric matrix. Below we see Q is also nonnegative, in the sense $v \cdot Qv \geq 0$ for all vectors v.



Let us unpack the variance Q in two dimensions $\mathbf{R}^2$. When $d = 2$, each sample x_k has two features, so we may write $v_k = (s_k, t_k)$, $k = 1, 2, \ldots, N$. Since

$$v_k \otimes v_k = \begin{pmatrix} s_k^2 & s_k t_k \\ s_k t_k & t_k^2 \end{pmatrix},$$

the variance is

$$Q = \begin{pmatrix} a & b \\ b & c \end{pmatrix}, \tag{1.4.5}$$

where

$$a = \frac{1}{N}\sum_{k=1}^{N} s_k^2, \quad b = \frac{1}{N}\sum_{k=1}^{N} s_k t_k, \quad c = \frac{1}{N}\sum_{k=1}^{N} t_k^2. \tag{1.4.6}$$

From this, a is the variance of the first feature, c is the variance of the second feature, and b measures the interaction between the two features. In particular, both a and c are nonnegative.

We say a dataset in $\mathbf{R}^d$ is *standard* if each feature of the dataset is standard. When there are two features (x, y), this means the x samples are standard, and the y samples are standard. When this happens, $a = 1$, $c = 1$, and the variance has the form

$$Q' = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}, \tag{1.4.7}$$

for some scalar ρ. In Exercise **1.4.12**, it shown that Q' can only be a variance when $|\rho| \leq 1$.

Given a dataset with variance Q given by (1.4.5), let Σ be the diagonal matrix consisting of the standard deviations of the features,

$$\Sigma = \begin{pmatrix} \sqrt{a} & 0 \\ 0 & \sqrt{c} \end{pmatrix}.$$

Then, as in the scalar case, if $a > 0$ and $c > 0$, we may set

$$x'_k = \Sigma^{-1} v_k = \Sigma^{-1}(x_k - \mu), \qquad k = 1, 2, \ldots, N. \tag{1.4.8}$$

We claim $x'_1, x'_2, \ldots, x'_N$ is standard.

To see this, the key step is to use a tensor product identity (A.4.24) and write

$$x'_k \otimes x'_k = (\Sigma^{-1} v_k) \otimes (\Sigma^{-1} v_k) = \Sigma^{-1}(v_k \otimes v_k)\Sigma^{-1}, \qquad k = 1, 2, \ldots, N.$$

Summing over $k = 1, 2, \ldots, N$, we have

$$Q' = \frac{1}{N} \sum_{k=1}^{N} x'_k \otimes x'_k = \Sigma^{-1} \left(\frac{1}{N} \sum_{k=1}^{N} v_k \otimes v_k \right) \Sigma^{-1} = \Sigma^{-1} Q \Sigma^{-1}.$$

Multiplying out these three matrices, we conclude

$$Q' = \begin{pmatrix} 1/\sqrt{a} & 0 \\ 0 & 1/\sqrt{c} \end{pmatrix} \begin{pmatrix} a & b \\ b & c \end{pmatrix} \begin{pmatrix} 1/\sqrt{a} & 0 \\ 0 & 1/\sqrt{c} \end{pmatrix} = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix},$$

with

$$\rho = \frac{b}{\sqrt{ac}}. \tag{1.4.9}$$

This shows $x'_1, x'_2, \ldots, x'_N$ is standard.

For example,

$$Q = \begin{pmatrix} 9 & 2 \\ 2 & 4 \end{pmatrix} \quad \Longrightarrow \quad \rho = \frac{1}{3} \quad \Longrightarrow \quad Q' = \begin{pmatrix} 1 & 1/3 \\ 1/3 & 1 \end{pmatrix}.$$

The scalar ρ is the *correlation coefficient* ("rho") of the dataset, and the variance Q' of the standardized dataset $x'_1, x'_2, \ldots, x'_N$ is the *correlation matrix* of the original dataset $x_1, x_2, \ldots, x_N$.

When $\rho > 0$, we say the features are *positively correlated.* When $\rho < 0$, we say the features are *negatively correlated.*

Summarizing the above,

Variance and Correlation Matrices in Two Dimensions

- If Q in (1.4.5) is the variance of a dataset, then $a \geq 0$, $c \geq 0$, and $b^2 \leq ac$.
- If Q' in (1.4.7) is the variance of a standard dataset, then $|\rho| \leq 1$.
- If Q' is the correlation matrix of a dataset with variance Q, then $\rho = b/\sqrt{ac}$.

When the correlation coefficient is $\rho = \pm 1$, the dataset lies on a line passing through the mean. When $\rho = 1$, the line has slope 1, and when $\rho = -1$, the line has slope -1.



Here is a worked example. Suppose $N = 5$ and

$$x_1 = (1, 2), \quad x_2 = (3, 4), \quad x_3 = (5, 6), \quad x_4 = (7, 8), \quad x_5 = (9, 10). \tag{1.4.10}$$

Then $\mu = (5, 6)$ and

$$v_1 = x_1 - \mu = (-4, -4), \quad v_2 = x_2 - \mu = (-2, -2), \quad v_3 = x_3 - \mu = (0, 0),$$
$$v_4 = x_4 - \mu = (2, 2), \quad v_5 = x_5 - \mu = (4, 4).$$

Since

$$(\pm 4, \pm 4) \otimes (\pm 4, \pm 4) = \begin{pmatrix} 16 & 16 \\ 16 & 16 \end{pmatrix},$$
$$(\pm 2, \pm 2) \otimes (\pm 2, \pm 2) = \begin{pmatrix} 4 & 4 \\ 4 & 4 \end{pmatrix},$$
$$(0, 0) \otimes (0, 0) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix},$$

summing and dividing by N leads to the variance

$$Q = \begin{pmatrix} 8 & 8 \\ 8 & 8 \end{pmatrix}.$$

For this dataset,

$$\Sigma = \begin{pmatrix} 2\sqrt{2} & 0 \\ 0 & 2\sqrt{2} \end{pmatrix},$$

and

$$x'_1 = (-\sqrt{2}, -\sqrt{2}), \qquad x'_2 = (-1/\sqrt{2}, -1/\sqrt{2}), \qquad x'_3 = (0,0),$$
$$x'_4 = (1/\sqrt{2}, 1/\sqrt{2}), \qquad x'_5 = (\sqrt{2}, \sqrt{2})$$

is the standardized dataset.



Here is code from scratch for the variance of a dataset.

```
def tensor(u,v):
   return array([ [ a*b for b in v ] for a in u ])

N, d = 20, 2
# N x d array
dataset = samples((N,d))
mu = mean(dataset,axis = 0)

# center dataset
vectors = dataset - mu

Qtensor = mean([ tensor(v,v) for v in vectors ],axis = 0)
Qtensor
```

In `numpy`, variance is returned by

```
Qcov = cov(dataset.T,bias = True)
Qcov
```

This returns the same result as the previous code for Q. Here there is no need to compute the mean, this is taken care of automatically.

The variance matrix in (1.4.4) is the *biased* variance matrix. If instead the denominator is $N-1$, the matrix is the *unbiased* variance matrix.

For datasets with large N, it doesn't matter, since N and $N-1$ are almost equal. For simplicity, here we divide by N, and we only consider the biased variance matrix.

In practice, datasets are *standardized* before computing their variance. The variance of standard datasets — the correlation matrix — is the same whether one starts with bias or not.

In the `numpy` code above, the option `bias = True` indicates division by N, returning the biased variance. To return the unbiased variance and divide by $N-1$, change the option to `bias = False`, or remove it, since that is the default.

In `numpy`, the correlation matrix Q' is returned by

```
corrcoef(dataset.T)
```

Since $\text{trace}(v \otimes v) = |v|^2$, if Q is the variance matrix (1.4.4),

$$\text{trace}(Q) = \frac{1}{N}\sum_{k=1}^{N} |v_k|^2 = \frac{1}{N}\sum_{k=1}^{N} |x_k - \mu|^2 = MSD(\mu). \tag{1.4.11}$$

We call $\text{trace}(Q)$ the *total variance* or *explained variance* of the dataset. In Python,

```
Qcov.trace()
```

Let let $v_1, v_2, \ldots, v_N$ be a centered dataset in the plane. Let u be a unit vector (a vector of length one, $|u| = 1$). We wish to project this dataset onto the line through u, resulting in a scalar dataset.

According to Figure 1.13, when a vector b is projected onto the line through u, the length of the projected vector Pb equals $|b| \cos\theta$, where θ is the angle between the vectors b and u. Since $|u| = 1$, this length equals the dot product $b \cdot u$. Since Pb is a multiple of u,

$$Pb = (b \cdot u)u.$$

Repeating this with b replaced by each vector $v_1, v_2, \ldots, v_N$, we conclude: the *projected dataset* onto the line through u is

$$(v_1 \cdot u)u, (v_2 \cdot u)u, \ldots, (v_N \cdot u)u.$$

These vectors are all multiples of u, as they should be. The projected dataset is two-dimensional.

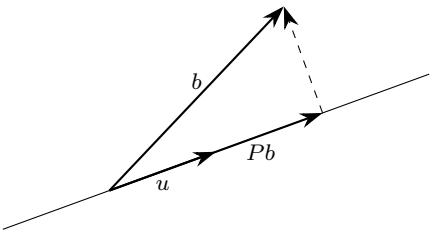


Fig. 1.13 Projecting a vector b onto the line through u.

Alternately, discarding u and retaining the scalar coefficients, we have the scalar dataset

$$v_1 \cdot u, v_2 \cdot u, \ldots, v_N \cdot u.$$

This is the *reduced dataset*. The reduced dataset is one-dimensional.

Since the vector u is fixed, the reduced dataset and the projected dataset contain the same information. The formulas for the projected and reduced datasets are correct only when u is a unit vector. For a non-unit vector u, replace u by $u/|u|$.

The reduced dataset is centered, since

$$\frac{v_1 \cdot u + v_2 \cdot u + \cdots + v_N \cdot u}{N} = \left(\frac{v_1 + v_2 + \cdots + v_N}{N}\right) \cdot u = 0 \cdot u = 0,$$

and the mean of the projected dataset is also 0.

The *variance of the reduced dataset*

$$q = \frac{1}{N}\sum_{k=1}^{N}(v_k \cdot u)^2$$

is a scalar that is positive or zero. According to (A.4.22), this equals

$$q = \frac{1}{N}\sum_{k=1}^{N} u \cdot (v_k \otimes v_k)u = u \cdot Qu.$$

Thus we conclude

Variance of Reduced Dataset

Let Q be the variance matrix of a dataset and let u be a unit vector. Then the variance of the dataset reduced onto the line through u equals $u \cdot Qu$.

A matrix Q is *positive* if it is symmetric and $u \cdot Qu > 0$ for any nonzero vector u. A matrix Q is *nonnegative* if it is symmetric and $u \cdot Qu \geq 0$ for any vector u.

A matrix Q is a *variance matrix* if Q is the variance matrix of some dataset. As a consequence of the above, since $q = u \cdot Qu$ is the variance of the reduced dataset, we have

Variance Matrix is Nonnegative

Every variance matrix is nonnegative: $u \cdot Qu$ is nonnegative for every vector u.

Later (§3.2), we see every nonnegative matrix is a variance matrix. Here is code for computing the variance of the projected dataset.

```
u = array([1/sqrt(2),1/sqrt(2)])

# project along unit vector u
q = dot(u,dot(Qcov,u))
q
```

Going back to the dataset (1.4.10), $x_k - \mu$, $k = 1, 2, 3, 4, 5$, are all multiples of $(1, 1)$. If we select $u = (1, -1)$, then $(x_k - \mu) \cdot u = 0$, so the variance Q satisfies $u \cdot Qu = 0$. This can also be seen by

$$Qu = 8((1, 1) \otimes (1, 1))u = 8(1, 1) \cdot u\,(1, 1) = 0.$$

This shows that the dataset lies on the line passing through μ and perpendicular to $(1, -1)$.



We describe the variance ellipses associated to a dataset. Let Q be a variance matrix and μ a point in $\mathbf{R}^2$. The contour of all points x satisfying

$$(x - \mu) \cdot Q(x - \mu) = k$$

is the *variance ellipse corresponding to level* k. When $k = 1$, the ellipse is the *unit variance ellipse*.

The contour of all points x satisfying

$$(x - \mu) \cdot Q^{-1}(x - \mu) = k$$

is the *inverse variance ellipse corresponding to level* k. When $k = 1$, the ellipse is the *unit inverse variance ellipse*.

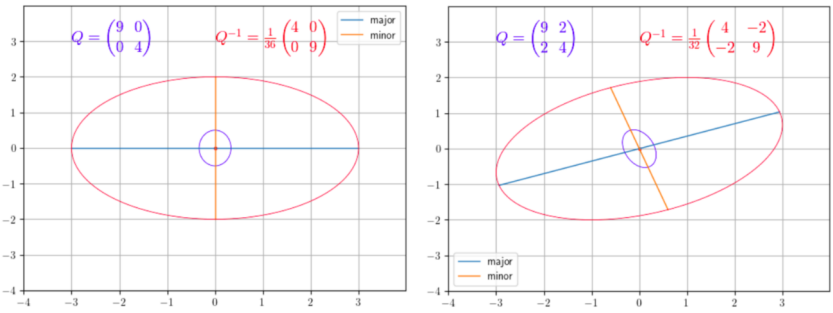


Fig. 1.14 Unit variance ellipses (blue) and unit inverse variance ellipses (red) with $\mu = 0$.

In two dimensions, a variance matrix has the form

$$Q = \begin{pmatrix} a & b \\ b & c \end{pmatrix}.$$

If $v = x - \mu$ has features (s, t), the variance ellipse equation centered at $\mu = 0$ is

$$v \cdot Qv = as^2 + 2bst + ct^2 = k.$$

The inverse variance ellipse centered at $\mu = 0$ is given by the same equation with Q replaced by Q^{-1}.

To use TEX to display the matrices in Figure 1.14, insert the function

```
rcParams['text.usetex'] = True
rcParams['text.latex.preamble'] = r'\usepackage{amsmath}'

def matrix_text(Q,mu,padding,color):
   a, b, c = Q[0,0],Q[0,1],Q[1,1]
   d,e = mu
   valign = e + 3*padding/4
   if color == 'blue': halign = d - padding/2; tex = "$Q="
   else: halign = d; tex = "$Q^{-1}="
   # r'...' means raw string
   tex += r'\begin{pmatrix}' + str(round(a,2)) + '&' +
    ↪ str(round(b,2))
   tex += r'\\' + str(round(b,2)) + '&' + str(round(c,2))
   tex += r'\end{pmatrix}$'
   return text(halign,valign,tex,fontsize = 15,color = color)
```

A minimal TEX installation is included in `matplotlib.pyplot`. To display matrices, this is not enough, and access to your laptop's TEX installation is needed. The `rcParams` lines enable this access. If TEX is installed on your laptop, uncomment `matrix_text` in `ellipse`.

The code for rendering the major and minor axes of the inverse variance ellipse uses (1.4.12) and (1.4.13),

```
def draw_major_minor_axes(Q,mu):
   a, b, c = Q[0,0],Q[0,1],Q[1,1]
   d, e = mu
   label = { 1:"major", -1:"minor" }
   for pm in [1,-1]:
      lamda = (a+c)/2 + pm * sqrt(b**2 + (a-c)**2/4)
      sigma = sqrt(lamda)
      lenv = sqrt(b**2 +(a-lamda)**2)
      lenw = sqrt(b**2 +(c-lamda)**2)
      if lenv: deltaX, deltaY = b/lenv, (a-lamda)/lenv
      elif lenw: deltaX, deltaY = (lamda-c)/lenw, b/lenw
      elif pm == 1:  deltaX, deltaY = 1, 0
      else:  deltaX, deltaY = 0, 1
      axesX = [d+sigma*deltaX,d-sigma*deltaX]
      axesY = [e-sigma*deltaY,e+sigma*deltaY]
```

```
    plot(axesX,axesY,linewidth = .5,label = label[pm])
    legend()
```

The code for rendering ellipses is

```
from matplotlib.pyplot import *
from numpy import *
from scipy.linalg import inv

def ellipse(Q,mu,padding = .5,levels = [1],render = "var"):
   scatter(*mu,c = "red",s = 5)
   a, b, c = Q[0,0],Q[0,1],Q[1,1]
   d,e = mu
   delta = .01
   x = arange(d-padding,d+padding,delta)
   y = arange(e-padding,e+padding,delta)
   x, y = meshgrid(x, y)
   if  render == "var" or render == "both":
      matrix_text(Q,mu,padding,'blue')
      eq = a*(x-d)**2 + 2*b*(x-d)*(y-e) + c*(y-e)**2
      contour(x,y,eq,levels = levels,colors = "blue",linewidths =
    ↪ .5)
   if render == "inv" or render == "both":
      draw_major_minor_axes(Q,mu)
      Q = inv(Q)
      matrix_text(Q,mu,padding,'red')
      A, B, C = Q[0,0],Q[0,1],Q[1,1]
      eq = A*(x-d)**2 + 2*B*(x-d)*(y-e) + C*(y-e)**2
      contour(x,y,eq,levels = levels,colors = "red",linewidths =
    ↪ .5)
```

With this code, `ellipse(Q,mu)` returns the unit variance ellipse in the unit square centered at μ. The codes for the functions `draw_major_minor_axes` and `matrix_text` are below.

Depending on whether `render` is `var`, `inv`, or `both`, the code renders the variance ellipse (blue), the inverse variance ellipse (red), or both. The code renders several ellipses, one for each level in the list `levels`. The default is `levels = [1]`, so the unit ellipse is returned. Also `padding` can be adjusted to enlarge the plot.

The code for Figure 1.14 is

```
mu = array([0,0])

Q = array([[9,0],[0,4]])
ellipse(Q,mu,padding = 4,render = "both")
grid()
show()
```

```
Q = array([[9,2],[2,4]])
ellipse(Q,mu,padding = 4,render = "both")
grid()
show()
```

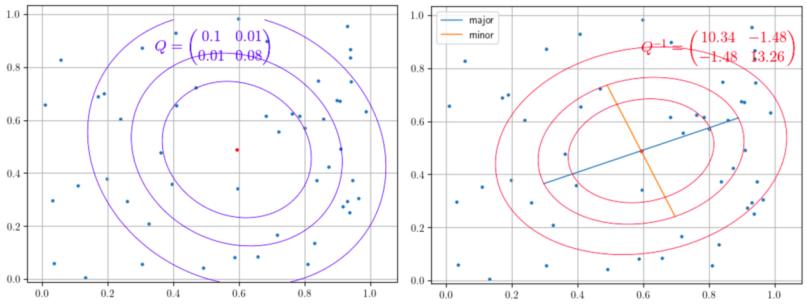


Fig. 1.15 Variance ellipses (blue) and inverse variance ellipses (red) for a dataset.

Figure 1.15 shows variance ellipses with `levels = [.005,.01,.02]`, and inverse variance ellipses with `levels = [.5,1,2]`, corresponding to a random dataset. The code for this is

```
from numpy.random import default_rng
samples = default_rng().random

N = 50
# N x d array
dataset = samples((N,2))
Q = cov(dataset.T,bias = True)
mu = mean(dataset,axis = 0)
print("mu = ",mu)
print("Q = ",Q)

scatter(*dataset.T,s = 5)
ellipse(Q,mu,render = "var",padding = .5,levels = [.005,.01,.02])
grid()
show()

scatter(*dataset.T,s = 5)
ellipse(Q,mu,render = "inv",padding = .5,levels = [.5,1,2])
grid()
show()
```

When Q is diagonal, the lengths of the major and minor axes of the unit inverse variance ellipse equal $2\sqrt{a}$ and $2\sqrt{c}$, and the lengths of the major and minor axes of the unit variance ellipse equal $2/\sqrt{a}$ and $2/\sqrt{c}$.

When the dataset is standard, the ellipses are particularly simple (Figure 1.16).

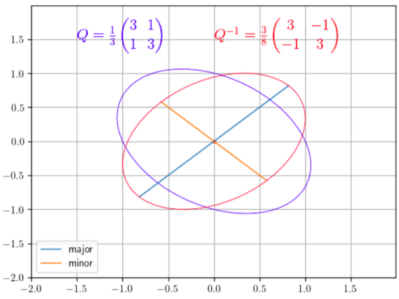


Fig. 1.16 Unit variance ellipse and unit inverse variance ellipse with standard Q.

We say a unit vector u is *best-aligned* or *best-fit* with the dataset if u maximizes the variance $v \cdot Qv$ over all unit vectors v,

$$u \cdot Qu = \max_{|v|=1} v \cdot Qv.$$

We calculate the best-aligned unit vector. When the dataset is standard, this is particularly straightforward.

The variance of a dataset projected onto a unit vector $v = (\cos\theta, \sin\theta)$ equals

$$v \cdot Qv = a\cos^2\theta + 2b\sin\theta\cos\theta + c\sin^2\theta.$$

When the dataset is standard, by the double-angle formula,

$$v \cdot Qv = \cos^2\theta + \sin^2\theta + 2\rho\sin\theta\cos\theta = 1 + \rho\sin(2\theta).$$

Since the sine function varies between $+1$ and -1, we conclude the projected variance varies between

$$1 - \rho \le v \cdot Qv \le 1 + \rho,$$

and

$$\theta = \frac{\pi}{4}, \quad v_+ = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right) \qquad \Longrightarrow \qquad v_+ \cdot Qv_+ = 1 + \rho,$$

$$\theta = \frac{3\pi}{4}, \quad v_- = \left(\frac{-1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right) \qquad \Longrightarrow \qquad v_- \cdot Qv_- = 1 - \rho.$$

Thus the best-aligned vector v_+ is at 45°, and the worst-aligned vector is at 135° (Figure 1.16).

Actually, the above is correct only if $\rho > 0$. When $\rho < 0$, it's the other way. The correct answer is

$$1 - |\rho| \le v \cdot Qv \le 1 + |\rho|,$$

and $v_\pm$ must be switched when $\rho < 0$. We study best-aligned vectors in $\mathbf{R}^d$ in §3.2.



With the variance given by (1.4.5), we always have $a \ge 0$ and $c \ge 0$. We say a dataset is *positively correlated* if $b > 0$, and *negatively correlated* if $b < 0$.

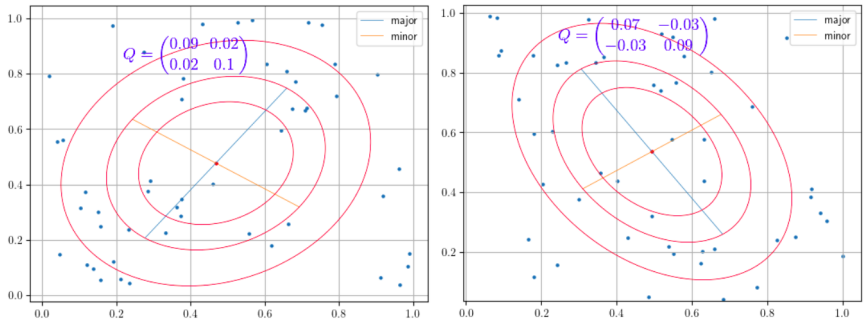


Fig. 1.17 Positively and negatively correlated datasets (inverse ellipses).

Here are two randomly generated datasets. The dataset on the left in Figure 1.17 is positively correlated. Its mean and variance are

$$\mu = (0.46819645, 0.47686339), \qquad Q = \begin{pmatrix} 0.08696118 & 0.0161062 \\ 0.0161062 & 0.09843542 \end{pmatrix}.$$

The dataset on the right in Figure 1.17 is negatively correlated. Its mean and variance are

$$\mu = (0.49349142, 0.53558378), \qquad Q = \begin{pmatrix} 0.06899527 & -0.02918281 \\ -0.02918281 & 0.09221775 \end{pmatrix}.$$



In general, for non-standard datasets, the projected variance $v \cdot Qv$ (v is a unit vector) varies between two extremes $\lambda_\pm$,

$$\lambda_- \le v \cdot Qv \le \lambda_+, \qquad |v| = 1.$$

where $\lambda_\pm$ are given by

$$\lambda_\pm = \frac{a+c}{2} \pm \sqrt{\left(\frac{a-c}{2}\right)^2 + b^2}. \tag{1.4.12}$$

When the dataset is standard, as we saw above, $\lambda_\pm = 1 \pm |\rho|$.

The major axis of the inverse variance ellipse $v \cdot Q^{-1}v = 1$ has length $2\sqrt{\lambda_+}$, and the minor axis has length $2\sqrt{\lambda_-}$. These are the *principal axes* of the dataset.

When the dataset is not standard, let

$$v_\pm = (-b, a - \lambda_\pm), \qquad \text{and} \qquad w_\pm = (\lambda_\pm - c, b), \tag{1.4.13}$$

If the inverse variance ellipse is not a circle, then Q is not a multiple of the identity, and either v_+ or w_+ is nonzero. If $v_+ \neq 0$, v_+ is the best-aligned vector. If $v_+ = 0$, w_+ is the best-aligned vector.

If the inverse variance ellipse is not a circle, then Q is not a multiple of the identity, and either v_- or w_- is nonzero. If $v_- \neq 0$, v_- is the worst-aligned vector. If $v_- = 0$, w_- is the worst-aligned vector.

If Q is a multiple of the identity, then any vector is best-aligned and worst-aligned. All this follows from solutions of homogeneous 2×2 systems (A.4.10). The formulas (1.4.13) are used in `draw_major_minor_axes(mu,Q)`.

The general $d \times d$ case is in §3.2. For the 2×2 case discussed here, see Exercise **3.2.3**.

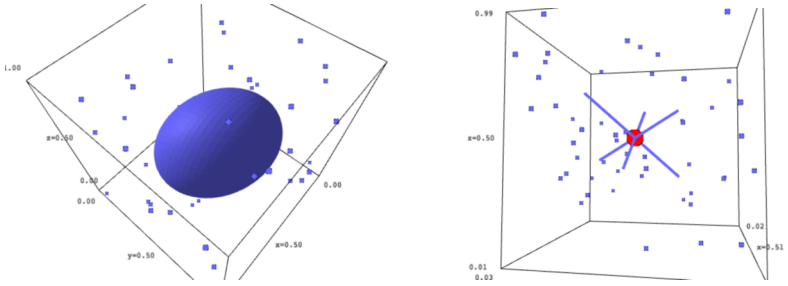


Fig. 1.18 Ellipsoid and axes in 3d.

In three dimensions, when $d = 3$, the ellipses are replaced by ellipsoids (Figure 1.18).

Exercises

Exercise 1.4.1 The dataset is

```
from numpy import *

d = 10
# 100 x 2 array
dataset = array([ array([i+j,j]) for i in range(d) for j in
    ↪ range(d) ])
```

Compute the mean and variance, and plot the dataset and the mean.

Exercise 1.4.2 Let the dataset be the petal lengths against the petal widths in the Iris dataset. Compute the mean and variance, and plot the dataset and the mean.

Exercise 1.4.3 Project the dataset in Exercise **1.4.1** onto the line through the vector $(1, 2)$. What is the projected dataset? What is the reduced dataset?

Exercise 1.4.4 Project the dataset in Exercise **1.4.2** onto the line through the vector $(1, 2)$. What is the projected dataset? What is the reduced dataset?

Exercise 1.4.5 Plot the variance ellipse and inverse variance ellipses of the dataset in Exercise **1.4.1**.

Exercise 1.4.6 Plot the variance ellipse and inverse variance ellipses of the dataset in Exercise **1.4.2**.

Exercise 1.4.7 Plot the dataset in Exercise **1.4.1** together with its mean and the line through the vector of best fit.

Exercise 1.4.8 Plot the dataset in Exercise **1.4.2** together with its mean and the line through the vector of best fit.

Exercise 1.4.9 Standardize the dataset in Exercise **1.4.1**. Plot the standardized dataset. What is the correlation matrix?

Exercise 1.4.10 Standardize the dataset in Exercise **1.4.2**. Plot the standardized dataset. What is the correlation matrix?

Exercise 1.4.11 Let $Q = \begin{pmatrix} a & b \\ b & a \end{pmatrix}$. Show Q is nonnegative when $a \geq |b|$. (Compute $v \cdot Qv$ with $v = (\cos\theta, \sin\theta)$ as in the text.)

Exercise 1.4.12 With Q as in (1.4.5), let $u = (x, 1)$, and let $q(x) = u \cdot Qu$ be the projected variance. Then $q(x) = ax^2 + 2bx + c$ is a nonnegative parabola. Compute the bottom of $q(x)$ to conclude $b^2 \leq ac$. In particular, for Q' as in (1.4.9), $|\rho| \leq 1$.

Exercise 1.4.13 Let ρ be a number satisfying $-1 \leq \rho \leq 1$. Compute the variance of the centered dataset in $\mathbf{R}^2$

$$x_1 = (1, \rho), \qquad x_2 = -x_1, \qquad x_3 = \left(0, \sqrt{1-\rho^2}\right), \qquad x_4 = -x_3.$$

1.5 High Dimensions

Although not used in later material, this section is here to boost intuition about high dimensions. Draw four disks inside a square, and a fifth disk in the center.

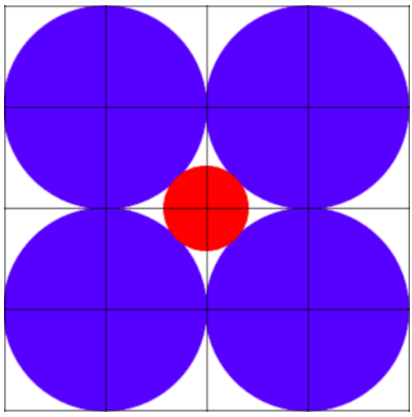


Fig. 1.19 Disks inside the square.

In Figure 1.19, the edge-length of the square is 4, and the radius of each blue disk is 1. Draw the diagonal of the square. Then the diagonal passes through two blue disks

Since the length of the diagonal of the square is $4\sqrt{2}$, and the diameters of the two blue disks add up 4, the portions of the diagonal outside the blue disks add up to $4\sqrt{2} - 4$. Hence the radius of the red disk is

$$\frac{1}{4}(4\sqrt{2} - 4) = \sqrt{2} - 1.$$

In three dimensions, draw eight balls inside a cube, as in Figure 1.20, and one ball in the center. Since the edge-length of the cube is 4, the radius of each blue ball is 1. Since the length of the diagonal of the cube is $4\sqrt{3}$, the radius of the red ball is

$$\frac{1}{4}(4\sqrt{3} - 4) = \sqrt{3} - 1.$$

.

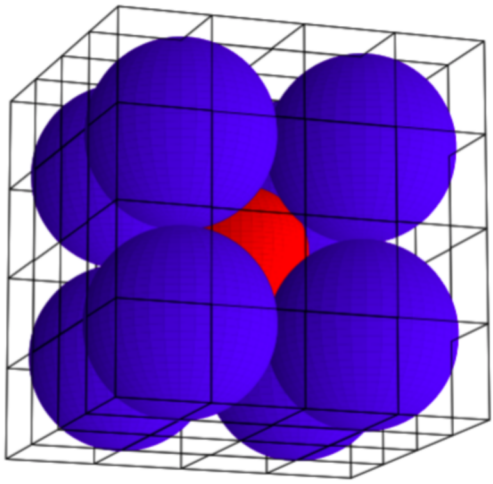


Fig. 1.20 Balls inside the cube.

Now we repeat in d dimensions. Here the edge-length of the cube remains 4, the radius of each blue ball remains 1, and there are 2^d blue balls. Since the length of the diagonal of the cube is $4\sqrt{d}$, the same calculation results in the radius of the red ball equal to $r = \sqrt{d} - 1$.



In two dimensions, when a region is scaled by a factor t, its area increases by the factor t^2. In three dimensions, when a region is scaled by a factor t, its volume increases by the factor t^3. The general result is

Scaling Principle: Dependence on Dimension

In d dimensions, when a region is scaled by a factor t, its volume scales by the factor t^d.

The radius of the red ball is $r = \sqrt{d} - 1$. By the scaling principle, in d dimensions, the volume of the red ball equals r^d times the volume of the blue ball. We conclude the following:

- Since $r = \sqrt{d} - 1 = 1$ exactly when $d = 4$, we have: *In four dimensions, the red ball and the blue balls are the same size.*
- Since there are 2^d blue balls, the ratio of the volume of the red ball over the total volume of all the blue balls is $r^d/2^d$.

- Since $r^d = 2^d$ exactly when $r = 2$, and since $r = \sqrt{d}-1 = 2$ exactly when $d = 9$, we have: *In nine dimensions, the volume of the red ball equals the sum total of the volumes of all blue balls.*
- Since $r = \sqrt{d} - 1 > 2$ exactly when $d > 9$, we have: *In ten or more dimensions, the red ball sticks out of the cube.*
- Since the length of the cube's diagonal is $4\sqrt{d}$, for any dimension d, the diameter of the red ball $2r = 2(\sqrt{d}-1)$ is less than $1/2$ the length of the diagonal. *As the dimension grows without bound, the proportion of the cube's diagonal covered by the red ball converges to* $1/2$.



The following code returns Figure 1.19.

```
from matplotlib.pyplot import *
from matplotlib.patches import Circle, Rectangle
from numpy import *
from itertools import product

# initialize figure
ax = axes()

square  = Rectangle((0,0), 4, 4,color = 'lightblue')
ax.add_patch(square)

xcent = ycent = [1,3]
# blue disks
for center in product(xcent,ycent):
   circle = Circle(center, radius = 1, color = 'blue')
   ax.add_patch(circle)

# red disk
circle  = Circle((2, 2), radius = sqrt(2)-1, color = 'red')
ax.add_patch(circle)

ax.set_xticks([0,1,2,3,4])
ax.set_yticks([0,1,2,3,4])

ax.axis('equal')
ax.grid(lw=.5,color="black")
show()
```

The code for Figure 1.20 is as follows.

```
%matplotlib ipympl
from  matplotlib.pyplot import *
from numpy import *
from itertools import product
```

```
# build sphere mesh
N = 40
theta = linspace(0,2*pi,N)
phi = linspace(0,pi,N)
theta,phi = meshgrid(theta,phi)

# spherical coordinates theta, phi
x = cos(theta)*sin(phi)
y = sin(theta)*sin(phi)
z = cos(phi)

# initialize figure
ax = axes(projection = "3d")

# render ball
def ball(a,b,c,r,color):
   return ax.plot_surface(a + r*x,b + r*y, c + r*z,color = color)

xcent = ycent = zcent = [1,3]
# blue balls
for center in product(xcent,ycent,zcent): ball(*center,1,"blue")

# red ball
ball(2,2,2,sqrt(3)-1,"red")

# cube grid
cube = ones((4,4,4),dtype = bool)
ax.voxels(cube, edgecolors = 'black',lw = .5,alpha = 0)

ax.set_aspect("equal")
ax.set_axis_off()
show()
```

If `theta` and `phi` have shapes `(m,)` and `(n,)` then

```
theta,phi = meshgrid(theta,phi)
```

returns arrays `theta` and `phi` having shapes `(m,n)`, with

`theta[i,j] = theta[i]`, `phi[i,j] = phi[j]`.

Here this is used to build a 2d mesh of 3d points

```
(x[i,j],y[i,j],z[i,j])
```

lying on a sphere. The cube grid is rendered using a *voxel grid*. Voxels are the 3d counterparts of 2d pixels.

In `Jupyter`, a *magic command* starts with a `%`. A magic command is sent to `Jupyter`, not to Python. The magic command `%matplotlib ipympl` allows for rotating the figure.



Another phenomenon that happens in high dimensions, discussed in §6.1, is that the angle between two randomly chosen vectors in a high-dimensional space is not arbitrary, it is pre-determined. This is a consequence of the law of large numbers.

Scaling and dimensionality work together in *suspensions*. (Figure 1.21).

Let $[a, b]$ be an interval and let V be a point not in the interval. To *suspend the interval from* V, draw line segments between V and all points in the interval. You end up with a triangle with vertex V. Therefore the suspension of an interval is a triangle. Here the dimension of the interval is one, and the dimension of the triangle is two.

Let D be a disk and let V be a point not in the disk. To *suspend the disk from* V, draw line segments between V and all points in the disk. You end up with a cone with vertex V. Therefore the suspension of a disk is a cone. Here the dimension of the disk is two, and the dimension of the cone is three.

In general, the suspension $\hat{G}$ of G is obtaining by drawing line segments from a point V not in G to every point x in G,

$$\hat{G} = \{(tx, 1-t) : 0 \le t \le 1, x \text{ in } G\}.$$

When $t = 1$, the suspension's base is the original region G, and when $t = 0$, we have the vertex at the top. For each $0 < t < 1$, the cross-section at level t of the suspension is tG, which is G scaled by the factor t.

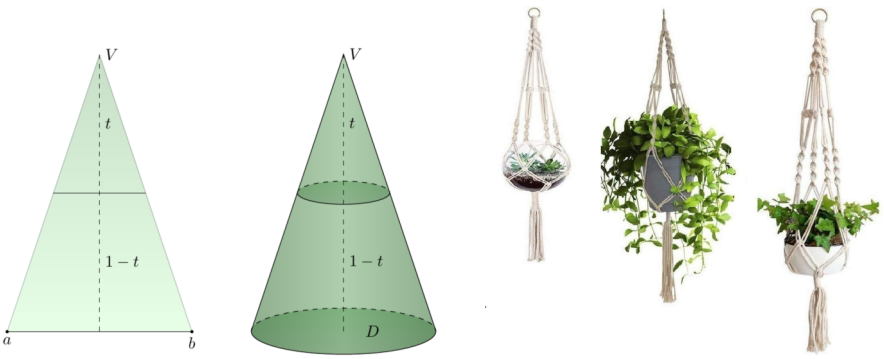


Fig. 1.21 Suspensions of interval $[a, b]$ and disk D.

We assume the point V is in a dimension orthogonal to the dimensions of G. Then the dimension of $\hat{G}$ is one more than the dimension of G: If G is d-dimensional, then $\hat{G}$ is $(d+1)$-dimensional, and the cross-sections at distinct levels do not intersect.

The volume of $\hat{G}$ is obtained by integrating over cross-sections,

$$\mathrm{Vol}(\hat{G}) = \int_0^1 \mathrm{Vol}(tG)\, dt.$$

By the scaling principle and (A.6.3),

$$\mathrm{Vol}(\hat{G}) = \int_0^1 t^d \mathrm{Vol}(G)\, dt = \mathrm{Vol}(G)\, \frac{t^{d+1}}{d+1}\Big|_{t=0}^{t=1} = \frac{\mathrm{Vol}(G)}{d+1}.$$

Thus

$$\mathrm{Vol}(\hat{G}) = \frac{\mathrm{Vol}(G)}{d+1}.$$

Exercises

Exercise 1.5.1 Why is the diagonal length of the square $4\sqrt{2}$?

Exercise 1.5.2 Why is the diagonal length of the cube $4\sqrt{3}$?

Exercise 1.5.3 Why does dividing by 4 yield the red disk radius and the red ball radius?

Exercise 1.5.4 Suspend the unit circle $G : x^2 + y^2 = 1$ from its center. What is the suspension $\hat{G}$? Conclude area(unit disk) = length(unit circle)/2.

Exercise 1.5.5 Suspend the unit sphere $G : x^2+y^2+z^2 = 1$ from its center. What is the suspension $\hat{G}$? Conclude vol(unit ball) = area(unit sphere)/3.

Chapter 2
Linear Geometry

In the present chapter, we study linear geometry in any dimension d. Readers with no prior exposure to linear geometry may wish to start with geometry in two dimensions §A.4.

The material in this chapter is usually referred to as *Linear Algebra*. We prefer the term *Linear Geometry*, to emphasize that the material is, like much of data science, geometric.

Even though parts of this chapter are heavy-going, all included material is necessary for later chapters. In particular, the derivations of chi-squared distribution (§5.5) and the chi-squared tests (§6.4) are clarified by the appropriate use of vectors and matrices.

2.1 Vectors and Matrices

A *vector* is a list of scalars

$$v = (t_1, t_2, \ldots, t_d).$$

The scalars are the *components* or the *features* of v. If there are d features, we say the *dimension* of v is d. We call v a d-dimensional vector.

A *point* x is also a list of scalars, $x = (t_1, t_2, \ldots, t_d)$. The relation between points x and vectors v is discussed in §1.3. The set of all d-dimensional vectors or points is d-dimensional space $\mathbf{R}^d$.

In Python, we use `numpy` or `sympy` for vectors and matrices. In Python, if `L` is a list, then `numpy.array(L)` or `sympy.Matrix(L)` return a vector or matrix.

```
from numpy import *
set_printoptions(legacy = "1.25")
```

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

O. Hijab, *Math for Data Science*, https://doi.org/10.1007/978-3-031-89707-8_2

```
v = array([1,2,3])
print(v.shape)

from sympy import *

v = Matrix([1,2,3])
print(v.shape)
```

The first `v.shape` returns `(3,)`, and the second `v.shape` returns `(3,1)`. In either case, v is a 3-dimensional vector.

Vectors are added and scaled component by component: With

$$v = (t_1, t_2, \dots) \qquad \text{and} \qquad v = (t'_1, t'_2, \dots),$$

we have

$$v + v' = (t_1 + t'_1, t_2 + t'_2, \dots), \qquad \text{and} \qquad sv = (st_1, st_2, \dots).$$

Addition $v + v'$ only works when v and v' have the same shape.

The *zero vector* is the vector $0 = (0, 0, 0, \dots)$. The zero vector is the only vector satisfying $0 + v = v = v + 0$ for every vector v. Even though the zero scalar and the zero vector are distinct objects, we use 0 to denote both. A vector v is *nonzero* if v is not the zero vector.

In $\mathbf{R}^4$, the vectors

$$e_1 = (1, 0, 0, 0), \qquad e_2 = (0, 1, 0, 0), \qquad e_3 = (0, 0, 1, 0), \qquad e_4 = (0, 0, 0, 1)$$

together are the *standard basis*. Similarly, in $\mathbf{R}^d$, we have the *standard basis* e_1, e_2, …, e_d.



A *matrix* is a listing arranged in *a rectangle of rows and columns*. Specifically, an $d \times N$ matrix A has d rows and N columns,

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \dots & \dots & & \dots \\ a_{d1} & a_{d2} & \dots & a_{dN} \end{pmatrix}.$$

In Python, if `L` is a list of lists, then both `array(L)` and `Matrix(L)` return a matrix. The code

```
# numpy vectors
u = array([1,2,3,4,5])
```

```
v = array([6,7,8,9,10])
w = array([11,12,13,14,15])

A = column_stack([u,v,w])
print(A.shape)
```

returns `(5,3)`, so A is a 5×3 matrix,

$$A = \begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix}.$$

The *transpose* of a matrix A is the matrix $B = A^t$ resulting from turning A on its side, so

$$B = A^t = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix}.$$

In `numpy`, the default is to arrange vectors as rows, so

```
B = array([u,v,w])
B == A.T
```

returns `True`. The transpose interchanges rows and columns: the rows of A^t are the columns of A. In both `numpy` or `sympy`, the transpose of `A` is `A.T`.

In `sympy`,

```
from sympy import *

# column vectors
u = Matrix([1,2,3,4,5])
v = Matrix([6,7,8,9,10])
w = Matrix([11,12,13,14,15])

A = Matrix.hstack(u,v,w)
A.shape
```

also returns `(5,3)`.

A vector v may be written as a $1 \times N$ matrix

$$v = \begin{pmatrix} t_1 & t_2 & \dots & t_N \end{pmatrix}.$$

In this case, we call v a *row vector*.

A vector v may be written as a $d \times 1$ matrix

$$v = \begin{pmatrix} t_1 \\ t_2 \\ \dots \\ t_d \end{pmatrix}.$$

In this case, we call v a *column vector*.



We will be considering matrices with different properties, and we use the following notation

- A, B: any matrix
- U, V: orthonormal rows or orthonormal columns
- Q: symmetric matrix
- P: projection or permutation matrix



Vectors $v_1, v_2, \dots, v_N$ in $\mathbf{R}^d$ may be horizontally stacked as columns of a $d \times N$ matrix,

$$A = \begin{pmatrix} v_1 & v_2 & \dots & v_N \end{pmatrix}.$$

Similarly, vectors $v_1, v_2, \dots, v_N$ in $\mathbf{R}^d$ may be vertically stacked as rows of an $N \times d$ matrix,

$$A = \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ v_N \end{pmatrix}.$$

By default, `sympy` creates column vectors. Because of this, it is easiest to build matrices as columns,

```
# 5x3 matrix
A = Matrix.hstack(u,v,w)

# column vector
b = Matrix([1,1,1,1,1])

# 5x4 matrix
M = Matrix.hstack(A,b)
display(A, b, M)
```

(`display`, included in `Jupyter`, returns enhanced output.)

In general, for any `sympy` matrix `A`, column vectors can be `hstack`ed and row vectors can be `vstack`ed. For any matrix `A`, the code

```
from sympy import *

A == Matrix.hstack(*[ A.col(j) for j in range(A.cols) ])
```

returns `True`. Note we use the unpacking operator `*` to unpack the list, before applying `hstack`.

In `numpy`, there is `column_stack` and `row_stack`, so the code

```
u = array([1,2,3,4,5])
v = array([6,7,8,9,10])
w = array([11,12,13,14,15])

A = column_stack([u,v,w])
print(A == vstack([ row for row in A ]))
print(A == column_stack([ col for col in A.T ]))
```

returns `True`. In `numpy`, the input is a list, there is no unpacking.



In `numpy`, a matrix A is a list of rows, so

```
print(A == array([ row for row in A ]))
print(A.T == array([ col for col in A.T ]))
```

both return `True`. Here `col` is a *row* of A^t, hence is a column of A.

In `numpy`, the number of rows is `len(A)`, and the number of columns is `len(A.T)`. To access row i, use `A[i]`. To access column j, access row j of the transpose, `A.T[j]`. To access the j-th entry in row i, use `A[i,j]` or `A[i][j]`.

In `sympy`, the number of rows in a matrix A is `A.rows`, and the number of columns is `A.cols`, so `A.shape == (A.rows,A.cols)`.

To access row i, use `A.row(i)`. Similarly, to access column j, use `A.col(j)`. So,

```
u = Matrix([1,2,3,4,5])
v = Matrix([6,7,8,9,10])
w = Matrix([11,12,13,14,15])
A = Matrix.hstack(u,v,w)

print(A == Matrix([ A.row(i) for i in range(A.rows) ]))
print(A.T == Matrix([ A.col(j).T for j in range(A.cols) ]))
```

both return `True`.

A matrix is *square* if the number of rows equals the number of columns, $N = d$. A matrix is *diagonal* if it looks like one of these

$$\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & d \end{pmatrix}, \quad \text{or} \quad \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \end{pmatrix}, \quad \text{or} \quad \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \\ 0 & 0 & 0 \end{pmatrix},$$

where some of the numbers on the diagonal a, b, c, d may be zero.



Matrices are added and scaled as follows. With

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \dots & \dots & & \dots \\ a_{d1} & a_{d2} & \dots & a_{dN} \end{pmatrix} \quad \text{and} \quad A' = \begin{pmatrix} a'_{11} & a'_{12} & \dots & a'_{1N} \\ a'_{21} & a'_{22} & \dots & a'_{2N} \\ \dots & \dots & & \dots \\ a'_{d1} & a'_{d2} & \dots & a'_{dN} \end{pmatrix},$$

we have *matrix addition*

$$A + A' = \begin{pmatrix} a_{11} + a'_{11} & a_{12} + a'_{12} & \dots & a_{1n} + a'_{1N} \\ a_{21} + a'_{21} & a_{22} + a'_{22} & \dots & a_{2n} + a'_{2N} \\ \dots & \dots & & \dots \\ a_{d1} + a'_{d1} & a_{d2} + a'_{d2} & \dots & a_{dN} + a'_{dN} \end{pmatrix}$$

and *matrix scaling*

$$tA = \begin{pmatrix} ta_{11} & ta_{12} & \dots & ta_{1N} \\ ta_{21} & ta_{22} & \dots & ta_{2N} \\ \dots & \dots & & \dots \\ ta_{d1} & ta_{d2} & \dots & ta_{dN} \end{pmatrix}.$$

Matrices may be added only if they have the same shape.

In Python, matrix scaling and matrix addition are `t*A` and `A + B`. The code

```
from sympy import *

A = zeros(2,3)
B = ones(2,2)
C = Matrix([[1,2],[3,4]])
D = B + C
E = 5 * C
F = eye(4)
display(A, B, C, D, E, F)
```

and the code

```
from numpy import *

A = zeros((2,3))
```

```
B = ones((2,2))
C = array([[1,2],[3,4]])
D = B + C
E = 5 * C
F = eye(4)
A, B, C, D, E, F
```

both return

$$\begin{pmatrix}0&0&0\\0&0&0\end{pmatrix}, \begin{pmatrix}1&1\\1&1\end{pmatrix}, \begin{pmatrix}1&2\\3&4\end{pmatrix}, \begin{pmatrix}2&3\\4&5\end{pmatrix}, \begin{pmatrix}5&10\\15&20\end{pmatrix}, \begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}.$$

Diagonal matrices are constructed using `diag`. The code

```
from sympy import *

A = diag(1,2,3,4)
B = diag(-1, ones(2, 2), Matrix([5, 7, 5]))
display(A, B)
```

```
from numpy import *
from scipy.linalg import block_diag

A = diag([1,2,3,4])
B = block_diag(-1, ones((2, 2)), array([[5], [7], [5]]))
A, B
```

returns

$$\begin{pmatrix}1&0&0&0\\0&2&0&0\\0&0&3&0\\0&0&0&4\end{pmatrix}, \begin{pmatrix}-1&0&0&0\\0&1&1&0\\0&1&1&0\\0&0&0&5\\0&0&0&7\\0&0&0&5\end{pmatrix}.$$

In Data Science, `numpy` is the default, but `sympy` is easier to start with, and comparing the two heightens understanding of both..

Exercises

Exercise 2.1.1 A vector is *one-hot encoded* if all features are zero, except for one feature which is one. For example, in $\mathbf{R}^3$ there are three one-hot encoded vectors

$$(1,0,0), \qquad (0,1,0), \qquad (0,0,1).$$

A matrix is a *permutation matrix* if it is square and all rows and all columns are distinct and one-hot encoded. How many 3×3 permutation matrices are there? What about $d \times d$? For more on permutations, see §A.1.

2.2 Products

Let t be a scalar, u, v, w be vectors, and let A, B be matrices. We already know how to compute tu, tv, and tA, tB. In this section, we study the dot product $u \cdot v$, the matrix-vector product Av, the matrix-matrix product AB, and the tensor product $u \otimes v$. In the appendix §A.4, this is worked out for 2×2 matrices and vectors in $\mathbf{R}^2$.

Suppose u, v are vectors in $\mathbf{R}^d$. Then their *dot product* $u \cdot v$ is the scalar obtained by multiplying corresponding features and then summing the products. In other words, if $u = (s_1, s_2, \ldots, s_d)$ and $v = (t_1, t_2, \ldots, t_d)$, then

$$u \cdot v = s_1 t_1 + s_2 t_2 + \cdots + s_d t_d. \tag{2.2.1}$$

It's best to think of this as "row-times-column" multiplication,

$$u \cdot v = \begin{pmatrix} s_1 & s_2 & s_3 \end{pmatrix} \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} = s_1 t_1 + s_2 t_2 + s_3 t_3.$$

As in §A.4, it's always *rows on the left, and columns on the right.*

In Python, the dot product is

```
from numpy import *
set_printoptions(legacy = "1.25")

u = array([1,2,3])
v = array([4, 5, 6])

print(dot(u,v), 1*4 + 2*5 + 3*6)

from sympy import *

u = Matrix([1,2,3])
v = Matrix([4, 5, 6])

print(u.T * v, 1*4 + 2*5 + 3*6)
```

For clarity, sometimes we write `(u.T)*v`; the parentheses don't change anything. In `sympy`, we take the transpose, since vectors are by default column

vectors, and it's always *row* × *column*. In `numpy`, the code returns a scalar; in `sympy`, a 1×1 matrix.



As in two dimensions, the *length* or *norm* or *magnitude* of a vector $v = (t_1, t_2, \ldots, t_d)$ is the square root of the dot product $v \cdot v$,

$$|v| = \sqrt{v \cdot v} = \sqrt{t_1^2 + t_2^2 + \cdots + t_d^2}.$$

In Python, the length of a vector `v` is

```
# length of v

from sympy import *
v = Matrix([4, 5, 6])
sqrt((v.T * v)[0])

from numpy import *
v = array([4, 5, 6])
print(sqrt(dot(v,v)))
```

A vector is a *unit vector* if its length equals 1. When $|v| = 0$, all the features of v equal zero. It follows the zero vector is the only vector with zero length. All nonzero vectors have positive length.

Let v be any nonzero vector. By dividing v by its length $|v|$, we obtain a unit vector $u = v/|v|$.



As in §A.4, the dot product may be expressed geometrically.

Dot Product Identity

The dot product $u \cdot v$ (2.2.1) satisfies

$$u \cdot v = |u|\,|v| \cos\theta, \tag{2.2.2}$$

where θ is the angle between u and v.

In two dimensions, this was equation (A.4.5) in §A.4. Since any two vectors lie in a two-dimensional plane, this remains true in any dimension. More exactly, we take (2.2.2) as the definition of $\cos\theta$.

Based on this, we can compute the angle θ,

$$\cos\theta = \frac{u \cdot v}{\sqrt{|u|\,|v|}} = \frac{u \cdot v}{\sqrt{(u \cdot u)(v \cdot v)}}.$$

Since $|\cos\theta| \le 1$, we have the

Cauchy-Schwarz Inequality

The dot product of two vectors is absolutely less or equal to the product of their lengths,

$$|u \cdot v| \le |u|\,|v| \qquad \text{or} \qquad (u \cdot v)^2 \le (u \cdot u)(v \cdot v). \tag{2.2.3}$$

Strictly speaking, the Cauchy-Schwarz inequality must be verified independently of the geometric definition, because it is used to *define* $\cos\theta$. But this is easy: The parabola

$$p(t) = |u + tv|^2 = (u + tv) \cdot (u + tv) = |u|^2 + 2tu \cdot v + t^2|v|^2 = a + 2tb + ct^2$$

is nonnegative, hence its bottom is nonnegative. By completing the square, the bottom is $a - b^2/c$. Since the bottom is nonnegative, we conclude $b^2 \le ac$, which is (2.2.3).



Vectors u and v are said to be *perpendicular* or *orthogonal* if $u \cdot v = 0$. In this case we often write $u \perp v$. A collection of vectors is *orthogonal* if any pair of vectors in the collection are orthogonal.

With this understood, the zero vector is orthogonal to every vector: $0 \cdot v = 0$ for every v. The converse is true as well: If $u \cdot v = 0$ for every v, then, by choosing $v = u$, $u \cdot u = 0$, which implies $u = 0$.

Vectors $v_1, \ldots, v_N$ are said to be *orthonormal* if they are both unit vectors and orthogonal. Orthogonal nonzero vectors can be made orthonormal by dividing each vector by its length.



An important application of the Cauchy-Schwarz inequality is the *triangle inequality*

$$|a + b| \le |a| + |b|. \tag{2.2.4}$$

To see this, let u be any unit vector. Then

$$(a + b) \cdot u = a \cdot u + b \cdot u \le |a||u| + |b||u| = |a| + |b|.$$

If we select $u = (a + b)/|a + b|$, then u is a unit vector and

$$(a + b) \cdot u = (a + b) \cdot \frac{(a + b)}{|a + b|} = \frac{|a + b|^2}{|a + b|} = |a + b|,$$

hence

$$|a+b| = (a+b)\cdot u \leq |a| + |b|.$$



Suppose v is a vector and A is a matrix. If the rows of A have the same dimension as that of v, we can take the dot product of each row of A with v, obtaining the *matrix-vector product* Av: *Av is the vector whose features are the dot products of the rows of A with v.*

In other words,

```
u = array([1,2,3,4,5])
v = array([6,7,8,9,10])
w = array([11,12,13,14,15])

A = column_stack([u,v,w])

r, s, t = 1, 2, 3
x = array([r,s,t])

dot(A,x) == array([ dot(row,x) for row in A ])
```

both return `True`.

If u and v are vectors, we can think of u as a row vector, or a matrix consisting of a single row. With this interpretation, the matrix-vector product uv equals the dot product $u \cdot v$.

If u and v are vectors, we can think of u as a column vector, or a matrix consisting of a single column. With this interpretation, u^t is a single row, and the matrix-vector product $u^t v$ equals the dot product $u \cdot v$.



Let A and B be two matrices. If the rows of A are $a_1, a_2, \ldots, a_N$, and the columns of B are $b_1, b_2, \ldots, b_d$,

$$A = \begin{pmatrix} a_1 \\ a_2 \\ \cdots \\ a_N \end{pmatrix}, \qquad B = \begin{pmatrix} b_1 \; b_2 \; \ldots \; b_d \end{pmatrix},$$

then the *matrix-matrix product* AB is the $N \times d$ matrix whose (i,j)-th entry is the dot product

$$(AB)_{ij} = a_i \cdot b_j.$$

Note the dot product is defined only if the rows of A and the columns of B have the same dimension.

Alternatively, using the definition of matrix-vector product, AB is the matrix with columns $Ab_1, Ab_2, \ldots, Ab_d$,

$$AB = \left(Ab_1\ Ab_2\ \ldots\ Ab_d\right).$$

Unpacking the dot product, and using summation notation, if $A = (a_{ij})$ and $B = (b_{ij})$, then

$$(AB)_{ij} = \sum_{k=1}^{d} a_{ik} b_{kj}, \qquad 1 \le i \le N, 1 \le j \le d. \tag{2.2.5}$$

In Python,

```
B = A.T

C = array([ [ dot(row,col) for col in B.T ] for row in A ])
dot(A,B) == C
```

returns `True`, and, with

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}, B = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix},$$

the above code returns

$$AB = \begin{pmatrix} 70 & 80 & 90 \\ 158 & 184 & 210 \end{pmatrix}.$$



The *trace* of a square matrix

$$A = \begin{pmatrix} a & b & c \\ b & d & e \\ c & e & f \end{pmatrix}$$

is the sum of its diagonal elements,

$$\text{trace}(A) = \text{trace} \begin{pmatrix} a & b & c \\ b & d & e \\ c & e & f \end{pmatrix} = a + d + f.$$

In general, the trace of a $d \times d$ matrix is

$$\text{trace}(A) = \sum_{i=1}^{d} a_{ii}. \tag{2.2.6}$$

Even though in general $AB \neq BA$, it is always true that

$$\text{trace}(AB) = \text{trace}(BA), \tag{2.2.7}$$

This can be verified by switching the i and the k in the sums

$$\text{trace}(AB) = \sum_{i=1}^{d} (AB)_{ii} = \sum_{i=1}^{d} \sum_{k=1}^{d} a_{ik} b_{kj}.$$



A matrix Q is *symmetric* if $Q = Q^t$. For any matrix A, $Q = AA^t$ and $Q = A^t A$ are symmetric.

A symmetric matrix Q satisfying $v \cdot Qv \geq 0$ for every vector v is *nonnegative*. When Q is nonnegative, we write $Q \geq 0$. A symmetric matrix Q satisfying $v \cdot Qv > 0$ for every nonzero vector v is *positive*. When Q is positive, we write $Q > 0$. Since any vector may be rescaled into a unit vector, the vectors v in these definitions may be assumed to be unit vectors.

For any $d \times N$ matrix A, $A^t A$ is a symmetric $N \times N$ matrix, and AA^t is a symmetric $d \times d$ matrix.

As we saw in §1.4, the variance matrix of a dataset is nonnegative. In fact, when a dataset in $\mathbf{R}^d$ fills up all d dimensions, the variance matrix is positive (see §2.5).



As in §A.4, we have

Dot Product Transpose Identity

For any vectors u, v, and matrices A, we have

$$(Au) \cdot v = u \cdot (A^t v) \qquad \text{and} \qquad (A^t u) \cdot v = u \cdot Av, \tag{2.2.8}$$

whenever the shapes of u, v, A are compatible.

The is established by the same proof as in §A.4.

In Python,

```
dot(dot(A,x),v) == dot(x,dot(A.T,v))
```

returns `True`.

As a consequence, we have, as in §A.4,

$$(AB)^t = B^t A^t.$$

In Python,

```
dot(A,B).T == dot(B.T,A.T)
```

both return `True`.



Let A be a matrix. We compute useful expressions for AA^t and A^tA.

Assume the columns of A are $v_1, v_2, \ldots, v_N$, so A is $d \times N$. Since the transpose interchanges rows and columns, $v_1, v_2, \ldots, v_N$ are the rows of A^t. Since matrix-matrix multiplication is *row* × *column*, we have

$$A^tA = \begin{pmatrix} v_1 \cdot v_1 & v_1 \cdot v_2 & \ldots & v_1 \cdot v_N \\ v_2 \cdot v_1 & v_2 \cdot v_2 & \ldots & v_2 \cdot v_N \\ \ldots & \ldots & \ldots & \ldots \\ v_N \cdot v_1 & v_N \cdot v_2 & \ldots & v_N \cdot v_N \end{pmatrix}. \tag{2.2.9}$$

As a consequence,[1]

Orthonormal Rows and Columns

Let U be a matrix.

- U has orthonormal columns iff $U^tU = I$.
- U has orthonormal rows iff $UU^t = I$.

The second statement follows from the first by substituting U^t for U.



To compute AA^t, we bring in the tensor product (see §A.4 for vectors in the plane). If u and v are vectors, the *tensor product* $u \otimes v$ is the matrix

$$(u \otimes v)_{ij} = u_i v_j.$$

If u is in $\mathbf{R}^N$ and v is in $\mathbf{R}^d$, then $u \otimes v$ is an $N \times d$ matrix.

From the definition,

$$(u \otimes v)^t = v \otimes u$$

and

$$\text{trace}(u \otimes v) = u \cdot v, \tag{2.2.10}$$

[1] *Iff* is short for *if and only if*.

the latter valid when $u \otimes v$ is square.

The basic tensor identities in §A.4, and the proofs presented there, remain valid in any dimension. Here they are again.

Tensor Product Identities

The following identities hold, as long as the shapes of the matrices and the vectors are compatible.

1. If u, v, w are vectors,
$$(u \otimes v)w = (v \cdot w)u. \tag{2.2.11}$$
2. If a, b, c, d are vectors,
$$a \cdot (b \otimes c)d = (a \cdot b)(c \cdot d). \tag{2.2.12}$$
3. If $Q = v \otimes v$,
$$u \cdot Qu = u \cdot (v \otimes v)u = (u \cdot v)^2. \tag{2.2.13}$$
4. If A has columns $v_1, v_2, \ldots, v_N$,
$$AA^t = v_1 \otimes v_1 + v_2 \otimes v_2 + \cdots + v_N \otimes v_N. \tag{2.2.14}$$
5. For matrices A, B, and vectors u, v,
$$A(u \otimes v)B^t = (Au) \otimes (Bv). \tag{2.2.15}$$



If $A = (a_{ij})$ is any matrix, then the *norm squared* of A is

$$\|A\|^2 = \sum_{i,j} a_{ij}^2.$$

This equals $\operatorname{trace}(A^t A)$ which equals $\operatorname{trace}(AA^t)$. By taking the trace in (2.2.14),

Norm Squared of Matrix

Let A be a matrix with columns $v_1, v_2, \ldots, v_N$. Then

$$\|A\|^2 = |v_1|^2 + |v_2|^2 + \cdots + |v_N|^2, \tag{2.2.16}$$

and

$$\|A\|^2 = \operatorname{trace}(A^t A) = \operatorname{trace}(AA^t). \tag{2.2.17}$$

By replacing A by A^t, the same results hold for rows.



If $x_1, x_2, \ldots, x_N$ is a dataset of points, and $v_1, v_2, \ldots, v_N$ is the corresponding centered dataset, then the *variance matrix* $Q = (q_{ij})$ is the average of tensor products (§1.4),

$$Q = \frac{v_1 \otimes v_1 + v_2 \otimes v_2 + \cdots + v_N \otimes v_N}{N}. \tag{2.2.18}$$

Let A be the $d \times N$ matrix with columns $v_1, v_2, \ldots, v_N$,

$$A = \begin{pmatrix} v_1 & v_2 & \ldots & v_N \end{pmatrix}. \tag{2.2.19}$$

By (2.2.14),

$$Q = \frac{1}{N} A A^t. \tag{2.2.20}$$

Let A be the centered dataset matrix (2.2.19). By (2.2.20), each q_{ij} is the dot product of the i-th row and the j-th row of A. In particular, q_{ii} and q_{jj} are the lengths squared of the i-th and j-th rows of A. Hence, by the Cauchy-Schwarz inequality, $q_{ij}^2 \leq q_{ii} q_{jj}$.

Let `dataset` be the Iris dataset. If `centered` is the corresponding centered dataset, code for the variance is

```
from sklearn import datasets

iris = datasets.load_iris()
dataset = iris["data"]

# dataset is Nxd
mu = mean(dataset, axis = 0)
centered = dataset - mu

N = len(dataset)
dot(centered.T,centered) / N
```

Of course, it is simpler to avoid centering and just do directly

```
Q = cov(dataset.T, bias = True)

round(mu,decimals=2), round(Q,decimals=2),
    ↪ round(trace(Q),decimals=2)
```

obtaining the mean, the variance, and the total variance.

Recall (§1.4) a scalar dataset is standard if it has mean zero and variance one. We say a dataset with d features is *standard* if each feature of the dataset is standard.. When this happens, the variance $Q' = (q'_{ij})$ satisfies $q'_{ii} = 1$. From above, $q'^2_{ij} \le q_{ii}q_{jj}$, so for a standard dataset, $|q'_{ij}| \le 1$.

In §1.4, we standardized datasets in $\mathbf{R}^2$. To do this in $\mathbf{R}^d$, let $Q = (q_{ij})$ be the variance of a dataset $x_1, x_2, \ldots, x_N$ in $\mathbf{R}^d$, and set

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & 0 & \ldots & 0 \\ 0 & \sigma_2 & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & \sigma_d \end{pmatrix} = \begin{pmatrix} \sqrt{q_{11}} & 0 & 0 & \ldots & 0 \\ 0 & \sqrt{q_{22}} & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & \sqrt{q_{dd}} \end{pmatrix}.$$

Then the inverse Σ^{-1} is (inverses are studied in §2.3),

$$\Sigma^{-1} = \begin{pmatrix} 1/\sigma_1 & 0 & 0 & \ldots & 0 \\ 0 & 1/\sigma_2 & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 1/\sigma_d \end{pmatrix}.$$

and, by the same calculation as in §1.4,

$$x'_k = \Sigma^{-1}(x_k - \mu), \qquad k = 1, 2, \ldots, N,$$

is a standard dataset, with variance given by

$$Q' = \Sigma^{-1} Q \Sigma^{-1}.$$

Multiplying these three matrices, $Q' = (q'_{ij})$ has entries

$$q'_{ij} = \frac{q_{ij}}{\sqrt{q_{ii}q_{jj}}}, \qquad i, j = 1, 2, \ldots, d.$$

Since $q'_{ii} = 1$, this shows $x'_1, x'_2, \ldots, x'_N$ is standard.

The variance matrix of the standardized dataset is the *correlation matrix* of the original dataset, and the correlation q'_{ij} between feature i and feature j satisfies $|q'_{ij}| \le 1$.

As in §1.4, we say feature i and feature j are *positively correlated* or *negatively correlated* if $q'_{ij} > 0$ or $q'_{ij} < 0$ respectively.

Summarizing the above,

Variance and Correlation Matrices

- If $Q = (q_{ij})$ is the variance of a dataset, then $q_{ii} \ge 0$ and $q^2_{ij} \le q_{ii}q_{jj}$.
- If $Q' = (q'_{ij})$ is the variance of a standard dataset, then $|q'_{ij}| \le 1$.

- If Q' is the correlation matrix of a dataset with variance Q, then $q'_{ij} = q_{ij}/\sqrt{q_{ii}q_{jj}}$.

In Python,

```
from numpy import *
from scipy.linalg import inv
from sklearn.preprocessing import StandardScaler

# standardize dataset
standardized = StandardScaler().fit_transform(dataset)

Qcorr = corrcoef(dataset.T)
Qcov = cov(standardized.T,bias=True)

Sigma = diag(sqrt(diagonal(Q)))
Qprime = dot(inv(Sigma),dot(Q,inv(Sigma)))

Sigma, Qcov, Qcorr, Qprime
```

returns the standardized variance three ways.

Exercises

Exercise 2.2.1 In the Iris dataset, are the sepal length and sepal width positively or negatively correlated? What about sepal length and petal length?

Exercise 2.2.2 For $n = 1, 2, 3, \ldots$, let v be the vector

$$v = (1, 2, 3, \ldots, n).$$

Let $|v| = \sqrt{v \cdot v}$ be the length of v. Then, for example, when $n = 1$, $|v| = 1$ and, when $n = 2$, $|v| = \sqrt{5}$. There is one other n for which $|v|$ is a whole number. Use Python to find it.

Exercise 2.2.3 If μ is a unit vector and $Q = I - \mu \otimes \mu$, then $Q^2 = Q$.

Exercise 2.2.4 Give an example of a 3×3 matrix A satisfying $A^2 = 0$ but $A \neq 0$.

Exercise 2.2.5 If $Q^2 = 0$ and $Q^t = Q$, then $Q = 0$. (Compute $|Qx|^2$.)

Exercise 2.2.6 Matrices A and B *commute* if $AB = BA$. For what condition on a and b do these matrices commute?

$$A = \begin{pmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & b & 1 \end{pmatrix}.$$

Exercise 2.2.7 Use (2.2.10) and (2.2.15) to show

$$Au \cdot v = \text{trace}(A(u \otimes v)). \tag{2.2.21}$$

Exercise 2.2.8 Let Q and Q' be symmetric $d \times d$ matrices. Show that $Q = Q'$ iff

$$x \cdot Qx = x \cdot Q'x, \qquad \text{for all } x.$$

(Replace x by $u + v$ and expand, then insert u and v standard basis vectors.)

Exercise 2.2.9 Compute the means and variances μ_1, μ_2, μ_3 and Q_1, Q_2, Q_3 of the classes of the Iris dataset.

Exercise 2.2.10 With

```
from sympy import *

def row(i,d): return [ (-1)**(i+j) for j in range(d) ]
def R(d): return Matrix([ row(i,d) for i in range(d) ])
```

print $R(d)$ for $d = 1, 2, 3, \ldots$.

Exercise 2.2.11 With $R(d)$ as in Exercise **2.2.10**,

$$R(d)^3 = c(d) \times R(d)$$

for some scalar $c(d)$. Use Python to find $c(d)$. Here $d = 1, 2, 3, \ldots$.

Exercise 2.2.12 Suppose A and B are matrices with rows and columns

$$A = \begin{pmatrix} u_1 \\ u_2 \\ \ldots \\ u_N \end{pmatrix} \qquad \text{and} \qquad B = (v_1, v_2, \ldots, v_d),$$

all with the same dimension. Show that

$$AB = \begin{pmatrix} u_1 \cdot v_1 & u_1 \cdot v_2 & \ldots & u_1 \cdot v_d \\ u_2 \cdot v_1 & u_2 \cdot v_2 & \ldots & u_2 \cdot v_d \\ \ldots & \ldots & \ldots & \ldots \\ u_N \cdot v_1 & u_N \cdot v_2 & \ldots & u_N \cdot v_d \end{pmatrix}.$$

This generalizes (2.2.9).

Exercise 2.2.13 Suppose A and B are matrices with columns and rows

$$A = (u_1, u_2, \ldots, u_d) \qquad \text{and} \qquad B = \begin{pmatrix} v_1 \\ v_2 \\ \ldots \\ v_d \end{pmatrix}.$$

Use (2.2.5) to show

$$AB = u_1 \otimes v_1 + u_2 \otimes v_2 + \cdots + u_d \otimes v_d.$$

This generalizes (2.2.14).

Exercise 2.2.14 Let P and Q be permutation matrices (Exercise **2.1.1**). Show PQ is a permutation matrix.

Exercise 2.2.15 Let P be a 3×3 permutation matrix (Exercise **2.1.1**). Show that $P^6 = I$. Check this for every 3×3 permutation matrix. What about $d \times d$?

Exercise 2.2.16 The *matrix exponential* of a a square matrix A is

$$e^A = I + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + \dots$$

Let

$$N = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}.$$

Compute the matrix exponential of N and $I+N$ (for N the series terminates after finitely many terms).

2.3 Matrix Inverse

Let A be any matrix and b a vector. The goal is to solve the linear system

$$Ax = b. \tag{2.3.1}$$

In this section, we use the inverse A^{-1} and the pseudo-inverse A^+ to solve (2.3.1).

Of course, the system (2.3.1) doesn't even make sense unless the code

```
from numpy import *
set_printoptions(legacy = "1.25")

A = array([[1,2],[3,4]])
x = array([5,6])
b = dot(A,x)

A.shape == (len(b), len(x))
```

returns `True`. In what follows, we assume the dimensions of A, x, and b are compatible as above.

Even then, it's very easy to construct matrices A and vectors b for which the linear system (2.3.1) has no solutions at all! For example, take A the zero matrix and b any non-zero vector. Because of this, we must take some care when solving (2.3.1).



Given a square matrix A, the *inverse matrix* is the matrix B satisfying

$$AB = I = BA. \tag{2.3.2}$$

Here I is the identity matrix. Since I is a square matrix, A must also be a square matrix.

Only square matrices may have inverses. Moreover, not every square matrix has an inverse. For example, the zero matrix does not have an inverse. When A has an inverse, we say A is *invertible*.

If a matrix is $d \times d$, then the inverse is also $d \times d$. We write $B = A^{-1}$ for the inverse matrix of A. For example, it is easy to check

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \qquad \Longrightarrow \qquad A^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}.$$

Since we can't divide by zero, a 2×2 matrix is invertible only if $ad - bc \neq 0$.

Since

$$(AB)(B^{-1}A^{-1}) = A(BB^{-1})A^{-1} = AIA^{-1} = AA^{-1} = I,$$

we have

$$(AB)^{-1} = B^{-1}A^{-1}.$$



When A is invertible, the inverse A^{-1} provides a conceptual framework for solving the linear system $Ax = b$. Of course, a framework is not the same as a computational procedure. Many issues arise in the numerical construction of the inverse. These we sweep under the rug and ignore by accessing the inverse code `inv` in `numpy` and `sympy`.

Solution of $Ax = b$ when A invertible

Let A be a square matrix. If A is invertible, then

$$Ax = b \qquad \Longrightarrow \qquad x = A^{-1}b. \tag{2.3.3}$$

Conversely, if $Ax = b$ has a solution x for every b, then A is invertible.

The first statement is easy to check, since

$$Ax = A(A^{-1}b) = (AA^{-1})b = Ib = b.$$

The second statement follows from results in §2.9.

```
from scipy.linalg import inv

# solving Ax = b
x = dot(inv(A), b)

x
```

A matrix A cannot have more than one inverse. Either A has no inverse, or A has exactly one inverse. To verify this, suppose B and C are inverses of A. By definition, this means

$$AB = I = BA, \qquad AC = I = CA.$$

Then $B = C$ because

$$B = BI = B(AC) = (BA)C = IC = C.$$



In general, a matrix A is not invertible, and $Ax = b$ is solved using the pseudo-inverse $x = A^+b$. The definition and framework of the pseudo-inverse is in §2.6. The upshot is: *every (square or non-square) matrix A has a pseudo-inverse A^+*. Here is the general result.

Solution of $Ax = b$ for General A

If $Ax = b$ is solvable, then

$$x^+ = A^+b \qquad \Longrightarrow \qquad Ax^+ = b.$$

If $Ax = b$ is not solvable, then x^+ minimizes the residual $|Ax - b|^2$.

This says if $Ax = b$ has *some* solution, then $x^+ = A^+b$ is also a solution. On the other hand, $Ax = b$ may have no solution, in which case the error $|Ax - b|^2$ is minimized when $x = x^+$. From this point of view, it's best to think of x^+ as a *candidate* for a solution. It's a solution only after confirming equality of Ax^+ and b. All this is worked out in §2.6.

To put this in context, there are three possibilities for a linear system (2.3.1). A linear system $Ax = b$ can have

- no solutions, or

- exactly one solution, or
- infinitely many solutions.

As examples of these three possibilities, we have

- $A = 0$ and $b \neq 0$,
- A is invertible,
- $A = 0$ and $b = 0$.

The pseudo-inverse provides a conceptual framework for deciding among these three possibilities. Of course, a framework is not the same as a computational algorithm. Many issues arise in the numerical construction of the pseudo-inverse. These we ignore by accessing the pseudo-inverse code `pinv` in `numpy` and `sympy`.

In this section, we focus on using Python to solve $Ax = b$, and in §2.6, we explore the pseudo-inverse framework.

The rest of the section consists of examples of solving linear systems. The reader is encouraged to work out the examples below in Python. However, because the implementations of Python may differ, your solutions may differ from the solutions presented here.

It can be shown that if the entries of A are integers, then the entries of A^+ are fractions. This fact is reflected in `sympy`, but not in `numpy`, as the default in `numpy` is to work with floats.



Let

$$u = (1, 2, 3, 4, 5), v = (6, 7, 8, 9, 10), w = (11, 12, 13, 14, 15),$$

and let A be the matrix with columns u, v, w, and rows a, b, c, d, e,

$$A = \begin{pmatrix} u & v & w \end{pmatrix} = \begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix}. \tag{2.3.4}$$

```
# vectors
u = array([1,2,3,4,5])
v = array([6,7,8,9,10])
w = array([11,12,13,14,15])

# arrange as columns
A = column_stack([u,v,w])
A
```

For this A, the code

```
from scipy.linalg import pinv

pinvA = pinv(A)

round(pinvA,decimals=2), round(150*pinvA,decimals=2)
```

returns

$$A^+ = \frac{1}{150}\begin{pmatrix} -37 & -20 & -3 & 14 & 31 \\ -10 & -5 & 0 & 5 & 10 \\ 17 & 10 & 3 & -4 & -11 \end{pmatrix}.$$

Alternatively, in `sympy`,

```
from sympy import *

# column vectors
u = Matrix([1,2,3,4,5])
v = Matrix([6,7,8,9,10])
w = Matrix([11,12,13,14,15])

A = Matrix.hstack(u,v,w)

A.pinv()
```

returns the same result.



Let A be as in (2.3.4) and let

$$b_1 = (8, 9, 10, 11, 12), \qquad b_2 = (11, 6, 1, -4, -9).$$

We solve $Ax = b_1$ and $Ax = b_2$ by computing the candidates

$$x^+ = A^+ b_1 = \frac{1}{15}(2, 5, 8),$$

and

$$x^+ = A^+ b_2 = \frac{1}{30}(-173, -50, 73).$$

Then we check that the candidates are actually solutions, which they are, by comparing Ax^+ and b_1, in the first case, and Ax^+ and b_2, in the second case.



For

$$b_3 = (-9, -3, 3, 9, 10),$$

we have

$$x^+ = A^+ b_3 = \frac{1}{15}(82, 25, -32).$$

However, for this x^+, we have

$$Ax^+ = (-8, -3, 2, 7, 12),$$

which is not equal to b_3. From this, not only do we conclude x^+ is not a solution of $Ax = b_3$, but also, by the general result above, the system $Ax = b_3$ is not solvable at all.



Let B be the matrix with columns b_1 and b_2,

$$B = (b_1, b_2) = \begin{pmatrix} 8 & 11 \\ 9 & 6 \\ 10 & 1 \\ 11 & -4 \\ 12 & -9 \end{pmatrix}.$$

We solve

$$Bx = u, \qquad Bx = v, \qquad Bx = w$$

by constructing the candidates

$$B^+ u, \qquad B^+ v, \qquad B^+ w,$$

obtaining the solutions

$$x^+ = \frac{1}{51}(16, -7), \qquad x^+ = \frac{1}{51}(41, -2), \qquad x^+ = \frac{1}{51}(66, 3).$$



Let

$$C = A^t = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix}$$

and let $f = (0, -5, -10)$. By Exercise **2.6.8**, $C^+ = (A^+)^t$, so

$$C^+ = (A^+)^t = \frac{1}{150}\begin{pmatrix} -37 & -10 & 17 \\ -20 & -5 & 10 \\ -3 & 0 & 3 \\ 14 & 5 & -4 \\ 31 & 10 & -11 \end{pmatrix}$$

and

$$x^+ = C^+ f = \frac{1}{50}(32, 35, 38, 41, 44).$$

Once we confirm equality of Cx^+ and f, which is the case, we obtain a solution x^+ of $Cx = f$.



Let D be the matrix with columns a and f,

$$D = (a, f) = \begin{pmatrix} 1 & 0 \\ 6 & -5 \\ 11 & -10 \end{pmatrix},$$

where a, b, c, d, e are the rows of A, or, equivalently, the columns of C. Then

$$D^+ = \frac{1}{30}\begin{pmatrix} 25 & 10 & -5 \\ 28 & 10 & -8 \end{pmatrix}.$$

We solve

$$Dx = a, \qquad Dx = b, \qquad Dx = c, \qquad , Dx = d, \qquad Dx = e,$$

by constructing the candidates

$$D^+a, \qquad D^+b, \qquad D^+c, \qquad D^+d, \qquad D^+e,$$

obtaining the solutions

$$x^+ = (1, 0), \quad x^+ = (2, 1), \qquad x^+ = (3, 2), \quad x^+ = (4, 3), \qquad x^+ = (5, 4).$$

Exercises

Exercise 2.3.1 Verify the computations in this section using Python.

Exercise 2.3.2 With $R(d)$ as in Exercise **2.2.10**, find the formula for the inverse and pseudo-inverse of $R(d)$, whichever exists. Here $d = 1, 2, 3, \ldots$.

Exercise 2.3.3 The *sum matrix* and *difference matrix* are

$$S = \begin{pmatrix} 1&1&1&1&1\\ 0&1&1&1&1\\ 0&0&1&1&1\\ 0&0&0&1&1\\ 0&0&0&0&1 \end{pmatrix}, \qquad D = \begin{pmatrix} 1&-1&0&0&0\\ 0&1&-1&0&0\\ 0&0&1&-1&0\\ 0&0&0&1&-1\\ 0&0&0&0&1 \end{pmatrix}.$$

Compute SD and DS. What do you conclude?

Exercise 2.3.4 Let $D = D(d)$ be the $d \times d$ difference matrix as in Exercise **2.3.3**. Compute DD^t and D^tD, and SS^t and S^tS.

Exercise 2.3.5 Let u and v be vectors in $\mathbf{R}^d$ and let $A = I + u \otimes v$. Show that

$$A^{-1} = I - \frac{u \otimes v}{1 + u \cdot v}.$$

Exercise 2.3.6 Let P be a $d \times d$ permutation matrix (Exercise **2.1.1**). Show that P^t is the inverse of P.

Exercise 2.3.7 Let a, b, c be three distinct (non-equal) numbers, and let

$$V = \begin{pmatrix} 1&a&a^2\\ 1&b&b^2\\ 1&c&c^2 \end{pmatrix}.$$

Let $x = (r, s, t)$ and $p(z) = r + sz + tz^2$. Show $Vx = (p(a), p(b), p(c))$. Using Exercise **A.5.10**, conclude V is invertible.

2.4 Span and Linear Independence

In this section, the `sympy` code cells should be run *after* running the other code cells.

Let u, v, w be three vectors. Then

$$3u - \frac{1}{6}v + 9w, \qquad 5u + 0v - w, \qquad 0u + 0v + 0w$$

are linear combinations of u, v, w.

In general, a *linear combination* of vectors v_1, v_2, ..., v_d is

$$t_1v_1 + t_2v_2 + \cdots + t_dv_d. \tag{2.4.1}$$

Here the coefficients t_1, t_2, ..., t_d are scalars. In short, a linear combination is *a sum of scaled vectors.*

In terms of matrices, let

$$u = (1, 2, 3, 4, 5), v = (6, 7, 8, 9, 10), w = (11, 12, 13, 14, 15),$$

and let A be the matrix with columns u, v, w, as in (2.3.4). Let x be the vector $(r, s, t) = (1, 2, 3)$. Then an explicit calculation shows (do this calculation!) the matrix-vector product Ax equals $ru + sv + tw$,

$$Ax = ru + sv + tw.$$

The code

```
# In this section, run the sympy cells after the other code cells

from numpy import *
set_printoptions(legacy = "1.25")

# vectors
u = array([1,2,3,4,5])
v = array([6,7,8,9,10])
w = array([11,12,13,14,15])

# arrange as columns
A = column_stack([u,v,w])

r, s, t = 1, 2, 3
x = array([r,s,t])

dot(A,x) == r*u + s*v + t*w
```

returns `True`.

To repeat, the linear combination $ru + sv + tw$ is the same as the matrix-vector product Ax. This is a general fact on which everything depends:

Column Linear Combination Equals Matrix-Vector Product

Let A be a matrix with columns v_1, v_2, ..., v_d, and let

$$x = (t_1, t_2, \ldots, t_d).$$

Then

$$Ax = t_1v_1 + t_2v_2 + \cdots + t_dv_d, \tag{2.4.2}$$

In other words,

$$Ax = b \quad \text{is the same as} \quad b = t_1v_1 + t_2v_2 + \cdots + t_dv_d. \tag{2.4.3}$$



The *span* of vectors v_1, v_2, ..., v_d consists of *all linear combinations*

$$t_1v_1 + t_2v_2 + \cdots + t_dv_d$$

of the vectors. For example, $span(b)$ of a single vector b is the line through b, and $span(u, v, w)$ is the set of all linear combinations $ru + sv + tw$.

Span Definition I

The span of $v_1, v_2, \ldots, v_d$ is the set S of all linear combinations of $v_1, v_2, \ldots, v_d$, and we write

$$S = span(v_1, v_2, \ldots, v_d).$$

When we don't want to specify the vectors $v_1, v_2, v_3, \ldots, v_d$, we simply say S *is a span*.

From (2.4.2), we have

Span Definition II

Let A have columns $v_1, v_2, v_3, \ldots, v_d$. Then $span(v_1, v_2, \ldots, v_d)$ is the set S of all vectors of the form Ax.

If each vector v_k is a linear combination of vectors $w_1, w_2, \ldots, w_N$, then every vector v in $span(v_1, v_2, \ldots, v_d)$ is a linear combination of $w_1, w_2, \ldots, w_N$, so $span(v_1, v_2, \ldots, v_d)$ is contained in $span(w_1, w_2, \ldots, w_N)$.

If also each vector w_k is a linear combination of vectors $v_1, v_2, \ldots, v_d$, then every vector w in $span(w_1, w_2, \ldots, w_N)$ is a linear combination of $v_1, v_2, \ldots, v_d$, so $span(w_1, w_2, \ldots, w_N)$ is contained in $span(v_1, v_2, \ldots, v_d)$.

When both conditions hold, it follows

$$span(v_1, v_2, \ldots, v_d) = span(w_1, w_2, \ldots, w_N).$$

Thus there are many choices of spanning vectors for a given span.

For example, let u, v, w be the columns of A in (2.3.4). Let $\subset$ mean "is contained in". Then

$$span(u, v) \quad \subset \quad span(u, v, w),$$

since adding a third vector can only increase the linear combination possibilities. On the other hand, since $w = 2v - u$, we also have

$$span(u, v, w) \quad \subset \quad span(u, v).$$

It follows that

$$span(u, v, w) \quad = \quad span(u, v).$$



Let A be a matrix. The *column space* of A is the span of its columns. For A as in (2.3.4), the column space of A is $span(u, v, w)$. The code

```
from sympy import *

# column vectors
u = Matrix([1,2,3,4,5])
v = Matrix([6,7,8,9,10])
w = Matrix([11,12,13,14,15])

A = Matrix.hstack(u,v,w)

# returns minimal spanning set for column space of A
A.columnspace()
```

returns a *minimal list* of vectors spanning the column space of A. The *column rank* of A is the *length* of the list, i.e. the number of vectors returned.

For example, for A as in (2.3.4), this code returns the list

$$[u, v] = \left[\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix}, \begin{pmatrix} 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{pmatrix} \right].$$

Why is this? Because $w = 2v - u$, so

$$span(u, v, w) = span(u, v).$$

We conclude the column rank of A equals 2.



If the columns of A are v_1, v_2, ..., v_d, and $x = (t_1, t_2, \ldots, t_d)$ is a vector, then by definition of matrix-vector multiplication,

$$Ax = t_1 v_1 + t_2 v_2 + \cdots + t_d v_d.$$

By (2.4.3),

Column Space and $Ax = b$

The column space of a matrix A consists of all vectors of the form Ax. A vector b is in the column space of A when $Ax = b$ has a solution.



The corresponding code in `numpy` is

```
from scipy.linalg import orth

# vectors
u = array([1,2,3,4,5])
v = array([6,7,8,9,10])
w = array([11,12,13,14,15])

A = column_stack([u,v,w])

# returns minimal orthonormal spanning set
# for column space of A

orth(A)
```

This code returns the `array` in Figure 2.1.

```
array([[-0.35455706,  0.68868664],
       [-0.39869637,  0.37555453],
       [-0.44283568,  0.06242242],
       [-0.486975  , -0.2507097 ],
       [-0.53111431, -0.56384181]])
```

Fig. 2.1 Numpy column space array.

To explain this, let

$$b_1 = (8, 9, 10, 11, 12), \qquad b_2 = (11, 6, 1, -4, -9).$$

Then $b_1 \cdot b_2 = 0$, $|b_1| = \sqrt{510}$, $|b_2| = \sqrt{255}$, and the columns of the `array` in Figure 2.1 are the two *orthonormal* vectors $-b_1/|b_1|$ and $b_2/|b_2|$. (Why $-b_1/|b_1|$ instead of $b_1/|b_1|$? Because `numpy` has to make an arbitrary choice among the unit vectors $\pm b_1/|b_1|$.)

We conclude the column space of A can be described in at least three ways,

$$span(b_1, b_2) = span(u, v, w) = span(u, v).$$

Explicitly, b_1 and b_2 are linear combinations of u, v, w,

$$15b_1 = 2u + 5v + 8w, \qquad 30b_2 = -173u - 50v + 73w, \tag{2.4.4}$$

and u, v, w are linear combinations of b_1 and b_2,

$$51u = 16b_1 - 7b_2, \qquad 51v = 41b_1 - 2b_2, \qquad w = 2v - u. \tag{2.4.5}$$

By (2.4.3), to derive (2.4.4), we solve $Ax = b_1$ and $Ax = b_2$ for x. But this was done in §2.3.

Similarly, let B be the matrix with columns b_1 and b_2, and solve $Bx = u$, $Bx = v$, $Bx = w$, obtaining (2.4.5). This was also done in §2.3.

As a general rule, `sympy.columnspace` returns lists of spanning vectors, and `scipy.linalg.orth` returns arrays of orthonormal spanning vectors.



Let A be a matrix, and let b be a vector. How can we tell if b is in the column space of A? Given the above tools, here is an easy way to tell.

Write the *augmented matrix* $\bar{A} = (A, b)$; $\bar{A}$ obtained by adding b as an extra column next to the columns of A. If A is $d \times N$, then $\bar{A}$ is $d \times (N+1)$.

Given A and $\bar{A} = (A, b)$, compute their column ranks. Let v_1, v_2, ..., v_N be the columns of A. If these ranks are equal, then

$$span(v_1, v_2, \ldots, v_N) = span(v_1, v_2, \ldots, v_N, b),$$

so b is a linear combination of the columns, or b is in the column space of A.

Column Space of Augmented Matrix

Let $\bar{A}$ be the matrix A augmented by a vector b. Then $Ax = b$ is solvable iff b is in the column space of A iff

$$\text{column rank}(A) = \text{column rank}(\bar{A}). \tag{2.4.6}$$

For example, let $b_3 = (-9, -3, 3, 9, 10)$ and let $\bar{A} = (A, b_3)$. Using Python, check the column rank of $\bar{A}$ is 3. Since the column rank of A is 2, we conclude b_3 is not in the column space of A, so b_3 is not a linear combination of u, v, w.

When (2.4.6) holds, b is a linear combination of the columns of A. However, (2.4.6) does not tell us which linear combination. According to (2.4.3), finding the specific linear combination is equivalent to solving $Ax = b$.



$\mathbf{R}^3$ consists of all vectors (r, s, t) in three dimensions. If

$$e_1 = (1, 0, 0), \qquad e_2 = (0, 1, 0), \qquad e_3 = (0, 0, 1),$$

then

$$(r, s, t) = re_1 + se_2 + te_3.$$

This shows the vectors e_1, e_2, e_3 span $\mathbf{R}^3$, or

$$\mathbf{R}^3 = span(e_1, e_2, e_3).$$

As a consequence, $\mathbf{R}^3$ *is a span.* Similarly, in dimension d, we can write

$$\begin{aligned} e_1 &= (1, 0, 0, \ldots, 0, 0) \\ e_2 &= (0, 1, 0, \ldots, 0, 0) \\ e_3 &= (0, 0, 1, \ldots, 0, 0) \\ \ldots &= \ldots \\ e_d &= (0, 0, 0, \ldots, 0, 1) \end{aligned} \tag{2.4.7}$$

Then e_1, e_2, ..., e_d span $\mathbf{R}^d$, so

Standard Basis Spans

$\mathbf{R}^d$ is a span.

Following machine-learning terminology, a vector $v = (v_1, v_2, \ldots, v_d)$ is *one-hot encoded at slot* j if all components of v are zero except the j-th component. For example, when $d = 3$, the vectors

$$(a, 0, 0), \qquad (0, a, 0), \qquad (0, 0, a)$$

are one-hot encoded.

Sometimes one-hot encoded also means the nonzero slot must be a one. With this interpretation, when $d = 3$, the only one-hot encoded vectors

$$(1, 0, 0), \qquad (0, 1, 0), \qquad (0, 0, 1).$$

We use both interpretations.

The vectors e_1, e_2, ..., e_d are one-hot encoded. These vectors are the *standard basis* for $\mathbf{R}^d$, or *the one-hot encoded basis* for $\mathbf{R}^d$.



The *row space* of a matrix is the span of its rows.

```
from sympy import *

# returns minimal spanning set for row space of A
A.rowspace()
```

The *row rank* of a matrix is the number of vectors returned by `rowspace()`. This is the *minimal* number of vectors spanning the row space of A.

For example, call the rows of A in (2.3.4) a, b, c, d, e. Let

$$f = (0, -5, -10).$$

Then `sympy.rowspace` returns the vectors a and f, so

$$span(a, b, c, d, e) = span(a, f).$$

Explicitly, the linear combination

$$50f = 32a + 35b + 38c + 41d + 44e$$

is derived using $C = A^t$ and solving $Cx = f$. The linear combinations

$$a = a + 0f, \quad b = 2a - 5f, \quad c = 3a - 10f, \quad d = 4a - 15f, \quad e = 5a - 20f$$

are derived using $D = (a, f)$ and solving $Dx = a$, $Dx = b$, $Dx = c$, $Dx = d$, $Dx = e$. Again, these linear systems were solved in §2.3.

Since the transpose interchanges rows and columns, the row space of A equals the column space of A^t. Using this, we compute the row space in `numpy` by

```
from scipy.linalg import orth

# returns minimal spanning set for row space of A
orth(A.T)
```

`Numpy` returns *orthonormal* vectors.

Clearly, when Q is symmetric, the row space of Q equals the column space of Q.

It turns out the column rank equals the row rank, for any matrix. Even though we won't establish this till (2.9.1), we state this result here, because it helps ground the concepts.

Column Rank Equals Row Rank

For any matrix, the row rank equals the column rank.

Because of this, we refer to this common number as the *rank* of the matrix.



A linear combination $t_1v_1 + t_2v_2 + \cdots + t_dv_d$ is *trivial* if all the coefficients are zero, $t_1 = t_2 = \cdots = t_d = 0$. Otherwise it is *non-trivial*, if at least one coefficient is not zero. A linear combination $t_1v_1 + t_2v_2 + \cdots + t_dv_d$ *vanishes* if it equals the zero vector,

$$t_1v_1 + t_2v_2 + \cdots + t_dv_d = 0.$$

For example, with u, v, w as above, we have $w = 2v - u$, so

$$ru + sv + tw = 1u - 2v + 1w = 0 \tag{2.4.8}$$

is a vanishing non-trivial linear combination of u, v, w.

We say $v_1, v_2, \ldots, v_d$ are *linearly dependent* if there is a vanishing non-trivial linear combination of $v_1, v_2, \ldots, v_d$. Otherwise, if there is no non-trivial vanishing linear combination, we say $v_1, v_2, \ldots, v_d$ are *linearly independent*. For example, u, v, w above are linearly dependent.

Suppose u, v, w are any three vectors, and suppose u, v, w are linearly dependent. Then we have $ru + sv + tw = 0$ for some scalars r, s, t, where at least one is not zero. If $r \neq 0$, then we may solve for u, obtaining

$$u = -(s/r)v - (t/r)w.$$

If $s \neq 0$, then we may solve for v, obtaining

$$v = -(r/s)u - (t/s)w.$$

If $t \neq 0$, then

$$w = -(r/t)u - (s/t)v.$$

Hence linear dependence of u, v, w means one of the three vectors is a multiple of the other two vectors.

In general, a vanishing non-trivial linear combination of $v_1, v_2, \ldots, v_d$, or linear dependence of $v_1, v_2, \ldots, v_d$, is the same as saying one of the vectors is a linear combination of the remaining vectors.

In terms of matrices,

Homogeneous Linear Systems

Let A be the matrix with columns $v_1, v_2, \ldots, v_d$. Then

- $v_1, v_2, \ldots, v_d$ are linearly dependent when $Ax = 0$ has a nonzero solution x, and
- $v_1, v_2, \ldots, v_d$ are linearly independent when $Ax = 0$ has only the zero solution $x = 0$.



The set of vectors x satisfying $Ax = 0$, or the set of *solutions* x of $Ax = 0$, is the *nullspace* of the matrix A.

With this terminology, $v_1, v_2, \ldots, v_d$ are linearly dependent when there is a nonzero nullspace for the matrix A.

For example, with A as in (2.3.4), the `sympy` code

```
from sympy import *

A.nullspace()
```

returns a list with a single vector,

$$\left[\begin{pmatrix} r \\ s \\ t \end{pmatrix}\right] = \left[\begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix}\right].$$

This says the nullspace of A consists of all multiples of $(1, -2, 1)$. Since the code

```
[r,s,t] = A.nullspace()[0]

r*u + s*v + t*w
```

returns the column vector

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

we have $Ax = 0$, in agreement with (2.4.8).



The corresponding `numpy` code is

```
from scipy.linalg import null_space

null_space(A)
```

This code returns the *unit* vector

$$\frac{-1}{\sqrt{6}} \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix},$$

which is a multiple of $(1, -2, 1)$. `scipy.linalg.null_space` always returns *orthonormal* vectors.



Here is a simple result that is used frequently.

A Versus A^tA

Let A be any matrix. The nullspace of A equals the nullspace of A^tA.

If x is in the nullspace of A, then $Ax = 0$. Multiplying by A^t leads to $A^tAx = 0$, so x is in the nullspace of A^tA.

Conversely, if x is in the nullspace of A^tA, then $A^tAx = 0$. By the dot-product-transpose identity (2.2.8),

$$|Ax|^2 = Ax \cdot Ax = x \cdot A^tAx = 0,$$

so $Ax = 0$, which means x is in the nullspace of A.



An important example of linearly independent vectors are orthonormal vectors.

Orthonormal Implies Linearly Independent

If $v_1, v_2, \ldots, v_d$ are orthonormal, they are linearly independent.

To see this, suppose we have a vanishing linear combination

$$t_1v_1 + t_2v_2 + \cdots + t_dv_d = 0.$$

Take the dot product of both sides with v_1. Since the dot products of any two vectors is zero, and each vector has length one, we obtain

$$t_1 = t_1v_1 \cdot v_1 = t_1v_1 \cdot v_1 + t_2v_2 \cdot v_1 + \cdots + t_dv_d \cdot v_1 = 0.$$

Similarly, all other coefficients t_k are zero. This shows $v_1, v_2, \ldots, v_d$ are linearly independent.



In general, `nullspace()` returns a *minimal* set of vectors spanning the nullspace of A. The *nullity of* A is the number of vectors returned by the method `nullspace()`.

For example, to compute the nullspace of the matrix

$$C = A^t = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix},$$

we solve $Cx = 0$. Since the code

```
from sympy import *

u = Matrix([1,2,3,4,5])
v = Matrix([6,7,8,9,10])
w = Matrix([11,12,13,14,15])
```

```
A = Matrix.hstack(u,v,w)
C = A.T

C.nullspace()
```

returns the list of three vectors

$$\left[\begin{pmatrix}1\\-2\\1\\0\\0\end{pmatrix}, \begin{pmatrix}2\\-3\\0\\1\\0\end{pmatrix}, \begin{pmatrix}3\\-4\\0\\0\\1\end{pmatrix}\right],$$

here we can make three conclusions: (1) the nullspace of C is spanned by three vectors, (2) this is the least number of vectors that spans the nullspace of C, and (3) the nullity of C is 3.



Let m be a nonzero vector, and let

$$m^\perp = \{v : m \cdot v = 0\}\,. \tag{2.4.9}$$

Then $m^\perp$ (pronounced "m-perp"), the *orthogonal complement of* m, is a span and consists of all vectors orthogonal to m. From the definition, when t is a nonzero scalar, $(tm)^\perp = m^\perp$.

When m is in $\mathbf{R}^2$, $m^\perp$ is a line, and we defined $m^\perp$ in §A.4 to be a vector spanning that line. Specifically, when $m = (x, y)$, in §A.4 we defined $m^\perp = (-y, x)$.

More generally, suppose S is any collection of vectors, not necessarily a span, and let

$$S^\perp = \{v : m \cdot v = 0 \text{ for all } m \text{ in } S\}\,.$$

Then $S^\perp$ (pronounced "S-perp"), the *orthogonal complement of* S, is a span (even if S isn't) and consists of all vectors orthogonal to all vectors in S.

Suppose S consists of five vectors a, b, c, d, e. How do we compute $S^\perp$? The answer is by using `nullspace`: Let A be the matrix with rows a, b, c, d, e. By matrix-vector multiplication,

$$0 = Ax = \begin{pmatrix}a\\b\\c\\d\\e\end{pmatrix} x = \begin{pmatrix}a\cdot x\\b\cdot x\\c\cdot x\\d\cdot x\\e\cdot x\end{pmatrix}.$$

This shows x is orthogonal to a, b, c, d, e exactly when x is in the nullspace of A. Thus $S^\perp$ equals the nullspace of A.

In general, if $S = span(v_1, v_2, \ldots, v_N)$, let A be the matrix with rows v_1, v_2, …, v_N. Then $S^\perp$ equals the nullspace of A.



An important example of orthogonality is the relation between row space and the nullspace. Suppose A has rows v_1, v_2, …, v_N, and x is a vector, all of the same dimension. Then, by definition, the matrix-vector product is

$$Ax = (v_1 \cdot x, v_2 \cdot x, \ldots, v_N \cdot x).$$

If x is in the nullspace, $Ax = 0$, then

$$v_1 \cdot x = 0, v_2 \cdot x = 0, \ldots, v_N \cdot x = 0,$$

so x is orthogonal to the rows of A. Conversely, if x is orthogonal to the rows of A, then $Ax = 0$.

This shows *the nullspace of A and the row space of A are orthogonal complements*. Summarizing, we write

Row Space and Null Space are Orthogonal

Every vector in the row space is orthogonal to every vector in the nullspace,

$$\text{rowspace}^\perp = \text{nullspace} \qquad \text{and} \qquad \text{nullspace}^\perp = \text{rowspace}. \quad (2.4.10)$$

Actually, the above paragraph only established the first identity. For the second identity, we need to use the result (2.7.9) in a later section, as follows

$$\text{rowspace} = \left(\text{rowspace}^\perp\right)^\perp = \text{nullspace}^\perp.$$



Since the row space is the orthogonal complement of the nullspace, and the nullspace of A equals the nullspace of A^tA, we conclude

A Versus A^tA

Let A be any matrix. Then the row space of A equals the row space of A^tA.

Now replace A by A^t in this last result. Since the row space of A^t equals the column space of A, and AA^t is symmetric, we also have

A Versus AA^t

Let A be any matrix. Then the column space of A equals the column space of AA^t.



Let A be a matrix and b a vector. So far we've met four spaces,

- the nullspace: all x's satisfying $Ax = 0$,
- the row space: the span of the rows of A,
- the column space: the span of the columns of A,
- the solution space: the solutions x of $Ax = b$.

A set S of vectors is a *subspace* if $x_1 + x_2$ is in S whenever x_1 and x_2 are in S, and tx is in S whenever x is in S. When this happens, we say S is closed under addition and scalar multiplication: A subspace is a set of vectors closed under addition and scalar multiplication.

Since a linear combination of linear combinations is a linear combination, every span is a subspace. In particular, $\mathbf{R}^d$ is a subspace.

It's important to realize the first three are subspaces, but the fourth is not.

- If x_1 and x_2 are in the nullspace, and r_1 and r_2 are scalars, then so is $r_1x_1 + r_2x_2$, because

$$A(r_1x_1 + r_2x_2) = r_1Ax_1 + r_2Ax_2 = r_10 + r_20 = 0.$$

  This shows the nullspace is a subspace. In particular, $S^\perp$ is a subspace for any S.
- The row space is a span, so is a subspace.
- The column space is a span, so is a subspace.
- The solution space S of $Ax = b$ is not a subspace, nor a span: If x is in S, then $Ax = b$, so $A(5x) = 5Ax = 5b$, so $5x$ is not in S.

If x_1 and x_2 are solutions of $Ax = b$, then $A(x_1 + x_2) = 2b$, so the solution space is not a subspace. However

$$A(x_1 - x_2) = b - b = 0, \tag{2.4.11}$$

so the difference $x_1 - x_2$ of any two solutions x_1 and x_2 is in the nullspace of A, which is a span.



Let A be an $N \times d$ matrix. Then matrix multiplication by A transforms a vector x to the vector $b = Ax$. Since A is $N \times d$, x is in $\mathbf{R}^d$, and Ax is in $\mathbf{R}^N$.

From this point of view, the *source space* of A is $\mathbf{R}^d$, and the *target space* of A is $\mathbf{R}^N$.

Locations of Column, Row, and Null Spaces

Let A be any matrix. The nullspace of A and the row space of A are in the source space of A, and the column space of A is in the target space of A.



Let A be a $d \times d$ invertible matrix. Then the source space is $\mathbf{R}^d$ and the target space is $\mathbf{R}^d$. If $Ax = 0$, then

$$x = (A^{-1}A)x = A^{-1}(Ax) = A^{-1}0 = 0.$$

This shows the nullspace of an invertible matrix is zero, hence the nullity is zero.

Since the row space is the orthogonal complement of the nullspace, we conclude the row space is all of $\mathbf{R}^d$.

In §2.9, we see that the column rank and the row rank are equal. From this, we see also the column space is all of $\mathbf{R}^d$. In summary,

Null Space of Invertible Matrix

Let A be a $d \times d$ invertible matrix. Then the nullspace is zero, and the row space and column space are both $\mathbf{R}^d$. In particular, the nullity is 0, and the row rank rank and column rank are both d.

Exercises

Exercise 2.4.1 For what condition on a, b, c do the vectors $(1, a)$, $(2, b)$, $(3, c)$ lie on a line?

Exercise 2.4.2 Let

$$C = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{pmatrix}, \qquad x = \begin{pmatrix} 16 \\ 17 \\ 18 \\ 19 \\ 20 \end{pmatrix}.$$

Compute Cx in two ways, first by row times column, then as a linear combination of the columns of C.

Exercise 2.4.3 Check that the array in Figure 2.1 matches with b_1, b_2 as explained in the text, and the vectors b_1 and b_2 are orthogonal.

Exercise 2.4.4 [35] Let $a = (1, 1, 0, 0)$, $b = (0, 0, 1, 1)$, $c = (1, 0, 1, 0)$, $d = (0, 1, 0, 1)$. Check whether or not a, b, c, d are linearly independent by solving $ra + sb + tc + ud = 0$. Is $ra + sb + tc + ud = (0, 0, 0, 1)$ solvable? Do a, b, c, d span $\mathbf{R}^4$?

Exercise 2.4.5 Let A be as in (2.3.4) and let $b = (16, 17, 18, 19, 20)$. Is b in the column space of A? If yes, solve $b = ru + sv + tw$.

Exercise 2.4.6 Let $A = (u, v, w)$ be as in (2.3.4) and let $Q = A^tA$. What are the source and target spaces for A and Q? Calculate column spaces, row spaces, and nullspaces of A and Q. How are they related?

Exercise 2.4.7 Let $A = (u, v, w)$ be as in (2.3.4) and let $Q = AA^t$. What are the source and target spaces for A and Q? Calculate column spaces, row spaces, and nullspaces of A and Q. How are they related?

Exercise 2.4.8 [35] Let A be a 64×17 matrix with rank 11. How many linearly independent vectors x solve $Ax = 0$? How many linearly independent vectors x solve $A^tx = 0$?

Exercise 2.4.9 Let $A(N, d)$ be the matrix returned by the code

```
from sympy import *

def col(N,j): return Matrix([ 1+i+j*N for i in range(N) ])
def A(N,d): return Matrix.hstack(*[ col(N,j) for j in range(d) ])
```

What are $A(5, 3)$ and $A(3, 5)$? What are the source and target spaces for $A(N, d)$?

Exercise 2.4.10 Calculate the column rank of the matrix $A(N, d)$ for all $N \geq 2$ and all $d \geq 2$. (Column rank is the length of the list `columnspace` returns.)

Exercise 2.4.11 What is the nullity of the matrix $A(N, d)$ for all $N \geq 2$ and all $d \geq 2$?

Exercise 2.4.12 Show directly from the definition the vectors

$$u = (2, *, *, *, *, *), v = (0, 7, *, *, *, *), w = (0, 0, 0, 1, *, *), x = (0, 0, 0, 0, 0, 3)$$

are linearly independent.

Exercise 2.4.13 Let a, b, c, d be the rows of the matrix

$$E = \begin{pmatrix} 2 & 1 & 0 & 1 & 3 & 7 \\ \boxed{0} & 7 & 7 & 2 & 0 & 5 \\ 0 & 0 & 0 & \boxed{1} & 3 & 1 \\ 0 & 0 & 0 & 0 & \boxed{0} & 3 \end{pmatrix}$$

Show directly from the definition a, b, c, d are linearly independent. A matrix with this staircase pattern is in *echelon form*.

Exercise 2.4.14 Let E be the matrix in Exercise **2.4.13**. Solve $Ex = 0$ to obtain the nullspace of E and the nullity of E.

Exercise 2.4.15 [36] Let x, y, z be three nonzero vectors, and $w = 2y - 2x + z$. If $z = x - y$, find r and s with $w = rx + sy$. Which of the following must be true?

1. $span(x, y, z) = span(w, y, z)$,
2. $span(w, z) = span(y, z)$,
3. $span(x, z) = span(x, z, w)$,
4. $span(x, z) = span(w, x)$,
5. $span(w, x, y) = span(w, x, z)$.

Exercise 2.4.16 [36] Let a be a linear combination of x, y, z. Select the best statement.

1. $span(u, v, w)$ is contained or equal to $span(u, v, w, a)$,
2. $span(u, v, w)$ is equal to $span(u, v, w, a)$,
3. There is no obvious relation between $span(u, v, w)$ and $span(u, v, w, a)$,
4. $span(u, v, w)$ is not equal to $span(u, v, w, a)$.

2.5 Zero Variance Directions

Let $x_1, x_2, \ldots, x_N$ be a dataset in $\mathbf{R}^d$. Then $x_1, x_2, \ldots, x_N$ are N points in $\mathbf{R}^d$, and each x has d features, $x = (t_1, t_2, \ldots, t_d)$. From §1.4, the mean is

$$\mu = \frac{x_1 + x_2 + \cdots + x_N}{N}.$$

Center the dataset (see §1.3)

$$v_1 = x_1 - \mu, v_2 = x_2 - \mu, \ldots, v_N = x_N - \mu,$$

and let A be the dataset matrix with columns v_1, v_2, $\ldots$, v_N. By (2.2.14), the variance is

$$Q = \frac{v_1 \otimes v_1 + v_2 \otimes v_2 + \cdots + v_N \otimes v_N}{N} = \frac{1}{N} AA^t. \tag{2.5.1}$$

If u is a unit vector, the projection of the centered dataset onto the line through u results in the reduced dataset

$$v_1 \cdot u, v_2 \cdot u, \ldots, v_N \cdot u.$$

This reduced dataset is centered, and, by (2.2.13), its variance is

$$q = \frac{(v_1 \cdot u)^2 + (v_2 \cdot u)^2 + \cdots + (v_N \cdot u)^2}{N} = \frac{1}{N} v^t A^t A u = u \cdot Qu. \qquad (2.5.2)$$

We obtain this result, which was first stated in §1.4.

Variance of Reduced Dataset

Let Q be the variance matrix of a dataset and let u be a unit vector. Then the variance of the reduced dataset onto the line through the vector u equals the quadratic function $u \cdot Qu$.

A vector u is a *zero variance direction* if the reduced variance is zero,

$$u \cdot Qu = 0.$$

We investigate zero variance directions, but first we need a definition.

Let b be a scalar and m a nonzero vector in $\mathbf{R}^d$. A *hyperplane* (m, b) is the set of points x satisfying the equation

$$m \cdot x + b = 0.$$

This is a hyperplane *orthogonal to* m.

In $\mathbf{R}^3$, a hyperplane is a plane, in $\mathbf{R}^2$, a hyperplane is a line, and in $\mathbf{R}$, a hyperplane is a point, a threshhold. In general, in $\mathbf{R}^d$, a hyperplane is $(d-1)$-dimensional, one dimension less than the ambient dimension. When $b = 0$, the hyperplane orthogonal to m is $m^\perp$ (2.4.9).

The hyperplane (m, b) *passes through a point* μ if

$$m \cdot \mu + b = 0.$$

By subtracting the last two equations, the equation of the hyperplane orthogonal to m and passing through μ is

$$m \cdot (x - \mu) = 0.$$

If (m, b) is a hyperplane and t is a nonzero scalar, then (tm, tb) results in the same hyperplane. Thus, by dividing m by its length, one may assume that m is a unit vector.



Let $x_1, x_2, \ldots, x_N$ be a dataset. If $x_1, x_2, \ldots, x_N$ lies in a hyperplane (m, b), then

$$m \cdot x_k + b = 0, \qquad k = 1, 2, \ldots, N.$$

If μ is the mean of the dataset, then by summing these equations, $m \cdot \mu + b = 0$. Subtracting, the centered dataset $v_1 = x_1 - \mu$, $v_2 = x_2 - \mu$, $\ldots$, $v_N = x_N - \mu$ satisfies

$$m \cdot v_k = 0, \qquad k = 1, 2, \ldots, N.$$

Thus a dataset lies in a hyperplane iff the corresponding centered dataset lies in $m^\perp$, for some nonzero vector m. Since $m \neq 0$, this happens only when the centered dataset does not span the sample space. We conclude

Datasets Lying in Hyperplanes

A dataset lies in a hyperplane iff the corresponding centered dataset does not span the sample space. Equivalently, a dataset does not lie in a hyperplane iff the corresponding centered dataset spans the sample space.



We can now state

Zero Variance Directions

Let μ and Q be the mean and variance of a dataset in $\mathbf{R}^d$, and let m be a nonzero vector. Then $m \cdot Qm = 0$ iff the dataset lies in the hyperplane passing through μ and orthogonal to m.

This is easy to see. Let the dataset be $x_1, x_2, \ldots, x_N$, and center it to $v_1, v_2, \ldots, v_N$. If $m \cdot Qm = 0$, then, by (2.5.2), $v_k \cdot m = 0$ for $k = 1, 2, \ldots, N$. Thus $x_1, x_2, \ldots, x_N$ lie on the hyperplane $m \cdot (x - \mu) = 0$. Here are some examples.

In two dimensions $\mathbf{R}^2$, a line is determined by a point on the line and a vector orthogonal to the line. If $m = (a, b)$ is the vector orthogonal to the line and (x_0, y_0), (x, y) are points on the line, then $(x, y) - (x_0, y_0)$ is orthogonal to m, or

$$(a, b) \cdot ((x, y) - (x_0, y_0)) = 0.$$

Writing this out, the equation of the line is

$$a(x - x_0) + b(y - y_0) = 0, \qquad \text{or} \qquad ax + by = c,$$

where $c = ax_0 + by_0$.

If the mean and variance of a dataset are $\mu = (2, 3)$ and

$$Q = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix},$$

and $m = (1, 1)$, then $Qu = 0$, so $m \cdot Qm = 0$. Since the line $x + y = 5$ passes through the mean, the dataset lies on this line. We conclude this dataset is one-dimensional.

If

$$Q = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix},$$

and $m = (x, y)$, then

$$m \cdot Qm = 3x^2 + y^2,$$

so $m \cdot Qm$ is never zero unless $m = 0$. In this case, we conclude the dataset is two-dimensional, because it does not lie on a line.

In three dimensions $\mathbf{R}^3$, a plane is determined by a point (x_0, y_0, z_0) and a vector $m = (a, b, c)$. The point is in the plane, and the vector is orthogonal to the plane. If (x, y, z) is any point in the plane, then $(x, y, z) - (x_0, y_0, z_0)$ is orthogonal to m, so the equation of the plane is

$$(a, b, c) \cdot ((x, y, z) - (x_0, y_0, z_0)) = 0, \qquad \text{or} \qquad ax + by + cz = d,$$

where $d = ax_0 + by_0 + cz_0$.

Suppose we have a dataset in $\mathbf{R}^3$ with mean $\mu = (3, 2, 1)$, and variance

$$Q = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}. \tag{2.5.3}$$

Let $m = (2, -1, -1)$. Then $Qm = 0$, so $m \cdot Qm = 0$. We conclude the dataset lies in the plane

$$(2, -1, -1) \cdot ((x, y, z) - (x_0, y_0, z_0)) = 0, \qquad \text{or} \qquad 2x - y - z = 3.$$

In this case, the dataset is two-dimensional, it lies in a plane.

If a dataset has variance the 3×3 identity matrix I, then $m \cdot Im$ is never zero unless $m = 0$. Such a dataset is three-dimensional, it does not lie in a plane.

Sometimes there may be several zero variance directions. For example, for the variance (2.5.3) and $u = (2, -1, -1)$, $v = (0, 1, -1)$, we have both

$$u \cdot Qu = 0 \qquad \text{and} \qquad v \cdot Qv = 0.$$

From this we see the dataset corresponding to this Q lies in two planes: the plane orthogonal to u, and the plane orthogonal v. Since the intersection of two planes is a line, the dataset lies in a line, it is one-dimensional.

Which line does this dataset lie in? Well, the line has to pass through the mean, and is orthogonal to u and v. If we find a vector b satisfying $b \cdot u = 0$

and $b \cdot v = 0$, then the line will pass through the mean and will be parallel to b. But we know how to find such a vector. Let A be the matrix with rows u, v. Then b in the nullspace of A fulfills the requirements. We obtain $b = (1, 1, 1)$.



Let $v_1, v_2, \ldots, v_N$ be a centered dataset of vectors in $\mathbf{R}^d$, and let Q be the variance matrix of the dataset. If u is in the nullspace of Q, then $Qu = 0$, so $u \cdot Qu = 0$. This shows every vector in the nullspace is a zero variance direction. What is less clear is that this works in the other direction.

Zero Variance Directions and Nullspace I

Let Q be a variance matrix. Then the nullspace of Q equals the zero variance directions of Q.

To see this, we use the quadratic equation. If Q is symmetric, then $u \cdot Qv = v \cdot Qu$. For t scalar and u, v vectors, since $Q \geq 0$, the function

$$(v + tu) \cdot Q(v + tu)$$

is nonnegative for all t scalar. Expanding this function into powers of t, we see

$$t^2 u \cdot Qu + 2tu \cdot Qv + v \cdot Qv = at^2 + 2bt + c$$

is nonnegative for all t scalar. Thus the bottom of the parabola $at^2 + 2bt + c$ is nonnegative. By completing the square, the bottom is $c - b^2/a$, so $b^2 \leq ac$, which yields

$$(u \cdot Qv)^2 \leq (u \cdot Qu)\,(v \cdot Qv). \tag{2.5.4}$$

Now we can derive the result. If u is a zero variance direction, then $u \cdot Qu = 0$. By (2.5.4), this implies $u \cdot Qv = 0$ for all v, so $Qu = 0$, so u is in the nullspace of Q. This derivation is valid for any nonnegative matrix Q, not just variance matrices. Later (§3.2) we see every nonnegative matrix is the variance matrix of a dataset.



Based on the above result, here is code that returns zero variance directions.

```
from numpy import *
from scipy.linalg import null_space

from numpy.random import default_rng
samples = default_rng().random
```

```
d, N = 2, 20
dataset  = samples((d,N))

def zero_variance(dataset):
   Q = cov(dataset)
   return null_space(Q)

zero_variance(dataset)
```

Let A be an $N \times d$ dataset matrix, and let Q be the variance of the dataset. If the dataset is centered, by (2.2.20), $Q = A^t A/N$. Then the nullspace of Q equals the nullspace of $A^t A$, which equals the nullspace of A. We conclude

Zero Variance Directions and Nullspace II

Let Q be a variance matrix of a centered dataset A. Then the nullspace of A equals the zero variance directions of Q.

Suppose the dataset is

$$(1, 2, 3, 4, 5), (6, 7, 8, 9, 10), (11, 12, 13, 14, 15), (16, 17, 18, 19, 20).$$

This is four vectors in $\mathbf{R}^5$. Since it is only four vectors, it is at most a four-dimensional dataset. The code `zero_variance` returns three vectors

$$(1, -2, 1, 0, 0), (2, -3, 0, 1, 0), (3, -4, 0, 0, 1).$$

Thus this dataset is orthogonal to three directions, hence lies in the intersection of three hyperplanes. Each hyperplane is one condition, so each hyperplane cuts the dimension down by one, so the dimension of this dataset is $5 - 3 = 2$. Dimension of a dataset is discussed further in §2.9.

2.6 Pseudo-Inverse

What is the pseudo-inverse? In §2.3, we used both the inverse and the pseudo-inverse to solve $Ax = b$, but we didn't explain the framework behind them. It turns out the framework is best understood geometrically. Throughout this section, we refer to

$$Ax = b$$

as *the linear system.*

Think of b and Ax as points, and measure the distance between them, and think of x and the origin 0 as points, and measure the distance between them (Figure 2.2).

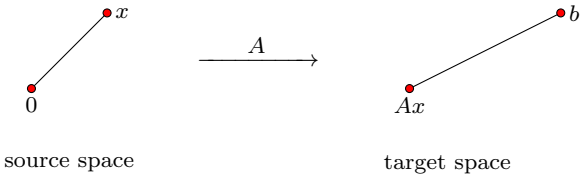


Fig. 2.2 The points 0, x, Ax, and b.

If the linear system is solvable, and x^* is a solution, then the distance between Ax^* and b is zero. Among all solutions x^*, select the solution x^+ closest to 0.

More generally, if the linear system is not solvable, select the points x^* so that Ax^* is closest to b, then, among all such x^*, select the point x^+ closest to the origin (this is "closest twice").

Even though the point x^+ may not solve the linear system, this procedure results in a uniquely determined x^+: While there may be several points x^*, there is only one x^+. Figure 2.3 summarizes the situation for a 2×2 matrix A with $rank(A) = 1$.

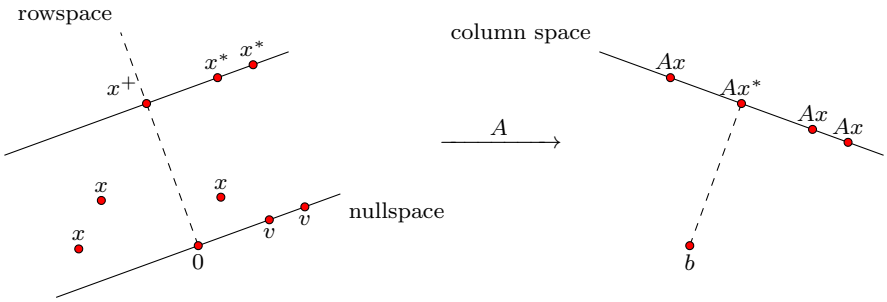


Fig. 2.3 The points x, Ax, the points x^*, Ax^*, and the point x^+.

Key concepts in this section are the *residual*

$$|Ax - b|^2 \tag{2.6.1}$$

and the *regression equation*

$$A^t Ax = A^t b. \tag{2.6.2}$$

Then

Zero Residual

x is a solution of the linear system iff the residual (2.6.1) is zero.



The results in this section are as follows. Let A be any matrix. There is a unique matrix A^+ — the *pseudo-inverse of* A — with the following properties.

- The linear system is solvable when $b = AA^+b$.
- The regression equation is always solvable.
- $x^+ = A^+b$ is a solution of
  1. the linear system, if it is solvable.
  2. the regression equation, always.
- In either case,
  1. there is exactly one solution x^* with minimum norm.
  2. Among all solutions, x^+ has minimum norm.
  3. Every other solution is $x^* = x^+ + v$ for some v in the nullspace of A.



For A as in (2.3.4) and $b = (-9, -3, 3, 9, 10)$, the linear system is

$$\begin{aligned} x + 6y + 11z &= -9 \\ 2x + 7y + 12z &= -3 \\ 3x + 8y + 13z &= 3 \\ 4x + 9y + 14z &= 9 \\ 5x + 10y + 15z &= 10 \end{aligned} \tag{2.6.3}$$

and the regression equation is

$$\begin{aligned} 11x + 26y + 41z &= 16 \\ 13x + 33y + 53z &= 13 \\ 41x + 106y + 171z &= 36. \end{aligned} \tag{2.6.4}$$



Let b be any vector, not necessarily in the column space of A. To see how close we can get to solving the linear system, we minimize the residual (2.6.1). We say x^* is a *residual minimizer* if

$$|Ax^* - b|^2 = \min_x |Ax - b|^2. \tag{2.6.5}$$

A residual minimizer always exists.

Existence of Residual Minimizer

There is a residual minimizer x^* in the row space of A.

The derivation of this technical result is in §4.3, see (4.3.10), (4.3.11).

Regression Equation

x^* is a residual minimizer iff x^* solves the regression equation.

To see this, let v be any vector, and t a scalar. Insert $x = x^* + tv$ into the residual and expand in powers of t to obtain

$$|Ax - b|^2 = |Ax^* - b|^2 + 2t(Ax^* - b) \cdot Av + t^2|Av|^2 = f(t).$$

If x^* is a residual minimizer, then $f(t)$ is minimized when $t = 0$. But a parabola

$$f(t) = \alpha + 2\beta t + \gamma t^2$$

is minimized at $t = 0$ only when $\beta = 0$. Thus the linear coefficient vanishes, $\beta = (Ax^* - b) \cdot Av = 0$. This implies

$$A^t(Ax^* - b) \cdot v = (Ax^* - b) \cdot Av = 0.$$

Since v is any vector, this implies

$$A^t(Ax^* - b) = 0,$$

which is the regression equation. Conversely, if the regression equation holds, then the linear coefficient in the parabola $f(t)$ vanishes, so $t = 0$ is a minimum, establishing that x^* is a residual minimizer.



If x_1 and x_2 are solutions of the regression equation, then

$$A^tA(x_1 - x_2) = A^tAx_1 - A^tAx_2 = A^tb - A^tb = 0,$$

so $x_1 - x_2$ is in the nullspace of A^tA. But from §2.4, the nullspace of A^tA equals the nullspace of A. We conclude $x_1 - x_2$ is in the nullspace of A. This establishes

Multiple Solutions

Any two residual minimizers differ by a vector in the nullspace of A.



We say x^+ is a *minimum norm* residual minimizer if x^+ is a residual minimizer and

$$|x^+|^2 \leq |x^*|^2$$

for any residual minimizer x^*.

Since any two residual minimizers differ by a vector in the nullspace of A, x^+ is a minimum norm residual minimizer if x^+ is a residual minimizer and

$$|x^+|^2 \leq |x^+ + v|^2$$

for any v in the nullspace of A.

Minimum Norm Residual Minimizer

Let x^* be a residual minimizer. Then x^* is a minimum norm residual minimizer iff x^* is in the row space of A.

Since we know from above there is a residual minimizer in the row space of A, we always have a minimum norm residual minimizer.

Let v be in the nullspace of A, and write

$$|x^* + v|^2 = |x^*|^2 + 2x^* \cdot v + |v|^2.$$

This shows x^* is a minimum norm solution of the regression equation iff

$$2x^* \cdot v + |v|^2 \geq 0. \tag{2.6.6}$$

If x^* is in the row space of A, then $x^* \cdot v = 0$, so (2.6.6) is valid.

Conversely, if (2.6.6) is valid for every v in the nullspace of A, replacing v by tv yields

$$2tx^* \cdot v + t^2|v|^2 \geq 0.$$

Dividing by t and inserting $t = 0$ yields

$$x^* \cdot v \geq 0.$$

Since both $\pm v$ are in the nullspace of A, this implies $\pm x^* \cdot v \geq 0$, hence $x^* \cdot v = 0$. Since the row space is the orthogonal complement of the nullspace, the result follows.



Now we use this to show

Uniqueness

There is exactly one minimum norm residual minimizer x^+.

If x_1^+ and x_2^+ are minimum norm residual minimizers, then $v = x_1^+ - x_2^+$ is both in the row space and in the nullspace of A, so $x_1^+ - x_2^+ = 0$. Hence $x_1^+ = x_2^+$.

Putting the above all together, each vector b leads to a unique x^+. Defining A^+ by setting

$$x^+ = A^+ b,$$

we obtain A^+, the *pseudo-inverse* of A.

Notice if A is, for example, 5×4, then $Ax = b$ implies x is a 4-vector and b is a 5-vector. Then from $x = A^+ b$, it follows A^+ is 4×5. Thus *the shape of* A^+ *equals the shape of* A^t.

Summarizing what we have so far,

Regression Equation is Always Solvable

The regression equation is always solvable. The solution of minimum norm is $x^+ = A^+ b$. Any other solution differs by a vector in the nullspace of A.

For A as in (2.3.4) and $b = (-9, -3, 3, 9, 10)$,

$$x^+ = A^+ b = \frac{1}{15} \begin{pmatrix} 82 \\ 25 \\ -32 \end{pmatrix}$$

is the minimum norm solution of the regression equation (2.6.4).



Returning to the linear system $Ax = b$, we show

Linear System Versus Regression Equation

If the linear system is solvable, then every solution of the regression equation is a solution of the linear system, and vice-versa.

We know any two solutions of the linear system differ by a vector in the nullspace of A (2.4.11), and any two solutions of the regression equation (2.6.2) differ by a vector in the nullspace of A (above).

If x is a solution of the linear system, then, by multiplying the linear system by A^t, x is a solution of the regression equation (2.6.2). Since $x^+ = A^+ b$ is

a solution of the regression equation, $x^+ = x + v$ for some v in the nullspace of A, so

$$Ax^+ = A(x + v) = Ax + Av = b + 0 = b.$$

This shows x^+ is a solution of the linear system. Since all other solutions differ by a vector v in the nullspace of A, this establishes the result.

Now we can state when the linear system is solvable,

Solvability of $Ax = b$

The linear system is solvable iff $b = AA^+b$. When this happens, $x^+ = A^+b$ is the solution of minimum norm.

If the linear system is solvable, then from above, x^+ is a solution, so

$$AA^+b = A(A^+b) = Ax^+ = b.$$

Conversely, if $AA^+b = b$, then clearly $x^+ = A^+b$ is a solution of the linear system.

When the linear system is solvable, the linear system and the regression equation have the same solutions, so x^+ is the minimum norm solution of the linear system.

For example, let $b = (-9, -3, 3, 9, 10)$, and let A be as in (2.3.4). Since

$$AA^+b = \begin{pmatrix} -8 \\ -3 \\ 2 \\ 7 \\ 12 \end{pmatrix} \tag{2.6.7}$$

is not equal to b, the linear system $Ax = b$ is not solvable.



Suppose A is invertible. Then the linear system has only the solution $x = A^{-1}b$, so $A^{-1}b$ is the minimum norm residual minimizer. We conclude

Inverse Equals Pseudo-Inverse

If A is invertible, then $A^+ = A^{-1}$.



The key properties [25] of A^+ are

Properties of Pseudo-Inverse

The pseudo-inverse of A is the unique matrix A^+ satisfying

$$\begin{aligned} &\mathbf{A}.\ AA^+A = A \\ &\mathbf{B}.\ A^+AA^+ = A^+ \\ &\mathbf{C}.\ AA^+ \text{ is symmetric} \\ &\mathbf{D}.\ A^+A \text{ is symmetric} \end{aligned} \tag{2.6.8}$$

The verification of these properties is very enlightening, so we do it carefully. Let u be a vector and set $b = Au$. Then the residual

$$|Ax - b|^2 = |Ax - Au|^2$$

is minimized at $x = u$. Since $A^+b = A^+Au$ is the minimum norm residual minimizer, u and A^+Au differ by a vector v in the nullspace of A,

$$u = A^+Au + v. \tag{2.6.9}$$

Since $Av = 0$, multiplying both sides by A leads to

$$Au = AA^+Au.$$

Since u was any vector, this yields **A.**

Now let w be a vector and set $u = A^+w$. Inserting into (2.6.9) yields

$$A^+w = A^+AA^+w + v$$

for some v in the nullspace of A. But both A^+w and A^+AA^+w are in the row space of A, hence so is v. Since v is in both the nullspace and the row space, v is orthogonal to itself, so $v = 0$. This implies $A^+AA^+w = A^+w$. Since w was any vector, we obtain **B.**

Since A^+b solves the regression equation, $A^tAA^+b = A^tb$ for any vector b. Hence $A^tAA^+ = A^t$. With $P = AA^+$,

$$P^tP = (AA^+)^t(AA^+) = (A^+)^tA^tAA^+ = (A^+)^tA^t = P^t.$$

Since the left side is symmetric, so is P^t. Hence P is symmetric, obtaining **C.**

For any vector x,

$$A(x - A^+Ax) = Ax - AA^+Ax = 0,$$

so $x - A^+Ax$ is in the nullspace of A. For any y, A^+Ay is in the row space of A. Since the row space and the nullspace are orthogonal,

$$(x - A^+Ax) \cdot A^+Ay = 0.$$

Let $P = A^+A$. This implies

$$x \cdot Py = Px \cdot Py = x \cdot P^tPy$$

Since this is true for any vectors x and y, $P = P^tP$. This shows $P = A^+A$ is symmetric, obtaining **D.**

Having arrived at **A**, **B**, **C**, **D**, the reasoning is reversible: It can be shown any matrix A^+ satisfying **A**, **B**, **C**, **D** must equal the pseudo-inverse (see Exercise **2.6.17**).



An immediate consequence of (2.6.8) is

Pseudo-Inverse of Pseudo-Inverse

$(A^+)^+ = A$ for any matrix A.

Also we have

Pseudo-Inverse and Transpose

If U has orthonormal columns or orthonormal rows, then $U^+ = U^t$.

From (2.2.9), such a matrix U satisfies $UU^t = I$ or $U^tU = I$. In either case, **A**, **B**, **C**, **D** are immediate consequences.



In general, $(AB)^+ = B^+A^+$ is not correct. However, in special cases, it is correct.

Exercises

Exercise 2.6.1 Let A be the 1×3 matrix $(1, 2, 3)$. What is A^+?

Exercise 2.6.2 Let $A(N, d)$ be as in Exercise **2.4.9**, and let $A = A(6, 4)$. Let $b = (1, 1, 1, 1, 1, 1)$. Write out $Ax = b$ as a linear system. How many equations, how many unknowns?

Exercise 2.6.3 With A and b as in Exercise **2.6.2**, is $Ax = b$ solvable? If so, provide a solution.

Exercise 2.6.4 Continuing with the same A and b, write out the corresponding regression equation. How many equations, how many unknowns?

Exercise 2.6.5 With A and b as in Exercise **2.6.2**, is the regression equation solvable? If so, provide a solution.

Exercise 2.6.6 With A and b as in Exercise **2.6.2**, what is the minimum norm residual minimizer x^+?

Exercise 2.6.7 Let μ be a unit vector, and let $Q = I - \mu \otimes \mu$. Use (2.6.8) and Exercise **2.2.3** to show $Q^+ = Q$.

Exercise 2.6.8 Use (2.6.8) to show the transpose of the pseudo-inverse is the pseudo-inverse of the transpose,

$$(A^t)^+ = (A^+)^t.$$

Exercise 2.6.9 Let Q be symmetric. Show Q^+ is symmetric.

Exercise 2.6.10 Let Q be symmetric. Show Q and Q^+ commute,

$$QQ^+ = Q^+Q.$$

Exercise 2.6.11 Let A be any matrix. Then the nullspace of A equals the nullspace of A^+A. Use (2.6.8).

Exercise 2.6.12 Let A be any matrix. Then the row space of A equals the row space of A^+A.

Exercise 2.6.13 Let A be any matrix. Then the column space of A equals the column space of AA^+.

Exercise 2.6.14 Let A be any matrix and $Q = A^tA$. Then $Q^+ = A^+(A^+)^t$.

Exercise 2.6.15 Let A be a matrix with $Q = A^tA$ invertible. Show $A^+ = Q^{-1}A^t$ and $A^+A = I$. In this case, A^+ is called a *left inverse*.

Exercise 2.6.16 Let A be a matrix with $Q = AA^t$ invertible. Show $A^+ = A^tQ^{-1}$ and $AA^+ = I$. In this case, A^+ is called a *right inverse*.

Exercise 2.6.17 Suppose A^+ is any matrix satisfying (2.6.8). Let b be any vector, and let $x = A^+b$. Show x satisfies the regression equation and x is orthogonal to the null space of A. Conclude A^+ is the pseudo-inverse of A.

2.7 Projections

Let A be any matrix and let A^+ be its pseudo-inverse. In this section, we study projection matrices P, and we show

- $P = AA^+$ is the projection matrix onto the column space of A,
- $P = A^+A$ is the projection matrix onto the row space of A,
- $P = I - A^+A$ is the projection matrix onto the null space of A,



Let u be a *unit* vector, and let b be any vector. Let $span(u)$ be the line through u (Figure 2.4). The *projection* of b onto $span(u)$ is the vector v in $span(u)$ that is closest to b.

It turns out this closest vector v equals Pb for some matrix P, the *projection matrix*. Since $span(u)$ is a line, the projected vector Pb is a multiple tu of u.

From Figure 2.4, $b - Pb$ is orthogonal to u, so

$$0 = (b - Pb) \cdot u = b \cdot u - Pb \cdot u = b \cdot u - t\, u \cdot u = b \cdot u - t.$$

Solving for t, this implies $t = b \cdot u$. Thus

$$Pb = (b \cdot u)u = (u \otimes u)b. \tag{2.7.1}$$

From here, we conclude $P = u \otimes u$. Equivalently, if U is the matrix with the single column u, we obtain $P = UU^t$.

Notice $Pb = b$ when b is already on the line through u. In other words, *the projection of a vector onto a line equals the vector itself when the vector is already on the line.*

To summarize, the *projected vector* is the vector $(b \cdot u)u$, and the *reduced vector* is the scalar $b \cdot u$. If U is the matrix with the single column u, then the reduced vector is U^tb and the projected vector is UU^tb.

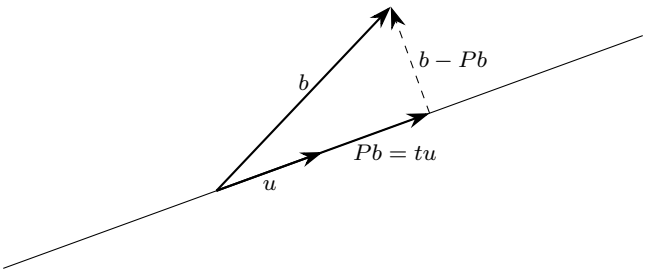


Fig. 2.4 Projecting onto a line.



Now we project onto a plane. Let u, v be an *orthonormal* pair of vectors, so $u \cdot v = 0$, $u \cdot u = 1 = v \cdot v$. We project a vector b onto $span(u, v)$. As before, there is a matrix P, the *projection matrix*, such that the projection of b onto the plane equals Pb. Then $b - Pb$ is orthogonal to the plane (Figure 2.5), which means $b - Pb$ satisfies

$$(b - Pb) \cdot u = 0 \qquad \text{and} \qquad (b - Pb) \cdot v = 0.$$

Since Pb lies in the plane, $Pb = ru + sv$ is a linear combination of u and v. Inserting $Pb = ru + sv$, we obtain

$$r = b \cdot u, \qquad s = b \cdot v.$$

If U is the matrix with columns u, v, by (2.2.11) and (2.2.14), this yields,

$$Pb = (b \cdot u)u + (b \cdot v)v = (u \otimes u + v \otimes v)b = UU^t b,$$

Hence the projection matrix is $P = UU^t$.

Notice $Pb = b$ when b is already in the plane. In other words, *the projection of a vector onto a plane equals the vector itself when the vector is already in the plane.*

To summarize, here the *projected vector* is the vector $UU^t b$, and the *reduced vector* is the vector $U^t b$. The projected vector has the same dimension as the original vector, and the reduced vector is in $\mathbf{R}^2$.

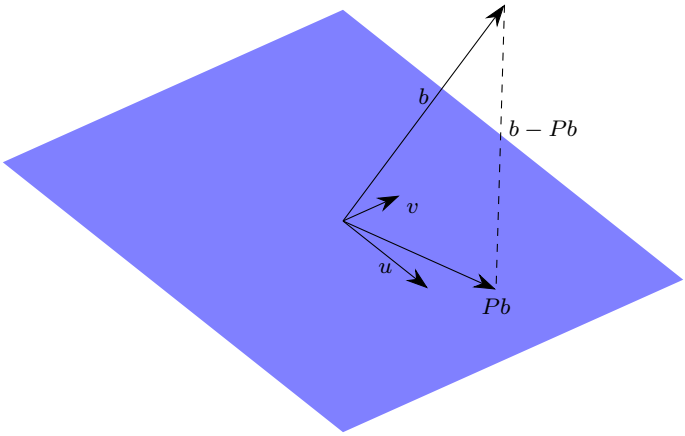


Fig. 2.5 Projecting onto a plane, $Pb = ru + sv$.

We define projection matrices in general. Let S be a span. A matrix P is the *projection matrix onto* S if

1. Pv is in S for any vector v,
2. $Pv = v$ if v is in S,
3. $v - Pv$ is orthogonal to S for any vector v.

We say *the* projection matrix onto S because there is only one such matrix corresponding to a given S, see Exercise **2.7.11**.

Here is a characterization without mentioning S. A matrix P is a *projection matrix* if

1. $P^2 = P$,
2. $P^t = P$.

What is the relation between these two versions? We show they are the same.

Characterization of Projections

If P is the projection matrix onto a span S, then P is a projection matrix. Conversely, if P is a projection matrix, then P is the projection matrix onto the column space S of P.

To prove this, suppose P is the projection matrix onto some span S. For any v, by 1., Pv is in S. By 2., $P(Pv) = Pv$. Hence $P^2 = P$. Also, for any u and v, Pv is in S, and $u - Pu$ is orthogonal to S. Hence

$$(u - Pu) \cdot Pv = 0$$

which implies

$$u \cdot Pv = (Pu) \cdot (Pv).$$

Switching u and v,

$$v \cdot Pu = (Pv) \cdot (Pu),$$

Hence

$$u \cdot (Pv) = (Pu) \cdot v,$$

which implies $P = P^t$.

For the other direction, suppose P is a projection matrix, and let S be the column space of P. Then a vector x is in S iff x is of the form $x = Pv$. This establishes 1. above. Since

$$Px = P(Pv) = P^2 v = Pv = x,$$

this establishes 2. above. Similarly, $P^t = P$ implies 3. above.



Projection Onto Column Space

Let A be any matrix. Then the projection matrix onto the column space of A is

$$P = AA^+. \tag{2.7.2}$$

To see this, let $P = AA^+$. By (2.6.8),

$$P^2 = AA^+AA^+ = (AA^+A)A^+ = AA^+ = P,$$

and P is symmetric. Hence P is a projection matrix. By the previous result, P is the projection matrix onto the column space of $P = AA^+$. But by Exercise **2.6.13**, the column spaces of A and of P agree. Thus P is the projection matrix onto the column space of A.



Now let $x = A^+b$. Then $Ax = AA^+b = Pb$ is the projection of b onto the column space of A. If the columns of A are v_1, v_2, ..., v_d, and $x = (t_1, t_2, \ldots, t_d)$, then by matrix-vector multiplication,

$$Pb = t_1v_1 + t_2v_2 + \cdots + t_dv_d.$$

Since the reduced vector x consists of the coefficients when writing Pb as a linear combination of the columns of A, this shows A^+b is the reduced vector.

```
from numpy import *
set_printoptions(legacy = "1.25")

from scipy.linalg import pinv

# projection of column vector b
# onto column space of A

# assume len(b) == len(A.T)

def project(A,b):
   Aplus = pinv(A)
   x = dot(Aplus,b)     # reduced
   return dot(A,x)      # projected
```

Projected and Reduced Vectors

Let A be a matrix and b a vector, and project onto the column space of A. Then the projected vector is $Pb = AA^+b$ and the reduced vector is $x = A^+b$.

For A as in (2.3.4) and $b = (-9, -3, 3, 9, 10)$ the reduced vector onto the column space of A is

$$x = A^+ b = \frac{1}{15}(82, 25, -32),$$

and the projected vector onto the column space of A is

$$Pb = Ax = AA^+ b = (-8, -3, 2, 7, 12).$$

The projection matrix onto the column space of A is

$$P = AA^+ = \frac{1}{10}\begin{pmatrix} 6 & 4 & 2 & 0 & -2 \\ 4 & 3 & 2 & 1 & 0 \\ 2 & 2 & 2 & 2 & 2 \\ 0 & 1 & 2 & 3 & 4 \\ -2 & 0 & 2 & 4 & 6 \end{pmatrix}.$$



In the same way, one can show

Projection Onto Row Space

The projection matrix onto the row space of A is

$$P = A^+ A. \tag{2.7.3}$$

For A as in (2.3.4), the projection matrix onto the row space is

$$P = A^+ A = \frac{1}{6}\begin{pmatrix} 5 & 2 & -1 \\ 2 & 2 & 2 \\ -1 & 2 & 5 \end{pmatrix}$$



When the columns of a matrix U are orthonormal, in the previous section we saw $U^+ = U^t$, so we have

Projection onto Orthonormal Vectors

If the columns of U are orthonormal, the projection matrix onto the column space of U is

$$P = UU^t \tag{2.7.4}$$

Here the projected vector is UU^tb, and the reduced vector is U^tb. The code here is

```
# projection of column vector b
# onto column space of U
# with orthonormal columns

# assume len(b) == len(U)

def project_to_ortho(U,b):
   x = dot(U.T,b)       # reduced
   return dot(U,x)      # projected
```



Let $v_1, v_2, \ldots, v_N$ be a dataset in $\mathbf{R}^d$, and let U be a $d \times n$ matrix with orthonormal columns. Then the projection matrix onto the column space of U is $P = UU^t$, and P is the projection onto an orthonormal span.

In this case, the dataset $U^tv_1, U^tv_2, \ldots, U^tv_N$ is the *reduced dataset*, and $UU^tv_1, UU^tv_2, \ldots, UU^tv_N$ is the *projected dataset*.

The projected dataset is in $\mathbf{R}^d$, and the reduced dataset is in $\mathbf{R}^n$. Table 2.1 summarizes the relationships.

dataset	v_k in $\mathbf{R}^d$, $k = 1, 2, \ldots, N$
reduced	U^tv_k in $\mathbf{R}^n$, $k = 1, 2, \ldots, N$
projected	UU^tv_k in $\mathbf{R}^d$, $k = 1, 2, \ldots, N$

Table 2.1 Dataset, reduced dataset, and projected dataset, $n < d$.

```
from sklearn import datasets

iris = datasets.load_iris()
dataset = iris["data"]

# dataset is Nxd
# A columns are first two vectors
A = dataset[:2].T

Aplus = pinv(A) # any matrix

reduced = array([ dot(Aplus,x) for x in dataset ])
projected = array([ dot(A,x) for x in reduced ])

reduced.shape, projected.shape
```



Let S and T be spans. Let $S+T$ consist of all sums of vectors $u+v$ with u in S and v in T. Then a moment's thought shows $S+T$ is itself a span. When the intersection of S and T is the zero vector, we write $S \oplus T$, and we say $S \oplus T$ is the *direct sum* of S and T.

Let S be a span and let $S^\perp$ consist of all vectors orthogonal to S (see (2.4.9)). We call $S^\perp$ the *orthogonal complement* of S. This is pronounced "S-perp". If v is in both S and in $S^\perp$, then v is orthogonal to itself, hence $v=0$. From this, we see $S+S^\perp$ is a direct sum $S \oplus S^\perp$.

Direct Sum and Orthogonal Complement I

If S is a span in $\mathbf{R}^d$, then

$$\mathbf{R}^d = S \oplus S^\perp. \tag{2.7.5}$$

This is an immediate consequence of what we already know. Let P be the projection matrix onto S. Since any vector v in $\mathbf{R}^d$ may be written

$$v = Pv + (v - Pv),$$

we see any vector is a sum of a vector in S and a vector in $S^\perp$.

Let S be the span of a dataset $x_1, x_2, \ldots, x_N$. If S does not equal $\mathbf{R}^d$, then there is a nonzero vector in $S^\perp$. This shows

Direct Sum and Orthogonal Complement II

If a dataset spans $\mathbf{R}^d$ and v is orthogonal to the dataset, then $v=0$. If v is not zero and is orthogonal to the dataset, then the dataset does not span $\mathbf{R}^d$.

Another way of saying the same thing: A vector v is orthogonal to the whole space iff v is zero.



An important example of (2.7.5) is the relation between the row space and the nullspace of a matrix. In §2.4, we saw that, for any matrix A, the row space and the nullspace are orthogonal complements.

Taking $S =$ nullspace in (2.7.5), we have the important

Null space plus Row Space Equals Source Space

If A is an $N \times d$ matrix,

$$\text{nullspace} \oplus \text{rowspace} = \mathbf{R}^d, \tag{2.7.6}$$

and the nullspace and row space are orthogonal to each other.

From this,

Projection Onto Null Space

The projection matrix onto the nullspace of A is

$$P = I - A^+ A. \tag{2.7.7}$$

For A as in (2.3.4), the projection matrix onto the nullspace is

$$P = I - A^+ A = \frac{1}{6} \begin{pmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{pmatrix}$$

Here $P = u \otimes u$, where $u = (1, -2, 1)/\sqrt{6}$, in agreement with (2.7.1).

The result (2.7.6) can be written as

Row Rank plus Nullity equals Source Space Dimension

For any matrix, the row rank plus the nullity equals the dimension of the source space. If the matrix is $N \times d$, r is the rank, and n is the nullity, then

$$r + n = d.$$



Let S be the column space of a matrix A, and let P be the projection matrix onto S. We end the section by establishing the claim made at the start of the section, that Pb is the point in S that is closest to b.

Since every point in S is of the form Ax, we need to check

$$|Pb - b|^2 = \min_x |Ax - b|^2.$$

But this was already done in §2.3, since $Pb = AA^+ b = Ax^+$ where $x^+ = A^+ b$ is a residual minimizer.

Projection is the Nearest Point in the Span

Let $Pb = AA^+b$ be the projection of b onto the column space of A, and let $x^+ = A^+b$ be the reduced vector. Then

$$|Ax^+ - b|^2 = \min_x |Ax - b|^2. \tag{2.7.8}$$

Exercises

Exercise 2.7.1 Let A be a 7×12 matrix. What is the greatest the rank of A can be? What is the least the rank of A can be? What if A is 12×7?

Exercise 2.7.2 Let A be a 7×12 matrix. What is the greatest the nullity of A can be? What is the least the nullity of A can be? What if A is 12×7?

Exercise 2.7.3 Let A be a matrix and let $u_1, u_2, \ldots, u_r$ be an orthonormal basis for the column space of A. Show that the projection onto the column space of A is

$$P = u_1 \otimes u_1 + u_2 \otimes u_2 + \cdots + u_r \otimes u_r.$$

Exercise 2.7.4 Let P be the projection matrix onto the column space of a matrix A. Use Exercise **2.7.3** to show trace(P) equals the rank of A.

Exercise 2.7.5 Let A be a 10×7 matrix and let $Q = A^tA$. Then Q is 7×7. If the row rank of A is 5, what is the row rank of Q?

Exercise 2.7.6 Let A be the dataset matrix of the centered MNIST dataset, so the shape of A is 60000×784. Using Exercise **2.7.4** and `sympy`, compute the rank of A.

Exercise 2.7.7 If μ is a unit vector, then $P = I - \mu \otimes \mu$ is a projection.

Exercise 2.7.8 If μ and ν are orthogonal unit vectors, then $P = I - \mu \otimes \mu - \nu \otimes \nu$ is a projection.

Exercise 2.7.9 Let S be a span, and let P be the projection matrix onto S. Use P to show

$$\left(S^\perp\right)^\perp = S. \tag{2.7.9}$$

($S \subset (S^\perp)^\perp$ is easy. For $S \supset (S^\perp)^\perp$, recall $v - Pv$ is in $S^\perp$ for any v. Use this to show $|v - Pv|^2 = 0$ when v in $(S^\perp)^\perp$.)

Exercise 2.7.10 Let m_1 and m_2 be nonzero vectors satisfying $m_1^\perp = m_2^\perp$ (2.4.9). Show m_1 is in $\left(m_2^\perp\right)^\perp$. Use the previous exercise with $S = span(m_2)$ to conclude m_1 and m_2 are parallel.

Exercise 2.7.11 Let S be a span and suppose P and Q are both projection matrices onto S. Show

$$P = Q.$$

Exercise 2.7.12 Let A be any matrix. Then the nullspace and row space of A equal the nullspace and row space of $(A^+)^t$. (Write out the projections onto the row spaces.)

2.8 Basis

Let S be the span of vectors v_1, v_2, ..., v_N. Then there are many other choices of spanning vectors for S. For example, $v_1 + v_2$, v_2, v_3, ..., v_N also spans S.

If S cannot be spanned by fewer than N vectors, then we say v_1, v_2, ..., v_N. is a *basis* for S, and we call N is the *dimension* of S.

In other words, when N is the smallest number of spanning vectors, we say N is the dimension $\dim S$ of S, and v_1, v_2, ..., v_N is a *minimal spanning set* for S. This definition is important enough to repeat,

Basis and Dimension Definition

A basis for a span S is a minimal spanning set of vectors. The dimension of S is the number of vectors in *any* basis for S.

A span has many choices for minimal spanning sets, but a span's dimension, as the number of vectors in any minimal spanning set, is uniquely determined.

Here are two immediate consequences of this terminology.

Span of N Vectors

If $S = span(v_1, v_2, \ldots, v_N)$, then $\dim S \leq N$.

Larger Span has Larger Dimension

If a span S_1 is contained in a span S_2, then $\dim S_1 \leq \dim S_2$.

With this terminology,

- `rowspace()` returns a basis of the row space,
- `columnspace()` returns a basis of the column space,
- `nullspace()` returns a basis for the nullspace,
- row rank equals the dimension of the row space,
- column rank equals the dimension of the column space,

- nullity equals the dimension of the nullspace.



Let S be the span of vectors $v_1, v_2, \ldots, v_N$. How can we check if these vectors constitute a basis for S? The answer is the main result of the section.

Spanning Plus Linearly Independent Equals Basis

Let S be the span of vectors $v_1, v_2, \ldots, v_N$. Then the vectors are a basis for S iff they are linearly independent.

Remember, to check for linear independence of given vectors, assemble the vectors as columns of a matrix A, and check whether `A.nullspace()` equals zero. If that is the case, the vectors are a basis for their span. If not, the vectors are not a basis for their span. The proof of the main result is at the end of the section.



When a basis $v_1, v_2, \ldots, v_N$ consists of orthogonal vectors, we say $v_1, v_2, \ldots, v_N$ is an *orthogonal basis*. When $v_1, v_2, \ldots, v_N$ are also unit vectors, we say $v_1, v_2, \ldots, v_N$ is an *orthonormal basis*.

As we saw in §2.4, orthonormal vectors $v_1, v_2, \ldots, v_N$ are linearly independent, so, by the main result, $v_1, v_2, \ldots, v_N$ are an orthonormal basis for their span.

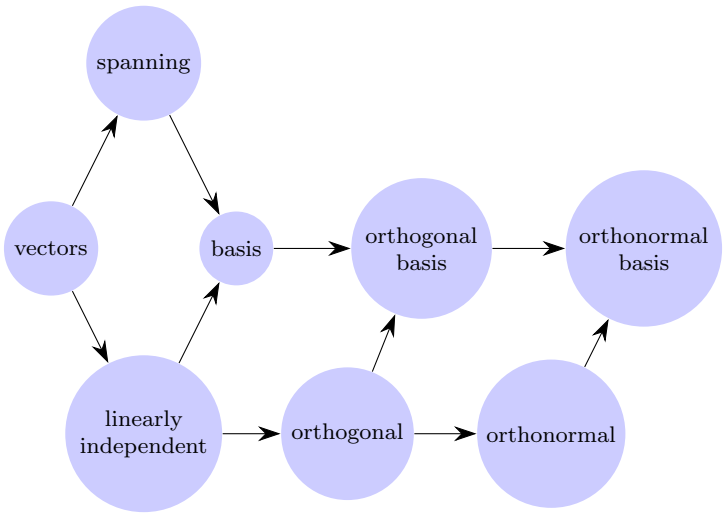


Fig. 2.6 Relations between vector classes.



Here is an example. The columns of the 3×3 identity matrix I are $e_1 = (1, 0, 0)$, $e_2 = (0, 1, 0)$, $e_3 = (0, 0, 1)$. Since the nullspace of I is zero, e_1, e_2, e_3 are linearly independent. Hence the standard basis e_1, e_2, e_3 is indeed a basis for $\mathbf{R}^3$, i.e. a minimal spanning set of vectors for $\mathbf{R}^3$. From this, we conclude $\dim \mathbf{R}^3 = 3$.

The statement $\dim \mathbf{R}^3 = 3$ may at first seem trivial or obvious. But, if we flesh this out following our terminology above, the statement is saying that any minimal spanning set of vectors in $\mathbf{R}^3$ must have exactly 3 vectors. Stated in this manner, the statement has content.

Since we can do the same calculation with the standard basis

$$\begin{aligned} e_1 &= (1, 0, \ldots, 0), \\ e_2 &= (0, 1, 0, \ldots, 0), \\ \ldots &= \ldots \\ e_d &= (0, 0, \ldots, 0, 1), \end{aligned}$$

in $\mathbf{R}^d$, we conclude e_1, e_2, $\ldots$, e_d are linearly independent, so

Dimension of Euclidean Space

The dimension of $\mathbf{R}^d$ is d.



The MNIST dataset consists of vectors v_1, v_2, $\ldots$, v_N in $\mathbf{R}^d$, where $N = 60000$ and $d = 784$. For the MNIST dataset, the dimension is 712, as returned by the code

```
from sklearn import datasets

iris = datasets.load_iris()
dataset = iris["data"]

from numpy import *
from numpy.linalg import matrix_rank
set_printoptions(legacy = "1.25")

# dataset is Nxd array

mu = mean(dataset,axis = 0)
vectors = dataset - mu

matrix_rank(vectors)
```

In particular, since $712 < 784$, approximately 10% of pixels are never touched by any image. For example, a likely pixel to remain untouched is at the top left corner $(0, 0)$. For this dataset, there are $784 - 712 = 72$ zero variance directions.

We pose the following question: What is the least n for which the first n images are linearly dependent? Since the dimension of the sample space is 784, we must have $n \leq 784$. To answer the question, we compute the rank of the first n vectors for $n = 1, 2, 3, \ldots$, and continue until we have linear dependence of v_1, v_2, $\ldots$, v_n.

If we load MNIST as `dataset`, as in §1.2, and run the code below, we obtain $n = 560$ (Figure 2.7). `matrix_rank` is discussed in §2.9.

```
from numpy import *
from numpy.linalg import matrix_rank

# dataset is Nxd array

def find_first_defect(dataset):
   d = len(dataset[0])
   previous = 0
   for n in range(len(dataset)):
      r = matrix_rank(dataset[:n+1,:])
      print((r,n+1),end=",")
      if r == previous: break
      if r == d: break
      previous = r
```

```
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,4
2,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,
81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,11
4,115,116,117,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,14
3,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,164,165,166,167,168,169,170,171,17
2,173,174,175,176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,191,192,193,194,195,196,197,198,199,200,20
1,202,203,204,205,206,207,208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,225,226,227,228,229,23
0,231,232,233,234,235,236,237,238,239,240,241,242,243,244,245,246,247,248,249,250,251,252,253,254,255,256,257,258,25
9,260,261,262,263,264,265,266,267,268,269,270,271,272,273,274,275,276,277,278,279,280,281,282,283,284,285,286,287,28
8,289,290,291,292,293,294,295,296,297,298,299,300,301,302,303,304,305,306,307,308,309,310,311,312,313,314,315,316,31
7,318,319,320,321,322,323,324,325,326,327,328,329,330,331,332,333,334,335,336,337,338,339,340,341,342,343,344,345,34
6,347,348,349,350,351,352,353,354,355,356,357,358,359,360,361,362,363,364,365,366,367,368,369,370,371,372,373,374,37
5,376,377,378,379,380,381,382,383,384,385,386,387,388,389,390,391,392,393,394,395,396,397,398,399,400,401,402,403,40
4,405,406,407,408,409,410,411,412,413,414,415,416,417,418,419,420,421,422,423,424,425,426,427,428,429,430,431,432,43
3,434,435,436,437,438,439,440,441,442,443,444,445,446,447,448,449,450,451,452,453,454,455,456,457,458,459,460,461,46
2,463,464,465,466,467,468,469,470,471,472,473,474,475,476,477,478,479,480,481,482,483,484,485,486,487,488,489,490,49
1,492,493,494,495,496,497,498,499,500,501,502,503,504,505,506,507,508,509,510,511,512,513,514,515,516,517,518,519,52
0,521,522,523,524,525,526,527,528,529,530,531,532,533,534,535,536,537,538,539,540,541,542,543,544,545,546,547,548,54
9,550,551,552,553,554,555,556,557,558,559,560,560,
```

Fig. 2.7 First defect for MNIST.



Let v_1, v_2, $\ldots$, v_N be a dataset. We want to compute the dimensions of the first n vectors, $n = 1, 2, 3, \ldots$,

$$d_1 = \dim(v_1), \qquad d_2 = \dim(v_1, v_2), \qquad d_3 = \dim(v_1, v_2, v_3), \qquad \text{and so on}$$

This we call the *dimension staircase*. For example, Figure 2.8 is the dimension staircase for

$$v_1 = (1,0,0), v_2 = (0,1,0), v_3 = (1,1,0), v_4 = (3,4,0), v_5 = (0,0,1).$$

In Figure 2.8, we call the points $(3,2)$ and $(4,2)$ *defects*.

In the code, the staircase is drawn by `stairs(X,Y)`, where the horizontal points `X` and the vertical values `Y` satisfy `len(X) == len(Y)+1`. In Figure 2.8, `X = [1,2,3,4,5,6]`, and `Y = [1,2,2,2,3]`.

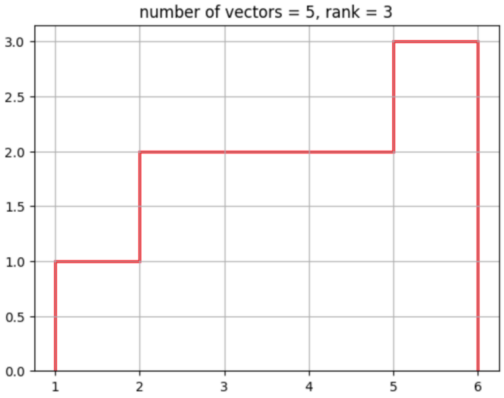


Fig. 2.8 The dimension staircase with defects.

With the MNIST dataset loaded as `vectors`, here is code returning Figures 2.8 and 2.9. This code is not efficient, but it works. Ideally the code should be run in sympy using exact arithmetic. However, this takes too long, so we use `numpy.linalg.matrix_rank`. Because datasets consist of floats in `numpy`, the `matrix_rank` and dimensions are approximate not exact. For more on this, see *approximate rank* in §3.2.

```
from numpy import *
from matplotlib.pyplot import *
from numpy.linalg import matrix_rank

# dataset is Nxd array

def dimension_staircase(dataset):
   N = len(dataset)
   rmax = matrix_rank(dataset)
   dimensions = [ ]
   for n in range(N):
      r = matrix_rank(dataset[:n+1,:])
      dimensions.append(r)
```

```
        if r == rmax: break
    title("number of vectors = " + str(n+1) + ", rank = " +
     ↪ str(rmax))
    stairs(dimensions, range(1,n+3),linewidth = 2,color = 'red')
    grid()
    show()

dimension_staircase(dataset)
```

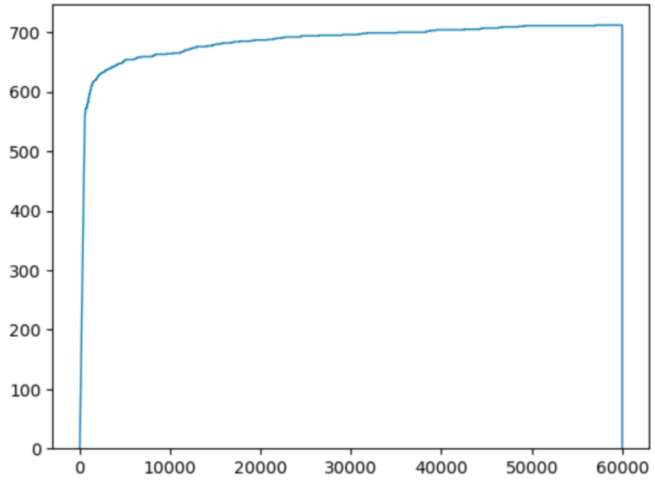


Fig. 2.9 The dimension staircase for the MNIST dataset.



Proof of main result. Here we derive: *Let S be the span of v_1, v_2, ..., v_N. Then v_1, v_2, ..., v_N is a basis for S iff v_1, v_2, ..., v_N are linearly independent.*

Suppose $v_1, v_2, \ldots, v_N$ are not linearly independent. Then $v_1, v_2, \ldots, v_N$ are linearly dependent, which means one of the vectors, say v_1, is a linear combination of the other vectors $v_2, v_3, \ldots, v_N$. Then any linear combination of $v_1, v_2, \ldots, v_N$ is necessarily a linear combination of $v_2, v_3, \ldots, v_N$, thus

$$span(v_1, v_2, \ldots, v_N) = span(v_2, v_3, \ldots, v_N).$$

This shows $v_1, v_2, \ldots, v_N$ is not a minimal spanning set, and completes the derivation in one direction.

In the other direction, suppose $v_1, v_2, \ldots, v_N$ are linearly independent, and suppose $b_1, b_2, \ldots, b_d$ is a minimal spanning set. Since $b_1, b_2, \ldots, b_d$ is

minimal, we must have $d \leq N$. Once we establish $d = N$, it follows v_1, v_2, $\ldots$, v_N is minimal, and the proof will be complete.

Since by assumption,

$$span(v_1, v_2, \ldots, v_N) = span(b_1, b_2, \ldots, b_d),$$

v_1 is a linear combination of b_1, b_2, $\ldots$, b_d,

$$v_1 = t_1 b_1 + t_2 b_2 + \cdots + t_d b_d.$$

Since $v_1 \neq 0$, at least one of the t coefficients is not zero. By rearranging the vectors, assume $t_1 \neq 0$. Then we can solve for b_1,

$$b_1 = \frac{1}{t_1}(v_1 - t_2 b_2 - t_3 b_3 - \cdots - t_d b_d).$$

This shows

$$span(v_1, v_2, \ldots, v_N) = span(v_1, b_2, b_3, \ldots, b_d).$$

Repeating the same logic, v_2 is a linear combination of v_1, b_2, b_3, $\ldots$, b_d,

$$v_2 = s_1 v_1 + t_2 b_2 + t_3 b_3 + \cdots + t_d b_d.$$

If all the coefficients of b_2, b_3, $\ldots$, b_d are zero, then v_2 is a multiple of v_1, contradicting linear independence of v_1, v_2, $\ldots$, v_N. Thus at least one of the t coefficients is not zero. By rearranging the vectors, assume $t_2 \neq 0$. Then we can solve for b_2, obtaining

$$b_2 = \frac{1}{t_2}(v_2 - s_1 v_1 - t_3 b_3 - \cdots - t_d b_d).$$

This shows

$$span(v_1, v_2, \ldots, v_N) = span(v_1, v_2, b_3, b_4, \ldots, b_d).$$

Repeating the same logic, v_3 is a linear combination of v_1, v_2, b_3, b_3, $\ldots$, b_d,

$$v_3 = s_1 v_1 + s_2 v_2 + t_3 b_3 + t_4 b_4 + \cdots + t_d b_d.$$

If all the coefficients of b_3, b_4, $\ldots$, b_d are zero, then v_3 is a linear combination of v_1, v_2, contradicting linear independence of v_1, v_2, $\ldots$, v_N. Thus at least one of the t coefficients is not zero. By rearranging the vectors, assume $t_3 \neq 0$. Then we can solve for b_3, obtaining

$$b_3 = \frac{1}{t_3}(v_3 - s_1 v_1 - s_2 v_2 - t_4 b_4 - \cdots - t_d b_d).$$

This shows

$$span(v_1, v_2, \ldots, v_N) = span(v_1, v_2, v_3, b_4, b_5, \ldots, b_d).$$

Continuing in this manner, we eventually arrive at

$$span(v_1, v_2, \ldots, v_N) = \cdots = span(v_1, v_2, \ldots, v_d).$$

This shows v_N is a linear combination of v_1, v_2, …, v_d. This shows $N = d$, because $N > d$ contradicts linear independence. Since d is the minimal spanning number, this shows v_1, v_2, …, v_N is a minimal spanning set for S.

2.9 Rank

If A is an $N \times d$ matrix, then (Figure 2.10) $x \mapsto Ax$ is a linear transformation that sends a vector x in $\mathbf{R}^d$ to the vector $b = Ax$ in $\mathbf{R}^N$. When this happens, we call $\mathbf{R}^d$ the *source space* and $\mathbf{R}^N$ the *target space*. The transpose A^t goes in the reverse direction: The linear transformation $b \mapsto A^t b$ sends a vector b in the target space $\mathbf{R}^N$ to the vector $A^t b$ in the source space $\mathbf{R}^d$.

It follows that for an $N \times d$ matrix, the dimension of the source space is d, and the dimension of the target space is N,

$$\dim(\text{source space}) = d, \qquad \dim(\text{target space}) = N.$$

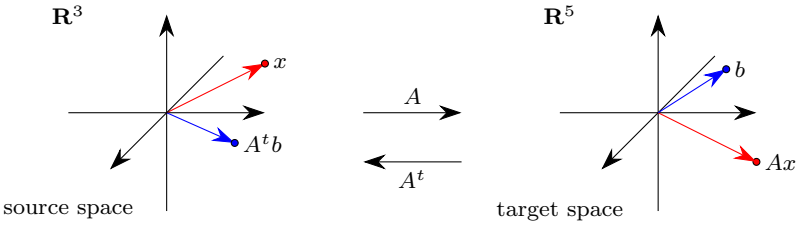


Fig. 2.10 A 5×3 matrix A is a linear transformation from $\mathbf{R}^3$ to $\mathbf{R}^5$.

By (2.4.2), the column space is in the target space, and the row space is in the source space. Thus we always have

$$0 \le \text{row rank} \le d \qquad \text{and} \qquad 0 \le \text{column rank} \le N.$$

For

$$A = \begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix},$$

the column rank is 2, the row rank is 2, and the nullity is 1. Thus the column space is a plane in $\mathbf{R}^5$, the row space is a plane in $\mathbf{R}^3$, and the nullspace is a line in $\mathbf{R}^3$.



The main result in this section is

Rank Theorem

Let A be any matrix. Then

$$\text{row rank}(A) = \text{column rank}(A). \tag{2.9.1}$$

This is established at the end of the section.

Because the row rank and the column rank are equal, below we just say *rank* of a matrix, and we write $rank(A)$. In `sympy`, the rank of A is `A.rank()`. In `numpy.linalg`, the rank of A is `matrix_rank(A)`. The main result implies

Rank of the Transpose

For any matrix A, $rank(A) = rank(A^t)$. In particular, the rank of an $N \times d$ matrix is never greater than $\min(N, d)$.



An $N \times d$ matrix A is *full-rank* if its rank is the highest it can be, $rank(A) = \min(N, d)$. Here are some consequences of the main result.

- When $N \geq d$, full-rank is the same as $rank(A) = d$, which is the same as saying the columns are linearly independent and the rows span $\mathbf{R}^d$.
- When $N \leq d$, full-rank is the same as $rank(A) = N$, which is the same as saying the rows are linearly independent and the columns span $\mathbf{R}^N$.
- When $N = d$, full-rank is the same as saying the rows are a basis of $\mathbf{R}^d$, and the columns are a basis of $\mathbf{R}^N$.

When A is a square matrix, we can say more:

Full Rank Square Equals Invertible

Let A be a *square* matrix. Then A is full-rank iff A is invertible.

Suppose A is $d \times d$. If A is invertible and B is its inverse, then $AB = I$. Since $ABx = A(Bx) = Ay$ with $y = Bx$, the column space of AB is contained in

the column space of A. Since the column space of $AB = I$ is $\mathbf{R}^d$, we conclude the column space of A is $\mathbf{R}^d$, thus $rank(A) = d$.

Conversely, suppose A is full-rank. This means the columns of A span $\mathbf{R}^d$. By (2.4.3), this implies

$$Ax = b$$

is solvable for any b. Let $e_1, e_2, \ldots, e_d$ be the standard basis. If we set successively $b = e_1$, $b = e_2$, $\ldots$, $b = e_d$, we then get solutions $x_1, x_2, \ldots, x_d$. If B is the matrix with columns $x_1, x_2, \ldots, x_d$, then

$$AB = A(x_1, x_2, \ldots, x_d) = (Ax_1, Ax_2, \ldots, Ax_d) = (e_1, e_2, \ldots, e_d) = I.$$

Thus we found a matrix B satisfying $AB = I$.

Repeating the same argument with rows instead of columns, we find a matrix C satisfying $CA = I$. Then

$$C = CI = CAB = IB = B,$$

so $B = C$ is the inverse of A.



Orthonormal Rows and Columns

Let U be a matrix.

- U has orthonormal rows iff $UU^t = I$.
- U has orthonormal columns iff $U^tU = I$.

If U is square and either holds, then they both hold.

The first two assertions are in §2.2. For the last assertion, assume $UU^t = I$. Then the rows are orthonormal, which implies U is full-rank. Since U is square, this implies U has an inverse U^{-1}. Multiplying $UU^t = I$ on the left by U^{-1},

$$U^{-1} = U^{-1}I = U^{-1}UU^t = U^t.$$

Since $U^{-1}U = I = UU^{-1}$, we have both $UU^t = I$ and $U^tU = I$. Similarly, $U^tU = I$ implies both $UU^t = I$ and $U^tU = I$.

A square matrix U satisfying

$$UU^t = I = U^tU \tag{2.9.2}$$

is an *orthogonal matrix*. Summarizing the above discussion, we can say

Orthogonal Matrix

A square matrix U is orthogonal iff its rows are an orthonormal basis iff its columns are an orthonormal basis.



Since

$$Uu \cdot Uv = u \cdot U^t U v = u \cdot v,$$

U preserves dot products. Since lengths squared are dot products, U also preserves lengths. Since angles are computed from dot products, U also preserves angles. Summarizing,

Angles, Lengths, and Dot Products

Orthogonal matrices preserve angles, lengths, and dot products of vectors.

As a consequence,

Orthogonal Matrix sends ON Vectors to ON Vectors

Let U be an orthogonal matrix. If $u_1, u_2, \ldots, u_d$ are orthonormal, and $v_1 = Uu_1, v_2 = Uu_2, \ldots, v_d = Uu_d$ are orthonormal, then $v_1, v_2, \ldots, v_d$ are orthonormal.

In two dimensions, $d = 2$, an orthogonal matrix must have two orthonormal columns, so must be of the form

$$U = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \qquad \text{or} \qquad U = \begin{pmatrix} \cos\theta & \sin\theta \\ \sin\theta & -\cos\theta \end{pmatrix}.$$

In the first case, U is a rotation, while in the second, U is a rotation followed by a reflection.



If $u_1, u_2, \ldots, u_d$ is an orthonormal basis of $\mathbf{R}^d$, and U has columns $u_1, u_2, \ldots, u_d$, then U is square and $UU^t = I = U^t U$. By (2.2.14), we have

$$I = u_1 \otimes u_1 + u_2 \otimes u_2 + \cdots + u_d \otimes u_d. \tag{2.9.3}$$

Multiplying both sides by u, we obtain

Orthonormal Basis Expansion

If $u_1, u_2, \ldots, u_d$ is an orthonormal basis, and u is any vector, then

$$u = (u \cdot u_1)u_1 + (u \cdot u_2)u_2 + \cdots + (u \cdot u_d)u_d \tag{2.9.4}$$

and

$$|u|^2 = (u \cdot u_1)^2 + (u \cdot u_2)^2 + \cdots + (u \cdot u_d)^2. \tag{2.9.5}$$



Let $x_1, x_2, \ldots, x_N$ be a dataset in $\mathbf{R}^d$, and let A be the $N \times d$ centered dataset matrix with rows $v_1, v_2, \ldots, v_N$. The dataset is *full-rank* if A is full-rank. Since A is full-rank iff its rows span (we assume $N >> d$, which means there are more samples than features), we have

Full-Rank Dataset

For any dataset, the following are equivalent.

1. The centered dataset matrix is full-rank.
2. The centered dataset spans the sample space.
3. The dataset does not lie in a hyperplane in sample space.

That equivalence of 2 and 3 was derived in §2.5. The *dimension* or *rank* of the dataset is the rank of its $N \times d$ centered dataset matrix A. Hence the dimension of the dataset equals the rank of A^tA. Since scaling a matrix has no effect on the rank, *the dimension or rank of a dataset equals the rank of its variance* $Q = A^tA/N$, (see (2.5.1)).



To derive the rank theorem, first we recall (2.7.6). Assume A has N rows and d columns. By (2.7.6), every vector x in the source space $\mathbf{R}^d$ can be written as a sum $x = u+v$ with u in the nullspace, and v in the row space. In other words, each vector x may be written as a sum $x = u + v$ with $Au = 0$ and v in the row space.

From this, we have

$$Ax = A(u + v) = Au + Av = Av.$$

This shows the column space consists of vectors of the form Av with v in the row space.

Let $v_1, v_2, \ldots, v_r$ be a basis for the row space. From the previous paragraph, it follows $Av_1, Av_2, \ldots, Av_r$ spans the column space of A. We claim

$Av_1, Av_2, \ldots, Av_r$ are linearly independent. To check this, we write

$$0 = t_1 Av_1 + t_2 Av_2 + \cdots + t_r Av_r = A(t_1 v_1 + t_2 v_2 + \cdots + t_r v_r).$$

If v is the vector $t_1 v_1 + t_2 v_2 + \cdots + t_r v_r$, this shows v is in the nullspace. But v is a linear combination of basis vectors of the row space, so v is also in the row space. Since the row space is the orthogonal complement of the nullspace, we must have v orthogonal to itself. Thus $v = 0$, or $t_1 v_1 + t_2 v_2 + \cdots + t_r v_r = 0$.

But $v_1, v_2, \ldots, v_r$ is a basis. By linear independence of $v_1, v_2, \ldots, v_r$, we conclude $t_1 = 0, \ldots, t_r = 0$. This establishes the claim, hence $Av_1, Av_2, \ldots, Av_r$ is a basis for the column space. This shows r is the dimension of the column space, which is by definition the column rank. Since by construction, r is also the row rank, this establishes the rank theorem.

Exercises

Exercise 2.9.1 Let u and v be nonzero vectors. Then the rank of $A = u \otimes v$ is one.

Exercise 2.9.2 Let μ be a unit vector in $\mathbf{R}^d$. Then the rank of $I - \mu \otimes \mu$ is $d - 1$.

Exercise 2.9.3 Use (2.9.4) to derive (2.9.5).

Exercise 2.9.4 If $u_1, u_2, \ldots, u_d$ and $v_1, v_2, \ldots, v_d$ are orthonormal bases, there is exactly one orthogonal U satisfying $v_k = Uu_k$, $1 \le k \le d$.

Exercise 2.9.5 Let $v_1, v_2, \ldots, v_N$ be an orthonormal basis in $\mathbf{R}^N$, and let Q be an $N \times N$ matrix. Use (2.9.3) and Exercise **2.2.7** to show

$$\operatorname{trace}(Q) = v_1 \cdot Qv_1 + v_2 \cdot Qv_2 + \cdots + v_N \cdot Qv_N. \tag{2.9.6}$$

Exercise 2.9.6 Let $v_1, v_2, \ldots, v_N$ be an orthonormal basis in $\mathbf{R}^N$, and let A be an $d \times N$ matrix. Use Exercise **2.9.5** and $Q = A^t A$ and (2.2.17) to show

$$\|A\|^2 = |Av_1|^2 + |Av_2|^2 + \cdots + |Av_N|^2. \tag{2.9.7}$$

Exercise 2.9.7 Let $v_1, v_2, \ldots, v_N$ be an orthonormal basis in $\mathbf{R}^N$, let $u_1, u_2, \ldots, u_d$ be an orthonormal basis in $\mathbf{R}^d$, and let A be an $d \times N$ matrix. Use Exercise **2.9.6** and (2.9.5) to show

$$\|A\|^2 = \sum_{i=1}^{d} \sum_{j=1}^{N} (u_i \cdot Av_j)^2. \tag{2.9.8}$$

Exercise 2.9.8 Let $u_1, u_2, \ldots, u_r$ be linearly independent, and $v_1, v_2, \ldots, v_r$ be linearly independent. Then the rank of

$$A = u_1 \otimes v_1 + u_2 \otimes v_2 + \cdots + u_r \otimes v_r$$

is r. (One way to do this is by writing out $Ax = 0$.)

Chapter 3
Principal Components

In this chapter, we look at the two fundamental methods of breaking or decomposing a matrix into elementary components, the eigenvalue decomposition and the singular value decomposition, then we apply this to principal component analysis.

Principal component analysis rests on an important phenomenon, that the eigenvalues of a large matrix cluster near the top and bottom: For a wide class of $d \times d$ variance matrices Q, *when d is large*, the eigenvalues of Q cluster near the top eigenvalue, or near the bottom eigenvalue.

When the bottom eigenvalue is zero, the eigenvalues near the bottom don't add up to anything substantial. On the other hand, because of this clustering, the eigenvalues of Q near the top provide the largest contribution to the explained variance trace(Q). We illustrate this for a specific class of matrices arising from mass-spring systems (§3.2).

We begin by looking at the geometry of a matrix as a linear transformation.

3.1 Geometry of Matrices

Matrix multiplication by an $N \times d$ matrix A sends a point x in the source space $\mathbf{R}^d$ to a point $b = Ax$ in the target space $\mathbf{R}^N$ (Figure 2.10).

Equivalently, since points in $\mathbf{R}^d$ are essentially the same as vectors in $\mathbf{R}^d$ (see §1.3), an $N \times d$ matrix A sends a vector v in $\mathbf{R}^d$ to a vector Av in $\mathbf{R}^N$.

Looked at this way, a matrix A induces a *linear transformation*: Matrix multiplication by A satisfies

$$A(v_1 + v_2) = Av_1 + Av_2, \qquad A(tv) = tAv.$$

One way to understand what the transformation does is to see how it distorts distances between vectors. If v_1 and v_2 are in $\mathbf{R}^d$, then the distance between them is $d = |v_1 - v_2|$. How does this compare with the distance

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

O. Hijab, *Math for Data Science*, https://doi.org/10.1007/978-3-031-89707-8_3

$|Av_1 - Av_2|$ between Av_1 and Av_2? If we let

$$u = \frac{v_1 - v_2}{|v_1 - v_2|},$$

then u is a unit vector, and, by linearity of A,

$$|Au| = \frac{|Av_1 - Av_2|}{|v_1 - v_2|}.$$

This ratio is a scaling factor of the linear transformation that depends on the given vectors v_1, v_2.

From this, it is enough to measure $|Au|$ for unit vectors u. Since $|Au| = 0$ for u in the nullspace of A, we measure $|Au|$ for u in the row space of A.

To this end, let σ_1 and σ_2 be the maximum and minimum of $|Au|$ over all unit vectors u in the row space of A, so

$$\sigma_2 \leq |Au| \leq \sigma_1.$$

Then σ_1 is the furthest distance of the image Au from the origin, and σ_2 is the nearest distance of the image Au to the origin. In this sense, σ_1 and σ_2 constrain $|Au|$ over unit vectors u in the row space of A.

Let A^+ be the pseudo-inverse of A. Related bounds on distortion are

$$\frac{1}{\sigma_1} \leq \left|(A^+)^t u\right| \leq \frac{1}{\sigma_2},$$

(Exercise **3.4.4**) and

$$1 \leq |Au| \times \left|(A^+)^t u\right| \leq \frac{1}{2}\left(\frac{\sigma_1}{\sigma_2} + \frac{\sigma_2}{\sigma_1}\right)$$

(Exercises **3.4.5** and **4.5.10**).



To keep things simple, assume both the source space and the target space are $\mathbf{R}^2$ and A is an invertible 2×2 matrix.

The image of the unit circle (in red in Figure 3.1) is the set of vectors of the form Au with $|u| = 1$. The annulus is the set (the region between the dashed circles in Figure 3.1) of vectors b satisfying

$$\{b : \sigma_2 < |b| < \sigma_1\}.$$

It turns out the image is an ellipse, and this ellipse lies in the annulus.

Thus the numbers σ_1 and σ_2 constrain how far the image of the unit circle is from the origin, and how near the image is to the origin.

To relate σ_1 and σ_2 to what we've seen before, let $Q = A^t A$. Then,

$$\sigma_1^2 = \max |Au|^2 = \max(Au) \cdot (Au) = \max u \cdot A^t A u = \max u \cdot Qu.$$

Thus σ_1^2 is the maximum projected variance corresponding to Q. Similarly, σ_2^2 is the minimum projected variance corresponding to Q.

Now let $Q = AA^t$, and let b be in the image. Then $b = Au$ for some unit vector u, and

$$b \cdot Q^{-1} b = (Au) \cdot Q^{-1} Au = u \cdot A^t (AA^t)^{-1} Au = u \cdot Iu = |u|^2 = 1.$$

This shows the image of the unit circle is the unit inverse variance ellipse (§1.4) corresponding to the variance Q, with major and minor axes length $2\sigma_1$ and $2\sigma_2$.

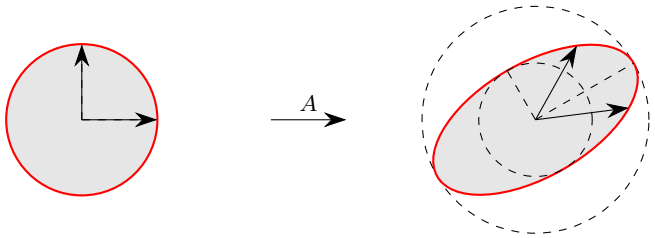


Fig. 3.1 Image of unit circle.

Let us look at some special cases. The first example is

$$V = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}. \tag{3.1.1}$$

If $e_1 = (1, 0)$, $e_2 = (0, 1)$ is the standard basis in $\mathbf{R}^2$. then the columns of V are

$$Ve_1 = (\cos\theta, \sin\theta), \qquad \text{and} \qquad Ve_2 = (-\sin\theta, \cos\theta).$$

Since $V^t V = I$, the columns of V are orthonormal. Thus V transforms the orthonormal basis e_1, e_2 into the orthonormal basis Ve_1, Ve_2 (see §2.9). By (A.4.4), V is a rotation by the angle θ.

The second example is

$$S = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix}.$$

Then S scales the horizontal direction by the factor σ_1, and S scales the vertical direction by σ_2.

The third example are the reflections

$$R = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad R = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

These reflect vectors across the horizontal axis, and across the vertical axis.

Recall an orthogonal matrix is a matrix U satisfying $U^tU = I = UU^t$ (2.9.2). Every orthogonal matrix U is a rotation V or a rotation times a reflection VR.



The SVD decomposition (§3.4) states that every matrix A can be written as a product

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} = USV.$$

Here S is a diagonal matrix as above, and U, V are orthogonal and rotation matrices as above.

In more detail, apart from a possible reflection, there are scalings σ_1 and σ_2 and angles α and β, so that A transforms vectors by first rotating by α, then scaling by (σ_1, σ_2), then by rotating by β (Figure 3.2).

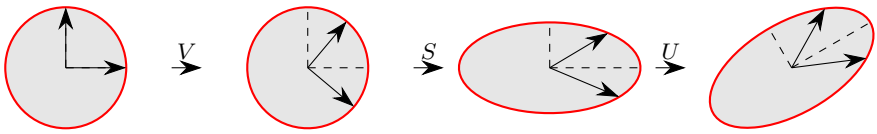


Fig. 3.2 SVD decomposition $A = USV$.

In other words, each 2×2 matrix A, consisting of four numbers a, b, c, d, may be described by four other numbers. These other numbers present a much clearer picture of the geometry of A: two angles α, β, and two scalings σ_1, σ_2.

Exercises

Exercise 3.1.1 Assume A is an invertible matrix satisfying $\sigma_2 \leq |Au| \leq \sigma_1$ for every unit vector u. Then for every unit vector u,

$$\frac{1}{\sigma_1} \leq \left|A^{-1}u\right| \leq \frac{1}{\sigma_2}.$$

3.2 Eigenvalue Decomposition

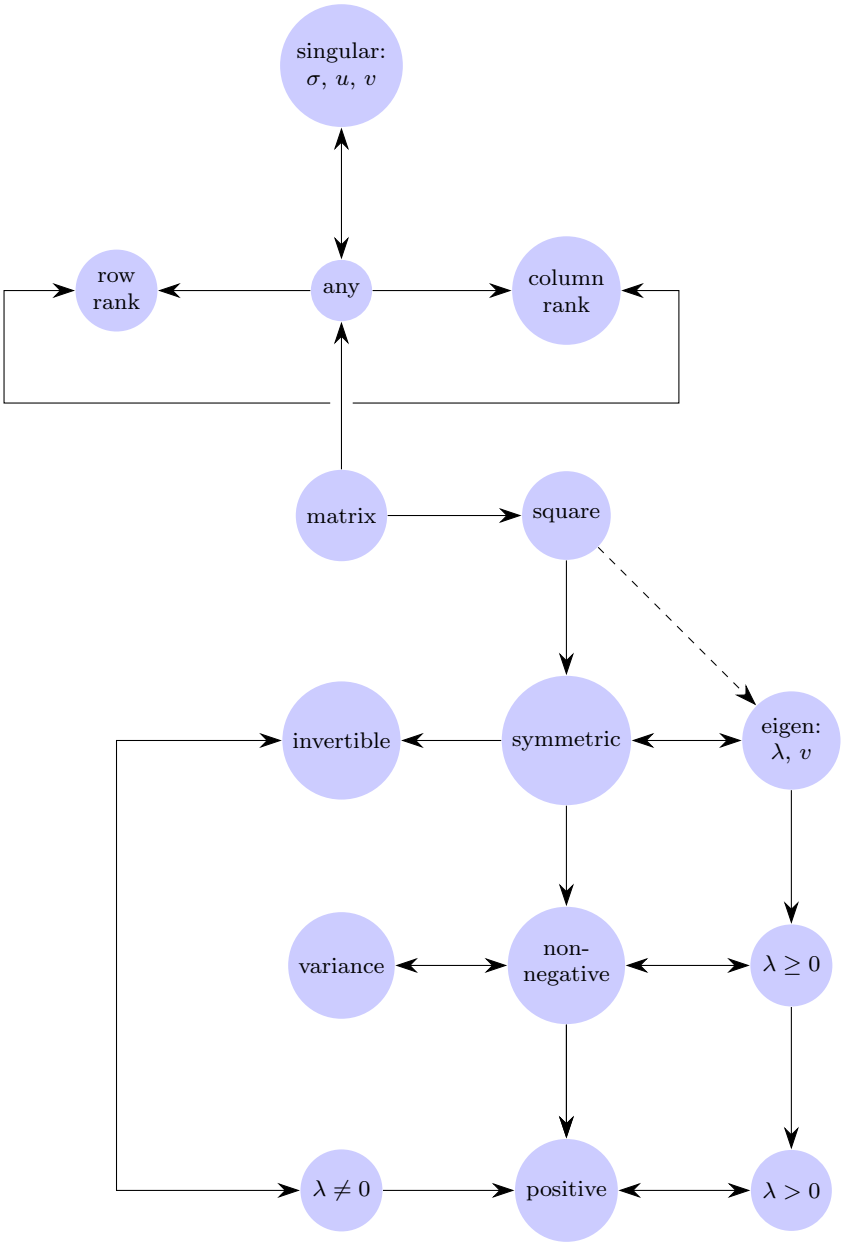


Fig. 3.3 Relations between matrix classes.

In §1.4 and §2.5, we saw every variance matrix Q is nonnegative, in the sense $u \cdot Qu \geq 0$ for every unit vector u. In this section, we see that every nonnegative matrix Q is the variance matrix of a specific dataset. The vectors in this dataset are the *principal components* of Q.

Let A be a matrix. An *eigenvector* for A is a *nonzero* vector v such that Av is aligned with v. This means

$$Av = \lambda v \tag{3.2.1}$$

for some scalar λ, the corresponding *eigenvalue*.

Because the solution $v = 0$ of (3.2.1) is not useful, we insist eigenvectors be nonzero. If v is an eigenvector, then the dimension of v equals the dimension of Av, which can only happen when A is a square matrix.

If v is an eigenvector corresponding to eigenvalue λ, then any scalar multiple $u = tv$ is also an eigenvector corresponding to eigenvalue λ, since

$$Av = \lambda v \qquad \Longrightarrow \qquad Au = A(tv) = t(Av) = t(\lambda v) = \lambda(tv) = \lambda u.$$

Because of this, we usually take eigenvectors to be unit vectors, by normalizing them.

Even then, this does not determine v uniquely, since both $\pm v$ are unit eigenvectors. This $\pm$ ambiguity is real, because different software packages make different sign choices. Because of this, when plotting or computing with datasets, units assumptions must be checked carefully.

Let

$$Q = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

Then Q has eigenvalues 3 and 1, with corresponding eigenvectors $(1, 1)$ and $(1, -1)$. These are not unit vectors, but the corresponding unit eigenvectors are $(1/\sqrt{2}, 1/\sqrt{2})$ and $(1/\sqrt{2}, -1/\sqrt{2})$.

The code

```
from numpy import *
from scipy.linalg import eig

A = array([[2,1],[1,2]])
lamda, U = eig(A)
lamda
```

returns the eigenvalues `[3,1]` as an array, and returns the eigenvectors v_1, v_2 of Q, as the *columns* of the matrix U. The matrix U is discussed further below.

The method `eig(A)` works on any square matrix `A`, but may return complex eigenvalues. When `eig(A)` returns real eigenvalues, they are not necessarily ordered in any predetermined fashion.

If the matrix `Q` is known to be symmetric, then the eigenvalues are guaranteed real. In this case, `eigh(Q)` returns the eigenvalues in increasing order. If `eigh` is used on a non-symmetric matrix, it will return erroneous data.

```
from numpy import *
from scipy.linalg import eigh

Q = array([[2,1],[1,2]])
lamda, U = eigh(Q)
lamda
```

returns the array `[1,3]`.



Let A be a square $d \times d$ matrix. The ideal situation is when there is a basis $v_1, v_2, \ldots, v_d$ in $\mathbf{R}^d$ of eigenvectors of A. However, this is not always the case. For example, if

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \tag{3.2.2}$$

and $Av = \lambda v$, then $v = (x, y)$ satisfies $x + y = \lambda x$, $y = \lambda y$. This system has only the nonzero solution $(x, y) = (1, 0)$ (or its multiples) and $\lambda = 1$. Thus A has only one eigenvector $e_1 = (1, 0)$, and the corresponding eigenvalue is $\lambda = 1$.

Let A be any square matrix.

Eigenvalues of A Versus Eigenvalues of A Transpose

The eigenvalues of A and the eigenvalues of A^t are the same.

This result is a consequence of the rank theorem in §2.9. To see why, suppose λ is an eigenvalue of A with corresponding eigenvector v. Then $Av = \lambda v$, which implies

$$(A - \lambda I)v = Av - \lambda v = 0.$$

As a consequence, if we let $B = A - \lambda I$, then v is an eigenvector of A corresponding to λ iff[1] v is in the nullspace of B. It follows λ is an eigenvalue for A iff B has a nonzero nullspace. Now $B^t = A^t - \lambda I$. If we show B^t has a nonzero nullspace, by the same logic, we will conclude λ is an eigenvalue of A^t. Now B has a nonzero nullspace iff B is not full-rank. Since B is square, by the rank theorem, this happens iff B^t is not full-rank, which happens iff B^t has a nonzero nullspace. Thus λ is an eigenvalue of A iff λ is an eigenvalue of A^t.

[1] *Iff* is short for *if and only if*.



Let v be a unit vector. From §2.5, when Q is the variance matrix of a dataset, $v \cdot Qv$ is the variance of the dataset projected onto the line through v. When v is an eigenvector, $Qv = \lambda v$, the variance equals

$$v \cdot Qv = v \cdot \lambda v = \lambda v \cdot v = \lambda.$$

More generally, this holds for any symmetric matrix Q. We conclude

Projected Variance along Eigenvector Direction

If v is a unit eigenvector of a symmetric matrix Q, then $v \cdot Qv$ equals the corresponding eigenvalue. In particular, the eigenvalues of a variance matrix are nonnegative.

In general, when Q is symmetric but not a variance matrix, some eigenvalues of Q may be negative.



Suppose λ and μ are eigenvalues of a symmetric matrix Q with corresponding eigenvectors u, v. Since Q is symmetric, $u \cdot Qv = v \cdot Qu$. Using $Qu = \lambda u$, $Qv = \mu v$, we compute $u \cdot Qv$ in two ways:

$$\mu u \cdot v = u \cdot (\mu v) = u \cdot Qv = v \cdot Qu = v \cdot (\lambda u) = \lambda u \cdot v.$$

This implies

$$(\mu - \lambda) u \cdot v = 0.$$

If $\lambda \neq \mu$, we must have $u \cdot v = 0$. We conclude:

Distinct Eigenvalues Have Orthogonal Eigenvectors

For a symmetric matrix Q, eigenvectors corresponding to distinct eigenvalues are orthogonal.

More generally, one can show (Exercise **3.2.15**)

Distinct Eigenvalues Have Linearly Independent Eigenvectors

Let A be any matrix and suppose $\lambda_1, \lambda_2, \ldots, \lambda_d$ are distinct (nonequal) eigenvalues of A Then the corresponding eigenvectors are linearly independent.



The main result in this section is

Eigenvalue Decomposition (EVD)

Let Q be a *symmetric* $d \times d$ matrix. There is an orthonormal basis v_1, v_2, ..., v_d in $\mathbf{R}^d$ of eigenvectors of Q, with corresponding eigenvalues

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d.$$

.

To see the implications of this main result, for simplicity, assume Q is a 2×2 symmetric matrix. By EVD, there is an orthonormal basis u, v in $\mathbf{R}^2$ and scalars λ, μ satisfying $Qu = \lambda u$, and $Qv = \mu v$. These are the eigenvalues and eigenvectors. Define three matrices

$$U = (u, v), \qquad E = \begin{pmatrix} \lambda & 0 \\ 0 & \mu \end{pmatrix}, \qquad V = \begin{pmatrix} u \\ v \end{pmatrix}.$$

Then the columns of U are u, v, the rows of V are u, v, and $V = U^t$.

By matrix-vector multiplication,

$$QU = (Qu, Qv) = (\lambda u, \mu v), \qquad UE = (\lambda u, \mu v).$$

We conclude $QU = UE$. Multiplying by V,

$$Q = QI = Q(UV) = (QU)V = UEV.$$

This result remains valid in general. To explain this, let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$ be the eigenvalues of a symmetric $d \times d$ matrix Q, and let

$$E = \begin{pmatrix} \lambda_1 & 0 & 0 & \dots & 0 \\ 0 & \lambda_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \lambda_{d-1} & 0 \\ 0 & 0 & \dots & 0 & \lambda_d \end{pmatrix}.$$

Then we have the following.

Diagonalization (EVD)

If v_1, v_2, ..., v_d is an orthonormal basis of eigenvectors of Q, with corresponding eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$, let V be the orthogonal matrix with rows v_1, v_2, ..., v_d, and let $U = V^t$. If E is the diagonal

matrix consisting of the eigenvalues, then

$$Q = UEV \tag{3.2.3}$$

When this happens, we say Q is *diagonalizable*. Thus A in (3.2.2) is not diagonalizable. In Python, E and U are computed by

```
# Q is any symmetric matrix
lamda, U = eigh(Q)
```

and `V = U.T`.

Be careful, though. `numpy` returns `lamda` as an array of eigenvalues, not a diagonal matrix. Nevertheless, these are *arranged in increasing order*, and the columns of U are the eigenvectors.

To verify this, check each row of $V = U^t$ is an eigenvector corresponding to an eigenvalue,

```
# lambda is a keyword in Python
# so we use lamda instead

Q = array([[2,1],[1,2]])

lamda, U = eigh(Q)

for eval, evec in zip(lamda,U.T):
   print(allclose(dot(Q,evec), eval*evec))
```

returns `True`.

The conclusion is: *With the correct choice of orthonormal basis, the matrix Q becomes a diagonal matrix E.*

The orthonormal basis eigenvectors $v_1, v_2, \ldots, v_d$ are the *principal components* of the matrix Q. The eigenvalues and eigenvectors of Q, taken together, are the *eigendata* of Q.

To obtain the diagonal matrix E,

```
E = diag(lamda)
E
```

Since `lambda` is a keyword in Python, we deliberately misspell it and write `lamda` in the code. When used as a `symbol` and pretty-printed, `sympy` knows to display `lamda` as λ.



In `sympy`, the corresponding commands are

```
from sympy import *

Q = Matrix([[2,1],[1,2]])

# eigenvalues
Q.eigenvals()

# eigenvectors
Q.eigenvects()

U, E = Q.diagonalize()
display(U, E)
```

This returns the diagonal E with the eigenvalues in increasing order.

If

$$A = \begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix},$$

and $Q = A^t A$, then the eigenvalues of Q are

$$\lambda_1 = 620 + 10\sqrt{3769}, \qquad \lambda_2 = 620 - 10\sqrt{3769}, \qquad \lambda_3 = 0, \tag{3.2.4}$$

and the eigenvectors are the rows of

$$V = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \frac{1}{94} \begin{pmatrix} -100 + 2\sqrt{3769} & -3 + \sqrt{3769} & 94 \\ -100 - 2\sqrt{3769} & -3 - \sqrt{3769} & 94 \\ 94 & -188 & 94 \end{pmatrix} \tag{3.2.5}$$

The third row is a multiple of $(1, -2, 1)$, which, as we know, is a basis for the nullspace of A (§2.4).

Let $\lambda_1, \lambda_2, \ldots, \lambda_r$ be the nonzero eigenvalues of Q. Then the diagonal matrix E has r nonzero entries on the diagonal, so $rank(E) = r$. Since U and $V = U^t$ are invertible, $rank(E) = rank(UEV)$. Since $Q = UEV$,

$$rank(Q) = rank(E) = r.$$

Rank Equals Number of Nonzero Eigenvalues

The rank of a diagonal matrix equals the number of nonzero entries. The rank of a square symmetric matrix Q equals the number of nonzero eigenvalues of Q.

For example, in (3.2.4), there are two positive eigenvalues, and the rank of Q, which equals the rank of A, is two.



Because real-life datasets are composed of floats, a more useful measure of the rank or dimension of a dataset matrix is the approximate dimension. The *approximate dimension* or *approximate rank* of A is the number of eigenvalues of the variance $Q = A^t A/N$ (see (2.5.1)) that are not almost zero, measured by `numpy`.

```
from sklearn import datasets

iris = datasets.load_iris()
dataset = iris["data"]

# dataset is Nxd
N, d = dataset.shape
Q = dot(dataset.T,dataset)/N

lamda, U = eigh(Q)

for i,eig in enumerate(lamda):
   if not allclose(eig,0):
      approx_nullity = i
      break

approx_rank = d - approx_nullity

approx_rank, approx_nullity
```

This code returns 712 for the MNIST dataset, agreeing with the code in §2.8.



Let's go back to diagonalization. Using `sympy`,

```
from sympy import *

Q = Matrix([[2,1],[1,2]])
```

```
U, E = Q.diagonalize()
display(U,E)
```

returns

$$U = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}, \qquad E = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}.$$

Also,

```
from sympy import *

a,b,c = symbols("a b c")

Q = Matrix([[a,b ],[b,c]])
U, E = Q.diagonalize()
display(Q,U,E)
```

returns

$$Q = \begin{pmatrix} a & b \\ b & c \end{pmatrix}, \qquad U = \frac{1}{2b} \begin{pmatrix} a - c - \sqrt{D} & a - c + \sqrt{D} \\ 2b & 2b \end{pmatrix}$$

and

$$E = \frac{1}{2} \begin{pmatrix} a + c - \sqrt{D} & 0 \\ 0 & a + c + \sqrt{D} \end{pmatrix}, \qquad D = (a - c)^2 + 4b^2.$$

When Q is a correlation matrix,

```
rho = symbols('rho')
Q = Matrix([[1,rho ],[rho,1]])

U, E = Q.diagonalize()
display(Q,U,E)
```

returns

$$\begin{pmatrix} 1 - \rho & 0 \\ 0 & \rho + 1 \end{pmatrix}, \qquad \begin{pmatrix} 1 - \rho & 0 \\ 0 & \rho + 1 \end{pmatrix}, \qquad \begin{pmatrix} 1 - \rho & 0 \\ 0 & \rho + 1 \end{pmatrix}.$$

(`display`, included in `Jupyter`, is used to pretty-print the output.)



When all the eigenvalues are nonzero, we can write

$$E^{-1} = \begin{pmatrix} 1/\lambda_1 & 0 & 0 & \dots & 0 \\ 0 & 1/\lambda_2 & 0 & \dots & 0 \\ \dots & \dots & \dots\dots & & \dots \\ 0 & 0 & \dots & 0 & 1/\lambda_d \end{pmatrix}.$$

Then a straightforward calculation using (3.2.3) shows

Nonzero Eigenvalues Equals Invertible

Let $Q = UEV$ be the EVD of a symmetric matrix Q. Then Q is invertible iff all its eigenvalues are nonzero. When this happens, we have

$$Q^{-1} = UE^{-1}V$$

More generally, using (2.6.8), one can check

Pseudo-Inverse and EVD

If $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r$ are the nonzero eigenvalues of Q, then $1/\lambda_1 \leq 1/\lambda_2 \leq \cdots \leq 1/\lambda_r$ are the nonzero eigenvalues of Q^+. Moreover, if U is an orthogonal matrix, and $V = U^t$, then

$$Q = UEV \qquad \Longrightarrow \qquad Q^+ = UE^+V. \tag{3.2.6}$$

Similarly, eigendata may be used to solve linear systems.

Nonzero Eigenvalues Equals Solvable

Let $v_1, v_2, \ldots, v_d$ be the orthonormal basis of eigenvectors of Q corresponding to eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_d$. Then the linear system

$$Qx = b$$

has a solution x for every vector b iff all eigenvalues are nonzero, in which case

$$x = \frac{1}{\lambda_1}(b \cdot v_1)v_1 + \frac{1}{\lambda_2}(b \cdot v_2)v_2 + \cdots + \frac{1}{\lambda_d}(b \cdot v_d)v_d. \tag{3.2.7}$$

The proof is straightforward using (2.9.4): multiply x by Q to verify. Another consequence of the eigenvalue decomposition is

Trace is the Sum of Eigenvalues

Let Q be a symmetric matrix with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_d$. Then

$$\operatorname{trace}(Q) = \lambda_1 + \lambda_2 + \cdots + \lambda_d. \tag{3.2.8}$$

To derive this, use (3.2.3): Since U is orthogonal, $VU = U^tU = I$. By (2.2.7), $\operatorname{trace}(AB) = \operatorname{trace}(BA)$, so

$$\operatorname{trace}(Q) = \operatorname{trace}(QUV) = \operatorname{trace}(VQU) = \operatorname{trace}(VUEVU) = \operatorname{trace}(E).$$

Since $E = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_d)$, $\text{trace}(E) = \lambda_1 + \lambda_2 + \cdots + \lambda_d$, and the result follows.

Let Q be symmetric with eigenvalues λ_1, λ_2, …, λ_d. Since

$$Qv = \lambda v \qquad \Longrightarrow \qquad Q^2 v = QQv = Q(\lambda v) = \lambda Q v = \lambda^2 v,$$

Q^2 is symmetric with eigenvalues λ_1^2, λ_2^2, …, λ_d^2. Applying the last result to Q^2, we have

$$\text{trace}(Q^2) = \text{trace}(QQ^t) = \text{trace}(Q^2) = \lambda_1^2 + \lambda_2^2 + \cdots + \lambda_d^2.$$



It turns out every nonnegative matrix Q is the variance of a simple dataset (Figure 3.4).

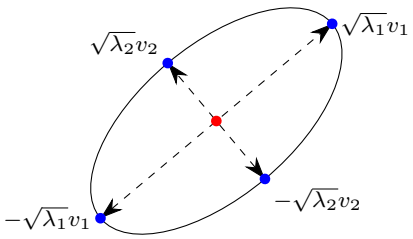


Fig. 3.4 Inverse variance ellipse and centered dataset.

Sum of Tensor Products

Let Q be a symmetric $d \times d$ matrix with eigenvalues λ_1, λ_2, …, λ_d and orthonormal eigenvectors v_1, v_2, …, v_d. Then

$$Q = \lambda_1 v_1 \otimes v_1 + \lambda_2 v_2 \otimes v_2 + \cdots + \lambda_d v_d \otimes v_d. \tag{3.2.9}$$

In particular, when Q is nonnegative, the dataset consisting of the $2d$ points

$$\pm\sqrt{d\lambda_1}v_1, \pm\sqrt{d\lambda_2}v_2, \ldots, \pm\sqrt{d\lambda_d}v_d$$

is centered and has variance Q.

The vectors in this dataset are the *principal components* of Q. Because of this result, we refer more broadly to any symmetric nonnegative matrix Q as a *variance matrix*.

Since v_1, v_2, …, v_d is an orthonormal basis, by (2.9.4), every vector v can be written

$$v = (v \cdot v_1)\, v_1 + (v \cdot v_2)\, v_2 + \cdots + (v \cdot v_d)\, v_d.$$

Multiply by Q. Since $Qv_k = \lambda_k v_k$,

$$\begin{aligned} Qv &= (v \cdot v_1)\, Qv_1 + (v \cdot v_2)\, Qv_2 + \cdots + (v \cdot v_d)\, Qv_d \\ &= \lambda_1 (v \cdot v_1)\, v_1 + \lambda_2 (v \cdot v_2)\, v_2 + \cdots + \lambda_d (v \cdot v_d)\, v_d \\ &= (\lambda_1 v_1 \otimes v_1 + \lambda_2 v_2 \otimes v_2 + \cdots + \lambda_d v_d \otimes v_d)\, v \end{aligned}$$

This proves the first part. For the second part, let $b_k = \sqrt{\lambda_k} v_k$. Then the mean of the $2d$ vectors $\pm b_1$, $\pm b_2$, …, $\pm b_d$ is clearly zero, and by (3.2.9), the variance matrix

$$\frac{2}{2d} (b_1 \otimes b_1 + b_2 \otimes b_2 + \cdots + b_d \otimes b_d)$$

equals Q/d.



Now we approach the eigenvalues of Q from a different angle. In §2.5, we studied zero variance directions. Since the eigenvalues of a variance matrix are nonnegative, for a variance matrix, they may also be called minimum variance directions. Now we study maximum variance directions.

Let

$$\lambda_1 = \max_{|v|=1} v \cdot Qv,$$

where the maximum is over all unit vectors v. We say a *unit* vector b is *best-fit for* Q or *best-aligned with* Q if the maximum is achieved at $v = b$: $\lambda_1 = b \cdot Qb$. When Q is a variance matrix, this means the unit vector b is chosen so that the variance $b \cdot Qb$ of the dataset projected onto b is maximized.

An eigenvalue λ_1 of Q is the *top eigenvalue* if $\lambda_1 \geq \lambda$ for any other eigenvalue. An eigenvalue λ_1 of Q is the *bottom eigenvalue* if $\lambda_1 \leq \lambda$ for any other eigenvalue. We establish the following results.

Maximum Projected Variance is an Eigenvalue

Let Q be a symmetric matrix. Then

$$\lambda_1 = \max_{|v|=1} v \cdot Qv, \tag{3.2.10}$$

is the top eigenvalue of Q.

Best-aligned vector is an eigenvector

Let Q be a symmetric matrix. Then a best-aligned vector b is an eigenvector of Q corresponding to the top eigenvalue λ_1.



To prove these results, we begin with a simple calculation, whose derivation we skip. This result may be derived directly using algebra, or by using calculus and setting $f'(0)$ to equal zero.

A Calculation

Suppose λ, a, b, c, d are real numbers and let

$$f(t) = \frac{\lambda + at + bt^2}{1 + ct + dt^2}.$$

If $f(t)$ is maximized at $t = 0$, then $a = \lambda c$.

Let λ be any eigenvalue of Q, with eigenvector v: $Qv = \lambda v$. Dividing v by its length, we may assume $|v| = 1$. Then

$$\lambda_1 \geq v \cdot Qv = v \cdot (\lambda v) = \lambda v \cdot v = \lambda.$$

This shows $\lambda_1 \geq \lambda$ for any eigenvalue λ.

Now we show λ_1 itself is an eigenvalue. Let v_1 be a unit vector maximizing $v \cdot Qv$, so v_1 is best-fit for Q. Then

$$\lambda_1 = v_1 \cdot Qv_1 \geq v \cdot Qv \tag{3.2.11}$$

for all unit vectors v. Let u be any vector. Then for any real t,

$$v = \frac{v_1 + tu}{|v_1 + tu|}$$

is a unit vector. Insert this v into (3.2.11) to obtain

$$\lambda_1 \geq \frac{(v_1 + tu) \cdot Q(v_1 + tu)}{|v_1 + tu|^2}.$$

Since Q is symmetric, $u \cdot Qv_1 = v_1 \cdot Qu$. Expanding with $|v_1|^2 = 1$, we obtain

$$\lambda_1 \geq \frac{\lambda_1 + 2tu \cdot Qv_1 + t^2 u \cdot Qu}{1 + 2tu \cdot v_1 + t^2|u|^2} = \frac{\lambda_1 + at + bt^2}{1 + ct + dt^2}.$$

Applying the calculation with $\lambda = \lambda_1$, $a = 2u \cdot Qv_1$, $b = u \cdot Qu$, $c = 2u \cdot v_1$, and $d = |u|^2$, we conclude

$$u \cdot Qv_1 = \lambda_1 u \cdot v_1$$

for all vectors u. But this implies

$$u \cdot (Qv_1 - \lambda_1 v_1) = 0$$

for all u. Thus $Qv_1 - \lambda_1 v_1$ is orthogonal to all vectors, hence orthogonal to itself. Since this can only happen if $Qv_1 - \lambda_1 v_1 = 0$, we conclude $Qv_1 = \lambda_1 v_1$. Hence λ_1 is itself an eigenvalue. This completes the proof of the two results.



Just as the maximum variance (3.2.10) is the top eigenvalue λ_1, the minimum variance

$$\lambda_d = \min_{|v|=1} v \cdot Qv, \tag{3.2.12}$$

is the bottom eigenvalue, and the corresponding eigenvector v_d is the worst-aligned vector.

By the eigenvalue decomposition, the eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$ of a symmetric matrix Q may be arranged in decreasing order, and may be positive, zero, or negative scalars. When Q is a variance, the eigenvalues are nonnegative, and the bottom eigenvalue is at least zero. When the bottom eigenvalue is zero, the corresponding eigenvectors are zero variance directions.



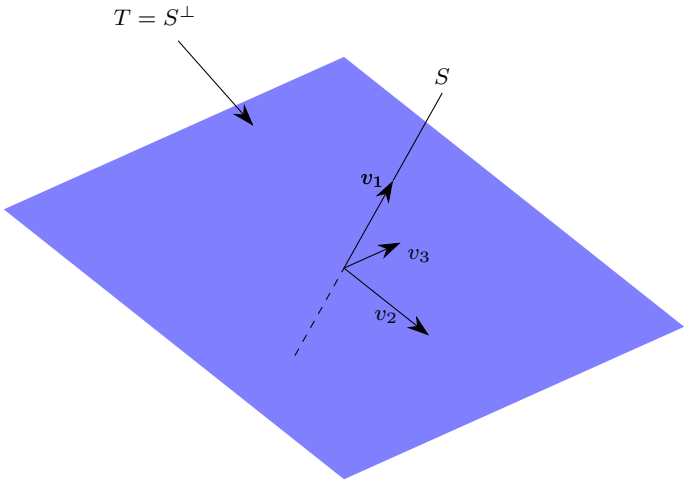


Fig. 3.5 $S = span(v_1)$ and $T = S^{\perp}$.

Now we can complete the proof the eigenvalue decomposition. Having found the top eigenvalue λ_1 with its corresponding unit eigenvector v_1, we

let $S = span(v_1)$ and $T = S^\perp$ be the orthogonal complement of v_1 (Figure 3.5).

Then $\dim(T) = d - 1$, and we can repeat the process and maximize $v \cdot Qv$ over all unit v in T, i.e. over all unit v orthogonal to v_1. This leads to another eigenvalue λ_2 with corresponding eigenvector v_2 orthogonal to v_1.

Since λ_1 is the maximum of $v \cdot Qv$ over all vectors in $\mathbf{R}^d$, and λ_2 is the maximum of $v \cdot Qv$ over the restricted space T of vectors orthogonal to v_1, we must have $\lambda_1 \geq \lambda_2$.

Having found the top two eigenvalues $\lambda_1 \geq \lambda_2$ and their orthonormal eigenvectors v_1, v_2, we let $S = span(v_1, v_2)$ and $T = S^\perp$ be the orthogonal complement of S. Then $\dim(T) = d - 2$, and we can repeat the process to obtain λ_3 and v_3 in T. Continuing in this manner, we obtain eigenvalues

$$\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_d.$$

with corresponding orthonormal eigenvectors

$$v_1, v_2, v_3, \ldots, v_d.$$

This proves the eigenvalue decomposition.



Let Q be a positive variance matrix and let $b \cdot Q^{-1}b = 1$ be the inverse variance ellipsoid. If v is a unit eigenvector corresponding to an eigenvalue λ, then $\lambda \geq 0$, and the vector $b = \sqrt{\lambda}v$ has length $\sqrt{\lambda}$. Moreover b satisfies

$$b \cdot Q^{-1}b = (\sqrt{\lambda}v) \cdot Q^{-1}(\sqrt{\lambda}v) = \lambda v \cdot Q^{-1}v = \lambda v \cdot (\lambda^{-1}v) = v \cdot v = 1.$$

Hence the line segment joining the vectors $\pm\sqrt{\lambda}v$ is an axis of the inverse variance ellipsoid, with length $2\sqrt{\lambda}$ (Figure 3.4).

When $\lambda = \lambda_1$ is the top eigenvalue, the axis is the *principal axis* of the inverse variance ellipsoid. When $\lambda = \lambda_2$ is the next highest eigenvalue, the axis is orthogonal to the principal axis, and is the second principal axis. Continuing in this manner, we obtain all the principal axes of the inverse variance ellipsoid.

Principal Axes of Inverse Variance Ellipsoid

Let v be a unit eigenvector of a variance matrix Q with eigenvalue λ. Then the line segment joining $-\sqrt{\lambda}v$ and $+\sqrt{\lambda}v$ is a principal axis of the inverse variance ellipsoid, with length $2\sqrt{\lambda}$.

Together with Figure 1.15, this result provides a geometric interpretation of eigenvalues: They control the variances of a dataset's points, in the principal directions.

Sometimes, several eigenvalues are equal, leading to several eigenvectors, say m of them, corresponding to a given eigenvalue λ. In this case, we say the eigenvalue λ has *multiplicity* m, and we call the span

$$S_\lambda = \{v : Qv = \lambda v\}$$

the *eigenspace* corresponding to λ. For example, suppose the top three eigenvalues are equal: $\lambda_1 = \lambda_2 = \lambda_3$, with b_1, b_2, b_3 the corresponding eigenvectors. Calling this common value λ, the eigenspace is $S_\lambda = span(b_1, b_2, b_3)$. Since b_1, b_2, b_3 are orthonormal, $\dim(V_\lambda) = 3$. In Python, the eigenspaces V_λ are obtained by the matrix U above: The columns of U are an orthonormal basis for the entire space, so selecting the columns corresponding to a specific λ yields an orthonormal basis for S_λ.



Let `(E,U)` be the list of eigenvalues and matrix U whose columns are the eigenvectors. Then the eigenvectors are the rows of U^t. Here is code for selecting just the eigenvectors corresponding to eigenvalue `s`.

```
iris = datasets.load_iris()
dataset = iris["data"]
N, d = dataset.shape
Q = dot(dataset.T,dataset)/N

lamda, U = eigh(Q)
V = U.T

s = 2.10302878
V[isclose(lamda,s)]
```

The function `isclose(a,b)` returns `True` when `a` and `b` are numerically close. Using this boolean, we extract only those rows of V whose corresponding eigenvalue is close to `s`.

The subspace S_λ is defined for any λ. However, $\dim(S_\lambda) = 0$ unless λ is an eigenvalue, in which case $\dim(S_\lambda) = m$, where m is the multiplicity of λ.

The proof of the eigenvalue decomposition provides a systematic procedure for finding eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$. Now we show there are no other eigenvalues.

The Eigenvalue Decomposition is Complete

If λ is an eigenvalue for Q, $Qv = \lambda v$, then λ equals one of the eigenvalues in the eigenvalue decomposition of Q.

To see this, suppose $Qv = \lambda v$ with $\lambda \neq \lambda_j$ for $j = 1, \ldots, d$. Since $\lambda \neq \lambda_j$ for $j = 1, \ldots, d$, the vector v must be orthogonal to every v_j, $j = 1, \ldots, d$.

Since $span(v_1, \ldots, v_d) = \mathbf{R}^d$, it follows v is orthogonal to every vector, hence v is orthogonal to itself, hence $v = 0$. We conclude λ cannot be an eigenvalue.



All this can be readily computed in Python. The eigenvalues of the variance Q of the Iris dataset are

$$4.2 > 0.24 > 0.08 > 0.02,$$

and the orthonormal eigenvectors are the *columns* of the matrix

$$U = \begin{pmatrix} 0.36 & -0.66 & -0.58 & 0.32 \\ -0.08 & -0.73 & 0.6 & -0.32 \\ 0.86 & 0.18 & 0.07 & -0.48 \\ 0.36 & 0.07 & 0.55 & 0.75 \end{pmatrix}$$

Since the eigenvalues are distinct, the multiplicity of each eigenvalue is 1. The variance Q was computed in §2.2 and its trace is

$$4.54 = \text{trace}(Q) = \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4.$$

For the Iris dataset, the top eigenvalue is $\lambda_1 = 4.2$, it has multiplicity 1, and its corresponding list of eigenvectors contains only one eigenvector,

$$v_1 = (0.36, -0.08, 0.86, 0.36).$$

The top eigenvalue accounts for 92.5% of the total variance.

The second eigenvalue is $\lambda_2 = 0.24$ with eigenvector

$$v_2 = (-0.66, -0.73, 0.18, 0.07).$$

The top two eigenvalues account for 97.8% of the total variance.

The third eigenvalue is $\lambda_3 = 0.08$ with eigenvector

$$v_3 = (-0.58, 0.60, 0.07, 0.55).$$

The top three eigenvalues account for 99.5% of the total variance.

The fourth eigenvalue is $\lambda_4 = 0.02$ with eigenvector

$$v_4 = (0.32, -0.32, -0.48, 0.75).$$

The top four eigenvalues account for 100% of the total variance. Here each eigenvalue has multiplicity 1, since there are four distinct eigenvalues.



An important class of symmetric matrices are of the form

$$\begin{pmatrix} 2 & -2 \\ -2 & 2 \end{pmatrix} \begin{pmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{pmatrix} \begin{pmatrix} 2 & -1 & 0 & -1 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ -1 & 0 & -1 & 2 \end{pmatrix}$$

$$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ -1 & 0 & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} 2 & -1 & 0 & 0 & 0 & -1 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ -1 & 0 & 0 & 0 & -1 & 2 \end{pmatrix}$$

$$\begin{pmatrix} 2 & -1 & 0 & 0 & 0 & 0 & -1 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ -1 & 0 & 0 & 0 & 0 & -1 & 2 \end{pmatrix}.$$

We denote these matrices $Q(2)$, $Q(3)$, $Q(4)$, $Q(5)$, $Q(6)$, $Q(7)$. The following code generates these symmetric $d \times d$ matrices $Q(d)$,

```
def row(i,d):
   v = [0]*d
   v[i] = 2
   if i > 0: v[i-1] = -1
   if i < d-1: v[i+1] = -1
   if i == 0: v[d-1] += -1
   if i == d-1: v[0] += -1
   return v

# using sympy
from sympy import Matrix

def Q(d): return Matrix([ row(i,d) for i in range(d) ])

# using numpy
from numpy import *

def Q(d): return array([ row(i,d) for i in range(d) ])
```



The eigenvalues of these symmetric matrices follow interesting patterns that are best explored using Python.

Below we will see, the eigenvalues of $Q(d)$ are between 4 and 0, and each eigenvalue other than 4 and 0 has multiplicity 2.

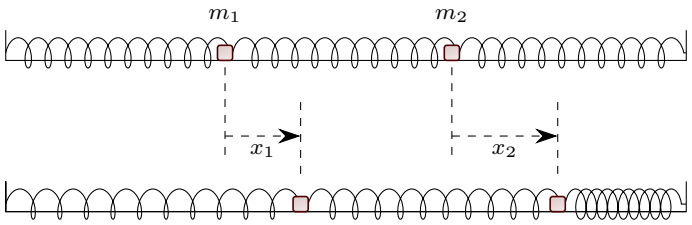


Fig. 3.6 Three springs at rest and perturbed.

To explain where these matrices come from, look at the mass-spring systems in Figures 3.6 and 3.7. Here we have springs attached to masses and walls on either side. At rest, the springs are the same length. When perturbed, some springs are compressed and some stretched. In Figure 3.6, let x_1 and x_2 denote the displacement of each mass from its rest position.

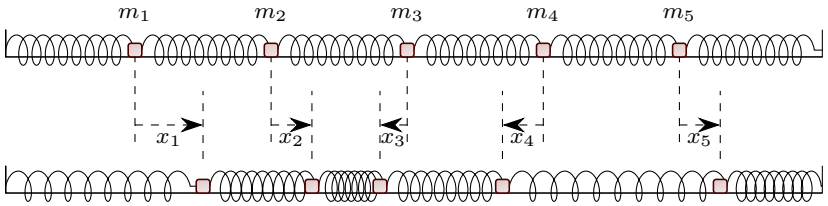


Fig. 3.7 Six springs at rest and perturbed.

When extended by x, each spring fights back by exerting a force kx proportional to the displacement x. Here k is the *spring constant*. For example, look at the mass m_1. The spring to its left is extended by x_1, so exerts a force of $-kx_1$. Here the minus indicates pulling to the left. On the other hand, the spring to its right is extended by $x_2 - x_1$, so it exerts a force $+k(x_2 - x_1)$. Here the plus indicates pulling to the right. Adding the forces from either side, the total force on m_1 is $-k(2x_1 - x_2)$. For m_2, the spring to its left exerts a force $-k(x_2 - x_1)$, and the spring to its right exerts a force $-kx_2$, so the total force on m_2 is $-k(2x_2 - x_1)$. We obtain the force vector

$$-k \begin{pmatrix} 2x_1 - x_2 \\ -x_1 + 2x_2 \end{pmatrix} = -k \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}.$$

However, as you can see, the matrix here is not exactly $Q(2)$.

For five masses, let x_1, x_2, x_3, x_4, x_5 denote the displacement of each mass from its rest position. In Figure 3.7, x_1, x_2, x_5 are positive, and x_3, x_4 are negative.

As before, the total force on m_1 is $-k(2x_1 - x_2)$, and the total force on m_5 is $-k(2x_5 - x_4)$. For m_2, the spring to its left exerts a force $-k(x_2 - x_1)$, and

the spring to its right exerts a force $+k(x_3 - x_2)$. Hence, the total force on m_2 is $-k(-x_1 + 2x_2 - x_3)$. Similarly for m_3, m_4. We obtain the force vector

$$-k\begin{pmatrix} 2x_1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + 2x_3 - x_4 \\ -x_3 + 2x_4 - x_5 \\ -x_4 + 2x_5 \end{pmatrix} = -k\begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}.$$

But, again, the matrix here is not $Q(5)$. Notice, if we place one mass and two springs in Figure 3.6, we obtain the 1×1 matrix 2.

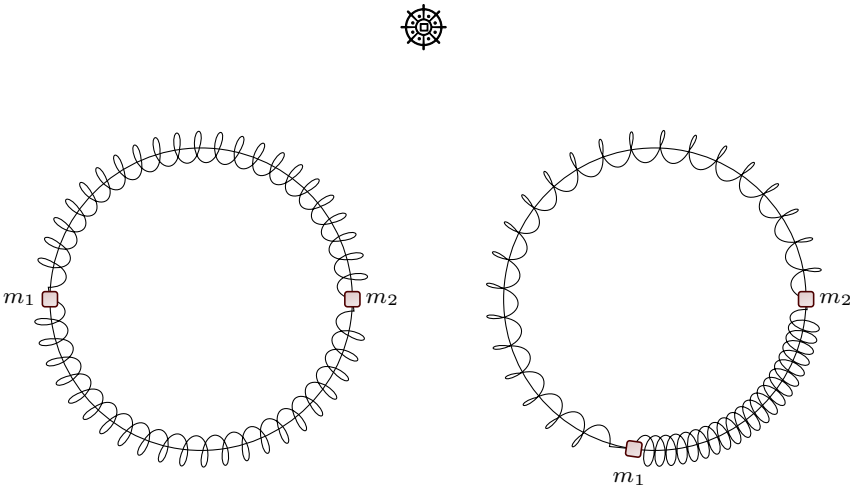


Fig. 3.8 Two springs along a circle leading to $Q(2)$.

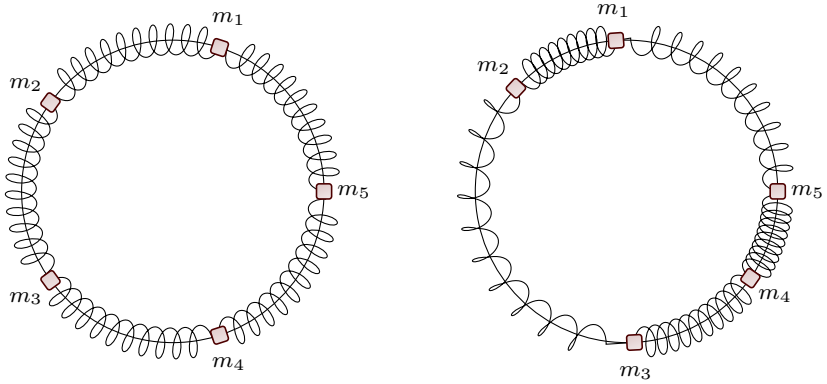


Fig. 3.9 Five springs along a circle leading to $Q(5)$.

To obtain $Q(2)$ and $Q(5)$, we place the springs along a circle, as in Figures 3.8 and 3.9. Now we have as many springs as masses. Repeating the same logic, this time we obtain $Q(2)$ and $Q(5)$. Notice if we place one mass and one spring in Figure 3.8, $d = 1$, we obtain the 1×1 matrix $Q(1) = 0$: There is no force if we move a single mass around the circle, because the spring is not being stretched.

Thus the matrices $Q(d)$ arise from mass-spring systems arranged on a circle. From Newton's law (force equals mass times acceleration), one shows the frequencies of the vibrating springs equal $\sqrt{\lambda k/m}$, where k is the spring constant, m is the mass of each of the masses, and λ is an eigenvalue of $Q(d)$. This is the physical meaning of the eigenvalues of $Q(d)$.



Let v have features $(x_1, x_2, \ldots, x_d)$, and let $Q = Q(d)$. By elementary algebra, check that

$$v \cdot Qv = (x_1 - x_2)^2 + (x_2 - x_3)^2 + \cdots + (x_{d-1} - x_d)^2 + (x_d - x_1)^2. \quad (3.2.13)$$

As a consequence of (3.2.13), show also the following.

- For any vector v, $0 \le v \cdot Qv \le 4|v|^2$. Conclude every eigenvalue λ satisfies $0 \le \lambda \le 4$.
- $\lambda = 0$ is an eigenvalue, with multiplicity 1.
- When d is even, $\lambda = 4$ is an eigenvalue with multiplicity 1.
- When d is odd, $\lambda = 4$ is not an eigenvalue.



To compute the eigenvalues, we use complex numbers, specifically the d-th root of unity ω (§A.5). Let

$$p(t) = 2 - t - t^{d-1},$$

and let

$$v_1 = \begin{pmatrix} 1 \\ \omega \\ \omega^2 \\ \omega^3 \\ \vdots \\ \omega^{d-1} \end{pmatrix}.$$

Then Qv_1 is

$$Qv_1 = \begin{pmatrix} 2-\omega-\omega^{d-1} \\ -1+2\omega-\omega^2 \\ -\omega+2\omega^2-\omega^3 \\ \vdots \\ -\omega^{d-2}+2\omega^{d-1}-1 \end{pmatrix} = p(\omega) \begin{pmatrix} 1 \\ \omega \\ \omega^2 \\ \omega^3 \\ \vdots \\ \omega^{d-1} \end{pmatrix} = p(\omega)v_1.$$

Thus v_1 is an eigenvector corresponding to eigenvalue $p(\omega)$.

For each $k = 0, 1, 2, \ldots, d-1$, define

$$v_k = \left(1, \omega^k, \omega^{2k}, \omega^{3k}, \ldots, \omega^{(d-1)k}\right). \tag{3.2.14}$$

Then

$$v_0 = \mathbf{1} = (1, 1, \ldots, 1),$$

and, by the same calculation, we have

$$Qv_k = p(\omega^k)v_k, \qquad k = 0, 1, 2, \ldots, d-1.$$

By (A.5.9),

$$p(\omega^k) = 2 - \omega^k - \omega^{(d-1)k} = 2 - \omega^k - \omega^{-k} = 2 - 2\cos(2\pi k/d).$$

Eigenvalues of $Q(d)$

The (unsorted) eigenvalues of $Q(d)$ are

$$\lambda_k = p(\omega^k) = 2 - 2\cos\left(\frac{2\pi k}{d}\right), \tag{3.2.15}$$

with corresponding eigenvectors v_k given by (3.2.14), $k = 0, 1, 2, \ldots, d-1$.

Corresponding to each eigenvalue λ_k, there is the complex eigenvector v_k. Separating v_k into its real and imaginary parts yields two real eigenvectors

$$\Re(v_k) = \left(1, \cos\left(\frac{2\pi k}{d}\right), \cos\left(\frac{4\pi k}{d}\right), \cos\left(\frac{6\pi k}{d}\right), \ldots, \cos\left(\frac{2(d-1)\pi k}{d}\right)\right),$$
$$\Im(v_k) = \left(0, \sin\left(\frac{2\pi k}{d}\right), \sin\left(\frac{4\pi k}{d}\right), \sin\left(\frac{6\pi k}{d}\right), \ldots, \sin\left(\frac{2(d-1)\pi k}{d}\right)\right).$$

When $k = 0$ or when $k = d/2$, d even, we have $\Im(v_k) = 0$. This explains the double multiplicity in Figure 3.10, except when $k = 0$ or $k = d/2$, d even.

Applying this formula, we obtain eigenvalues

$$
\begin{aligned}
Q(2) &= (4,0)\\
Q(3) &= (3,3,0)\\
Q(4) &= (4,2,2,0),\\
Q(5) &= \left(\frac{5}{2}+\frac{\sqrt{5}}{2},\frac{5}{2}+\frac{\sqrt{5}}{2},\frac{5}{2}-\frac{\sqrt{5}}{2},\frac{5}{2}-\frac{\sqrt{5}}{2},0\right)\\
Q(6) &= (4,3,3,1,1,0)\\
Q(8) &= (4,2+\sqrt{2},2+\sqrt{2},2,2,2-\sqrt{2},2-\sqrt{2},0)\\
Q(10) &= \left(4,\frac{5}{2}+\frac{\sqrt{5}}{2},\frac{5}{2}+\frac{\sqrt{5}}{2},\frac{3}{2}+\frac{\sqrt{5}}{2},\frac{3}{2}+\frac{\sqrt{5}}{2},\right.\\
&\qquad \left.\frac{5}{2}-\frac{\sqrt{5}}{2},\frac{5}{2}-\frac{\sqrt{5}}{2},\frac{3}{2}-\frac{\sqrt{5}}{2},\frac{3}{2}-\frac{\sqrt{5}}{2},0\right)\\
Q(12) &= \left(4,2+\sqrt{3},2+\sqrt{3},3,3,2,2,1,1,2-\sqrt{3},2-\sqrt{3},0\right).
\end{aligned}
$$

The matrices $Q(d)$ are *circulant matrices*. Each row in $Q(d)$ is obtained from the row above it in $Q(d)$ by shifting the entries to the right. The trick of using the roots of unity to compute the eigenvalues and eigenvectors works for any circulant matrix.



Our last topic is the distribution of the eigenvalues for large d. How are the eigenvalues scattered? Figure 3.10 plots the eigenvalues for $Q(50)$ using the code below.

```
from scipy.linalg import eigh
from matplotlib.pyplot import *

d = 50
E = eigh(Q(d))[0]
stairs(E,range(d+1),label = "numpy")

k = arange(d)
lamda = 2 - 2*cos(2*pi*k/d)
sorted = sort(lamda)

scatter(k,lamda,s = 5,label = "unordered")
scatter(k,sorted,c = "red",s = 5,label = "increasing order")

grid()
legend()
show()
```

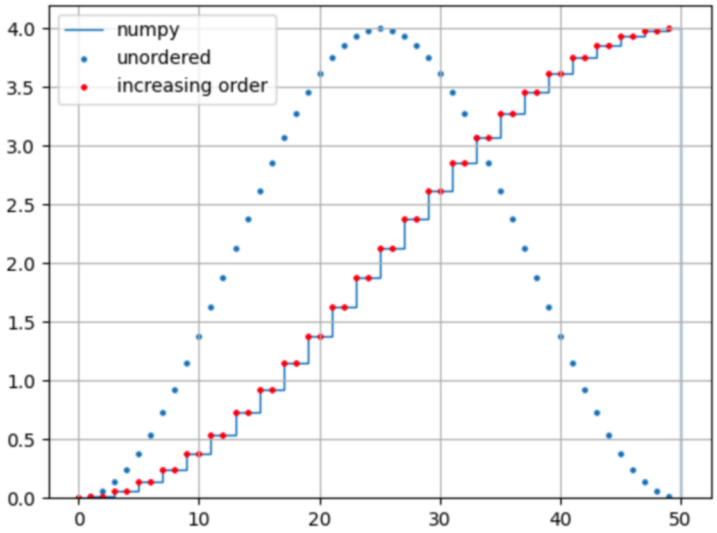


Fig. 3.10 Plot of eigenvalues of $Q(50)$.

Figure 3.10 shows the eigenvalues tend to cluster near the top $\lambda_1 \approx 4$ and the bottom $\lambda_d = 0$, they are sparser near the middle. Using the double-angle formula,

$$\lambda_k = 4\sin^2\left(\frac{\pi k}{d}\right), \qquad k = 0, 1, 2, \ldots, d-1.$$

Solving for k/d in terms of λ, and multiplying by two to account for the double multiplicity, we obtain the proportion of eigenvalues below threshold λ,

$$\frac{\#\{k : \lambda_k \le \lambda\}}{d} \approx \frac{2}{\pi}\arcsin\left(\frac{1}{2}\sqrt{\lambda}\right), \qquad 0 \le \lambda \le 4. \tag{3.2.16}$$

Here $\approx$ means asymptotic equality as $d \to \infty$, see §A.7.

Equivalently, the derivative (4.1.28) of the *arcsine law* (3.2.16) exhibits the eigenvalue clustering near the ends (Figure 3.11).

```
lamda = arange(0.1,3.9,.01)
density = 1/(pi*sqrt(lamda*(4-lamda)))
plot(lamda,density)
# r"..." means raw string
tex = r"$\displaystyle\frac1{\pi\sqrt{\lambda(4-\lambda)}}$"
text(.5,.45,tex,usetex=True,fontsize="x-large")

grid()
show()
```

The matrices $Q(d)$ are prototypes of matrices that are fundamental in many areas of physics and engineering, including time series analysis and information theory, see [12]. This clustering of eigenvalues near the top and bottom is valid for a wide class of matrices, not just $Q(d)$, as the matrix size d grows without bound, $d \to \infty$.

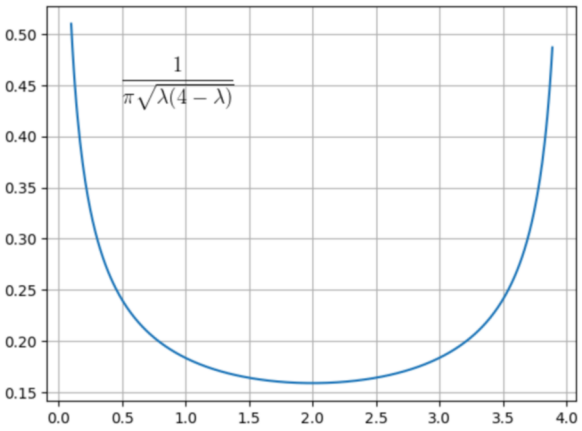


Fig. 3.11 Density of eigenvalues of $Q(d)$ for d large.

Exercises

Exercise 3.2.1 Let A be a 2×2 matrix. Show λ is an eigenvalue of A when $\det(A - \lambda I) = 0$. (See homogeneous systems in §A.4.)

Exercise 3.2.2 Let A be a 2×2 matrix. Show λ is an eigenvalue of A when

$$\lambda^2 - \lambda \operatorname{trace}(A) + \det(A) = 0.$$

Exercise 3.2.3 Let Q be a 2×2 symmetric matrix. Show the eigenvalues $\lambda_\pm$ of Q are given by (1.4.12).

Exercise 3.2.4 Let $A = \begin{pmatrix} 1 & -1 \\ 2 & 4 \end{pmatrix}$. Find the eigenvalues and eigenvectors of A. Verify that the trace and determinant of A are the sum and product of the eigenvalues.

Exercise 3.2.5 Let $A = \begin{pmatrix} a & b \\ -b & a \end{pmatrix}$. Show the eigenvalues are

$$\lambda_\pm = a \pm ib$$

(these are complex numbers §A.5).

Exercise 3.2.6 Let Q be a 2×2 symmetric matrix. Show $\det(Q)$ and $\text{trace}(Q)$ are the product and sum of the eigenvalues. Conclude Q is a positive variance matrix when $\det(Q) > 0$ and $\text{trace}(Q) > 0$.

Exercise 3.2.7 The symmetric 2×2 matrix Q has eigenvalues 1 and 2 and corresponding eigenvectors $u = (3, 4)$ and $v = u^{\perp}$ ($u^{\perp}$ is defined in §A.4). What is Q?

Exercise 3.2.8 [35] Let

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}.$$

Find all four eigenvalues and eigenvectors of A by solving $Av = \lambda v$.

Exercise 3.2.9 [35] Let

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}.$$

Find all four eigenvalues and eigenvectors of A by solving $Av = \lambda v$.

Exercise 3.2.10 Verify (3.2.4) and (3.2.5) using `sympy`.

Exercise 3.2.11 With $R(d)$ as in Exercise **2.2.10**, find the eigenvalues and eigenvectors of $R(d)$.

Exercise 3.2.12 Use Python to verify the entries in Table 3.1.

d	$4 \cdot \text{trace}(Q(d)^{+})$
4	4+1
16	(4+1)(16+1)
256	(4+1)(16+1)(256+1)

Table 3.1 Trace of pseudo-inverse (§2.3) of $Q(d)$.

Exercise 3.2.13 Verify (3.2.13). Conclude $Q(d)$ is nonnegative, hence a variance matrix.

Exercise 3.2.14 Let P be a projection matrix (§2.7). Show the eigenvalues of P are 0 and 1. Which vectors are eigenvectors for 1, and which for 0?

Exercise 3.2.15 Let a, b, c be three distinct eigenvalues of a matrix A, with corresponding eigenvectors u, v, w. Show that u, v, w are linearly independent. Use Exercise **A.5.9**: there is a quadratic $p(t) = \alpha t^2 + \beta t + \gamma$ satisfying $p(a) = 0$, $p(b) = 0$, $p(c) = 1$. Multiply $ru + sv + tw = 0$ by $p(A) = \alpha A^2 + \beta A + \gamma I$ to conclude $t = 0$, and repeat for r and s.

Exercise 3.2.16 Let Q be a symmetric matrix with positive eigenvalues. Use the Cauchy-Schwarz inequality (2.2.3) to show

$$(u \cdot Qu)(u \cdot Q^{-1}u) \geq 1$$

for every unit vector u. First assume Q is diagonal then use EVD.

Exercise 3.2.17 If Q is a symmetric matrix with positive top and bottom eigenvalues L and m, show

$$m \leq u \cdot Qu \leq L, \qquad \text{and} \qquad \frac{1}{L} \leq u \cdot Q^{-1}u \leq \frac{1}{m}$$

for every unit vector u.

Exercise 3.2.18 If Q is a symmetric matrix with positive top and bottom eigenvalues L and m, show

$$\frac{m}{L} \leq (u \cdot Qu)(u \cdot Q^{-1}u) \leq \frac{L}{m}$$

for every unit vector u.

3.3 Graphs

Graph theory is a kind of linear geometry, and depends on the material already covered. As such, the study of graphs is an application of the material in the previous sections. Since graph theory is the start of neural networks, we study it here.

A *graph* consists of *nodes* and *edges*. The nodes are also called *vertices*. For example, the graphs in Figure 3.12 each have four nodes and three edges. The left graph is *directed*, in that a direction is specified for each edge. The graph on the right is *undirected*, no direction is specified.



Fig. 3.12 Directed and undirected graphs.

In a directed graph, if there is an edge pointing *from node i to node j, we say (i, j) is an edge.* For directed or undirected graphs, if there is an edge connecting nodes i and j, we say i and j are *adjacent.*

An edge is *weighed* if a scalar w is attached to it. By convention, we denote the weight on the edge (j, i) by w_{ij}. To repeat, *w_{ij} is the weight on the edge pointing from node j to node i.*

If every edge in a graph is weighed, then the graph is a *weighed graph*. Any two nodes may be considered adjacent by assigning the weight zero to the edge between them.

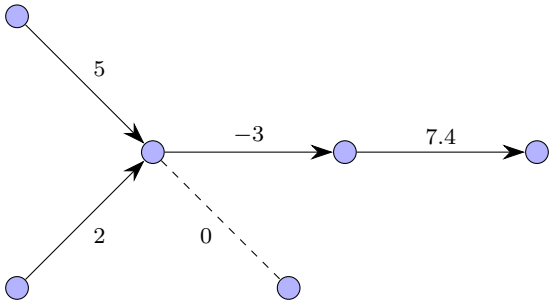


Fig. 3.13 A weighed directed graph.

In §4.4, back propagation on weighed directed graphs is used to calculate derivatives.

Let w_{ij} be the weight on the edge (j, i) in a weighed directed graph. The *weight matrix* of a weighed directed graph is the matrix $W = (w_{ij})$.

If the graph is unweighed, then we set $A = (a_{ij})$, where

$$a_{ij} = \begin{cases} 1, & \text{if } i \text{ and } j \text{ adjacent,} \\ 0, & \text{if not.} \end{cases} .$$

In this case, A consists of ones and zeros, and is called the *adjacency matrix*. If the graph is undirected, then the adjacency matrix is symmetric,

$$a_{ij} = a_{ji}.$$

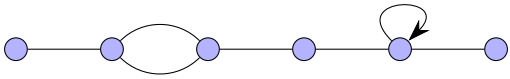


Fig. 3.14 A double edge and a loop.

Sometimes graphs may have multiple edges between nodes, or loops, which are edges starting and ending at the same node. A graph is *simple* if it has no loops and no multiple edges. In this section, we deal only with *simple undirected unweighed graphs*.

To summarize, a simple undirected graph $G = (V, E)$ is a collection V of nodes, and a collection of edges E, each edge corresponding to a pair of nodes.

The number of nodes is the *order* n of the graph, and the number of edges is the *size* m of the graph. In a (simple undirected) graph of order n, the number of pairs of nodes is n-choose-2, so the number of edges satisfies

$$0 \leq m \leq \binom{n}{2} = \frac{1}{2}n(n-1).$$

How many graphs of order n are there? Since graphs are built out of edges, the answer depends on how many subsets of edges you can grab from a maximum of $n(n-1)/2$ edges. The number of subsets of a set with m elements is 2^m, so the number G_n of graphs with n nodes is

$$G_n = 2^{\binom{n}{2}} = 2^{n(n-1)/2}.$$

For example, the number of graphs with $n = 5$ is $2^{5(5-1)/2} = 2^{10} = 1,024$, and the number of graphs with $n = 10$ is

$$n = 10 \qquad \Longrightarrow \; G_n = 2^{45} = 35,184,372,088,832.$$

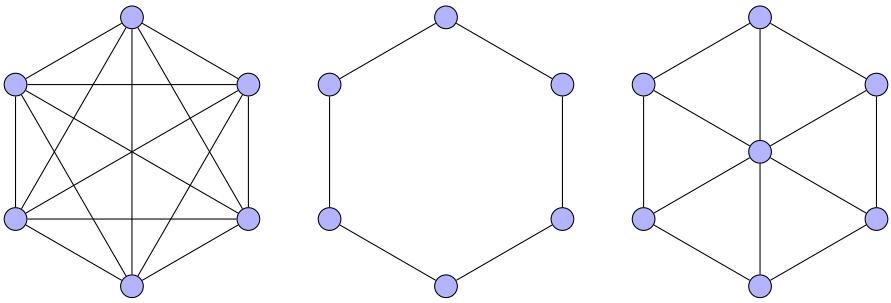


Fig. 3.15 The complete graph K_6, the cycle graph C_6, and the wheel graph W_6.

When $m = 0$, there are no edges, and we say the graph is *empty*. When $m = n(n-1)/2$, there are the maximum number of edges, and we say the graph is *complete*. The *complete graph* with n nodes is written K_n (Figure 3.15).

The *cycle graph* C_n with n nodes is as in Figure 3.15. The graph C_n has n edges. The *wheel graph* is the cycle graph with one vertex added at the center and connected to the spokes. The cycle graph C_3 is a triangle.



A graph G' is a *subgraph* of a graph G if every node of G' is a node of G, and every edge of G' is an edge of G. For example, *a triangle in* G is a graph triangle that is a subgraph of G. Below we see the graph K_6 in Figure 3.15 contains twenty triangles.

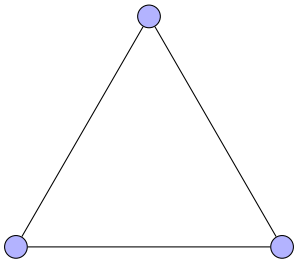


Fig. 3.16 The triangle $K_3 = C_3$.

Let v be a node in a (simple, undirected) graph G. The *degree* of v is the number d_v of edges containing v. If the nodes are labeled 1, 2, ..., n, with the degrees in decreasing order, then

$$d_1 \geq d_2 \geq d_3 \geq \cdots \geq d_n$$

is the *degree sequence* of the graph. We write

$$(d_1, d_2, d_3, \ldots, d_n)$$

for the degree sequence.

If the graph is directed, the *in-degree* is the number of incoming edges, and the *out-degree* is the number of outgoing edges. If a node has no incoming edges, it is an *input node*. If a node has no outgoing edges, it is an *output node*. Unless specified explicitly, all graphs in this section are undirected and unweighed.

If we add the degrees over all nodes, we obtain the number of edges counted twice, because each edge contains two nodes. Thus we have

Handshaking Lemma

If the order is n, the size is m, and the degrees are d_1, d_2, ..., d_n, then

$$d_1 + d_2 + \cdots + d_n = \sum_{k=1}^{n} d_k = 2m.$$

A node is *isolated* if its degree is zero. A node is *dominating* if it has the highest degree. Notice the highest degree is $\leq n - 1$, because there are no loops. We show

Nodes with Equal Degree

In any graph, there are at least two nodes with the same degree.

To see this, we consider two cases. First case, assume there are no isolated nodes. Then the degree sequence is

$$n - 1 \geq d_1 \geq d_2 \geq \cdots \geq d_n \geq 1.$$

So we have n integers spread between 1 and $n - 1$. This can't happen unless at least two of these integers are equal. This completes the first case. In the second case, we have at least one isolated node, so $d_n = 0$. If $d_{n-1} = 0$ also, then we have found two nodes with the same degree. If not, then the maximum degree is $n - 2$ (because node n is isolated), and

$$n - 2 \geq d_1 \geq d_2 \geq \ldots d_{n-1} \geq 1.$$

So now we have $n-1$ integers spread between 1 and $n-2$. This can't happen unless at least two of these integers are equal. This completes the second case.



A graph is *regular* if all the node degrees are equal. If the node degrees are all equal to k, we say the graph is k-regular. From the handshaking lemma, for a k-regular graph, we have $kn = 2m$, so

$$m = \frac{1}{2}kn.$$

For example, because $2m$ is even, there are no 3-regular graphs with 11 nodes. Both K_n and C_n are regular, with K_n being $(n - 1)$-regular, and C_n being 2-regular.

A *walk* on a graph is a sequence of nodes v_1, v_2, v_3, ... where each consecutive pair v_i, v_{i+1} of nodes are adjacent. For example, if v_1, v_2, v_3,

v_4, v_5, v_6 are the nodes (in any order) of the complete graph K_6, then $v_1 \to v_2 \to v_3 \to v_4 \to v_2$ is a walk. A *path* is a walk with no backtracking: A path visits each node at most once.

Two nodes a and b are *connected* if there is a walk starting at a and ending at b. If a and b are connected, then there is a path starting at a and ending at b, since we can cut out the cycles of the walk. A graph is *connected* if every two nodes are connected. A graph is *disconnected* if it is not connected. For example, Figure 3.15 may be viewed as two connected graphs K_6 and C_6, or a single disconnected graph $K_6 \cup C_6$.

A *closed walk* is a walk that ends where it starts. A *cycle* is a closed path. If a graph has no cycles, it is a *forest*. A connected forest is a *tree*. In a tree, any two nodes are connected by exactly one path.



The adjacency matrix of a graph is given by

$$a_{ij} = \begin{cases} 1, & \text{if } i \text{ and } j \text{ are adjacent,} \\ 0, & \text{if not.} \end{cases}$$

When a graph is undirected, its adjacency matrix is symmetric.

Let $\mathbf{1}$ be the vector $\mathbf{1} = (1, 1, 1, \ldots, 1)$. The adjacency matrix of the complete graph K_n is the $n \times n$ matrix A with all ones except on the diagonal. If I is the $n \times n$ identity matrix, then this adjacency matrix is

$$A = \mathbf{1} \otimes \mathbf{1} - I$$

For example, for the triangle K_3,

$$A = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}.$$

If we label the nodes of the cycle graph C_n consecutively, then node i is shares an edge with $i - 1$ and $i + 1$, except when $i = 1$ and $i = n$. Node 1 shares an edge with 2 and n, and node n shares an edge with $n - 1$ and 1. So for C_6 the adjacency matrix is

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Notice there are ones on the sub-diagonal, and ones on the super-diagonal, and ones in the upper-right and lower-left corners.

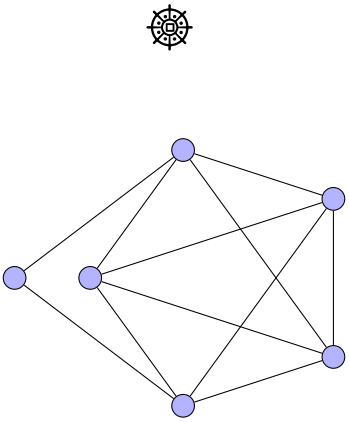


Fig. 3.17 An Eulerian graph.

The *Seven Bridges of Königsberg* was an important technology problem in the eighteenth century. The following description and map are from Wikipedia [38].

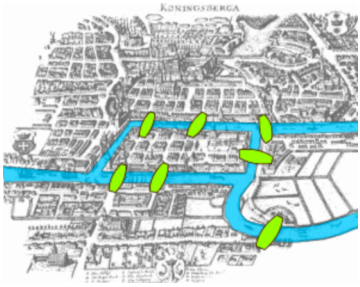


Fig. 3.18 An eighteenth-century map of Königsberg showing the seven bridges.

The city of Königsberg in Prussia (now Kaliningrad, Russia) was set on both sides of the Pregel River, and included two large islands — Kneiphof and Lomse — which were connected to each other, and to the two mainland portions of the city, by seven bridges.

The problem was to devise a path that would cross each of those bridges once and only once. The problem's negative resolution by Leonhard Euler[2] [pronounced "oy-ler"], in 1736, laid the foundations of graph theory.

We say a path is *Eulerian* [pronounced "oy-ler-yan"] if it visits *every edge* in the graph *exactly once*. An *Eulerian cycle* is an Eulerian path that is a cycle. Here is Euler's result.

[2] This is the Euler who named e §A.3.

Euler's Theorem

A connected graph has an Eulerian cycle if and only if every vertex has an even degree.

If a graph has an Eulerian cycle, it is an *Eulerian graph*. Thus the Eulerian graphs are those where all nodes have even degree.

Since the degree sequence of the graph in Figure 3.17 is $(4, 4, 4, 4, 4, 2)$, the graph is Eulerian.



For any adjacency matrix A, the sum of each row is equal to the degree of the node corresponding to that row. This is the same as saying

$$A\mathbf{1} = \begin{pmatrix} d_1 \\ d_2 \\ \dots \\ d_n \end{pmatrix}.$$

In particular, for a k-regular graph, we have

$$A\mathbf{1} = k\mathbf{1},$$

so *for a k-regular graph, k is an eigenvalue of A.*

What is the connection between degrees and eigenvalues in general? To explain this, let λ be an eigenvalue of A with eigenvector $v = (v_1, v_2, \dots, v_n)$, so $Av = \lambda v$. Since a multiple tv of v is also an eigenvector, we may assume the biggest component of v equals 1. Suppose the nodes are labeled so that $v = (1, v_2, v_3, \dots, v_n)$, with

$$v_1 = 1 \geq |v_j|, \qquad j = 2, 3, \dots, n.$$

Taking the first component of $Av = \lambda v$, we have

$$(Av)_1 = a_{11}v_1 + a_{12}v_2 + a_{13}v_3 + \cdots + a_{1n}v_n.$$

Since the sum $a_{11} + a_{12} + \cdots + a_{1n}$ equals the degree d_1 of node 1, this implies

$$d_1 = \sum_{j=1}^{n} a_{1j} =\geq \sum_{j=1}^{n} a_{1j}v_j = (Av)_1 = \lambda v_1 = \lambda.$$

Since d_1 is one of the degrees, d_1 is no greater than the maximum degree. This explains

Maximum Degree of Graph

If λ is any eigenvalue of the adjacency matrix A, then λ is less or equal to the maximum degree.

In particular, for a k-regular graph, the maximum degree equals k, and we already saw k is an eigenvalue, so

Top Eigenvalue

For a k-regular graph, k is the top eigenvalue of the adjacency matrix A.

Let $A = \mathbf{1} \otimes \mathbf{1} - I$ be the adjacency matrix of complete graph K_n. Then for any vector v orthogonal to $\mathbf{1}$,

$$Av = (\mathbf{1} \otimes \mathbf{1} - I)v = (\mathbf{1} \cdot v)\mathbf{1} - v = 0 - v = -v,$$

so $\lambda = -1$ is an eigenvalue with multiplicity $n - 1$. Since

$$A\mathbf{1} = (\mathbf{1} \cdot \mathbf{1})\mathbf{1} - \mathbf{1} = n\mathbf{1} - \mathbf{1} = (n-1)\mathbf{1},$$

$n - 1$is an eigenvalue. Hence the eigenvalues of A are $n - 1$ with multiplicity 1 and -1 with multiplicity $n - 1$.



The *complement* of graph G is the graph $\bar{G}$ obtained by switching 1's and 0's, so the adjacency matrix $\bar{A}$ of $\bar{G}$ is

$$\bar{A} = A(\bar{G}) = \mathbf{1} \otimes \mathbf{1} - I - A(G).$$

Let G be a k-regular graph, and suppose $k = \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ are the eigenvalues of $A = A(G)$. Since A is symmetric, we have an orthogonal basis of eigenvectors $v_1, v_2, \ldots, v_n$, with $v_1 = \mathbf{1}$. Then $\bar{G}$ is an $(n - 1 - k)$-regular graph, so the top eigenvalue of $\bar{A} = A(\bar{G})$ is $n-1-k$, with eigenvector $v_1 = \mathbf{1}$. If v_k is any eigenvector of A other than $\mathbf{1}$, then v_k is orthogonal to $\mathbf{1}$, hence

$$\bar{A}v = (\mathbf{1} \otimes \mathbf{1} - I - A)v_k = -v - \lambda_k v_k = (-1 - \lambda_k)v_k.$$

Hence the eigenvalues of $\bar{A}$ are $n - 1 - k$ and $-1 - \lambda_k$, $k = 2, \ldots, n$, with the same eigenbasis.



Now we look at powers of the adjacency matrix A. By definition of matrix-matrix multiplication,

$$(A^2)_{ij} = i\text{-th row} \times j\text{-th column} = \sum_{k=1}^{n} a_{ik}a_{kj}.$$

Now $a_{ik}a_{kj}$ is either 0 or 1, and equals 1 exactly if there is a 2-step path from i to j. Hence

$$(A^2)_{ij} = \text{number of 2-step walks connecting } i \text{ and } j.$$

Notice a 2-step walk between i and j is the same as a 2-step path between i and j.

When $i = j$, $(A^2)_{ii}$ is the number of 2-step paths connecting i and i, which means number of edges. Since this counts edges twice, we have

$$\frac{1}{2}\text{trace}(A^2) = m = \text{number of edges}.$$

Similarly, $(A^3)_{ij}$ is the number of 3-step walks connecting i and j. Since a 3-step walk from i to i is the same as a triangle, $(A^3)_{ii}$ is the number of triangles in the graph passing through i. Since the trace is the sum of the diagonal elements, $\text{trace}(A^3)$ counts the number of triangles. But this overcounts by a factor of $3! = 6$, since three labels may be rearranged in six ways. Hence

$$\frac{1}{6}\text{trace}(A^3) = \text{number of triangles}.$$

Loops, Edges, Triangles

Let A be the adjacency matrix. Then

- $\text{trace}(A) = \text{number of loops} = 0$,
- $\text{trace}(A^2) = 2 \times \text{number of edges}$,
- $\text{trace}(A^3) = 6 \times \text{number of triangles}$.

Let us compute these for the complete graph K_n. Since

$$(u \otimes v)^2 = (u \otimes v)(u \otimes v) = (u \cdot v)(u \otimes v),$$

and $\mathbf{1} \cdot \mathbf{1} = n$, we have $(\mathbf{1} \otimes \mathbf{1})^2 = n\mathbf{1} \otimes \mathbf{1}$. So

$$A^2 = (\mathbf{1} \otimes \mathbf{1} - I)^2 = (\mathbf{1} \otimes \mathbf{1})^2 - 2\mathbf{1} \otimes \mathbf{1} + I = (n-2)\mathbf{1} \otimes \mathbf{1} + I.$$

Since $\text{trace}(u \otimes v) = u \cdot v$, we have $\text{trace}(\mathbf{1} \otimes \mathbf{1}) = n$. Hence

$$\text{trace}(A^2) = \text{trace}((n-2)\mathbf{1} \otimes \mathbf{1} + I) = n(n-2) + n = n(n-1).$$

This is correct because for a complete graph, $n(n-1)/2$ is the number of edges.

Continuing,

$$\begin{aligned} A^3 = A^2A &= ((n-2)\mathbf{1}\otimes\mathbf{1} + I)(\mathbf{1}\otimes\mathbf{1} - I) \\ &= n(n-2)\mathbf{1}\otimes\mathbf{1} - (n-2)\mathbf{1}\otimes\mathbf{1} + \mathbf{1}\otimes\mathbf{1} - I \\ &= (n^2 - 3n + 3)\mathbf{1}\otimes\mathbf{1} - I. \end{aligned}$$

From this, we get

$$\operatorname{trace}(A^3) = n(n^2 - 3n + 3) - n = n(n^2 - 3n + 2) = n(n-1)(n-2).$$

This is correct because for a complete graph, we have a triangle whenever we have a triple of nodes, and there are n-choose-3 triples, which equals $n(n-1)(n-2)/6$.

Remember, a graph is connected if there is a walk connecting any two nodes. Since there is a 4-step walk between i and j exactly when there are r, s, and t satisfying

$$a_{ir}a_{rs}a_{st}a_{tj} = 1,$$

we see there is a 4-step walk connecting i and j if $(A^4)_{ij} > 0$. Hence

Connected Graph

Let A be the adjacency matrix. Then the graph is connected if for every $i \neq j$, there is a k with $(A^k)_{ij} > 0$.

If a graph is not connected, it may be decomposed as a disjoint union of two or more connected subgraphs. These subgraps are the *components* of the graph.



Two graphs are *isomorphic* if a re-labeling of the nodes in one makes it identical to the other. To explain this, we need permutations.

A *permutation* on n letters is a re-arrangement of 1, 2, 3,..., n. Here are two permutations of $(1, 2, 3, 4)$,

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix}, \qquad \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 2 \end{pmatrix}.$$

There are $n!$ permutations of $(1, 2, \ldots, n)$. If a permutation sends i to j, we write $i \to j$. Since a permutation is just a re-labeling, if $i \to k$ and $j \to k$, then we must have $i = j$.

Each permutation leads to a permutation matrix. A *permutation matrix* is a matrix of zeros and ones, with only one 1 in any column or row (see also Exercise **2.1.1**). For example, the above permutations correspond to the 4×4 matrices

$$P = \begin{pmatrix} 0&0&0&1 \\ 0&0&1&0 \\ 0&1&0&0 \\ 1&0&0&0 \end{pmatrix} \qquad P = \begin{pmatrix} 0&0&0&1 \\ 0&0&1&0 \\ 1&0&0&0 \\ 0&1&0&0 \end{pmatrix}.$$

In general, the permutation matrix P has $P_{ij} = 1$ if $i \to j$, and $P_{ij} = 0$ if not. If P is any permutation matrix, then $P_{ik}P_{jk}$ equals 1 if both $i \to k$ and $j \to k$. In other words, $P_{ik}P_{jk} = 1$ if $i = j$ and $i \to k$, and $P_{ik}P_{jk} = 0$ otherwise. Since $i \to k$ for exactly one k,

$$(PP^t)_{ij} = \sum_{k=1}^{n} P_{ik}P^t_{kj} = \sum_{k=1}^{n} P_{ik}P_{jk} = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases}$$

Hence P is orthogonal,

$$PP^t = I, \qquad P^{-1} = P^t.$$

Using permutation matrices, we can say two graphs are isomorphic if their adjacency matrices A, A' satisfy

$$A' = PAP^{-1} = PAP^t$$

for some permutation matrix P.

If two graphs are isomorphic, then it is easy to check their degree sequences are equal. However, the converse is not true. Figure 3.19 displays two non-isomorphic graphs with degree sequences $(3, 2, 2, 1, 1, 1)$. These graphs are non-isomorphic because in one graph, there are two degree-one nodes adjacent to a degree-three node, while in the other graph, there is only one degree-one node adjacent to a degree-three node.

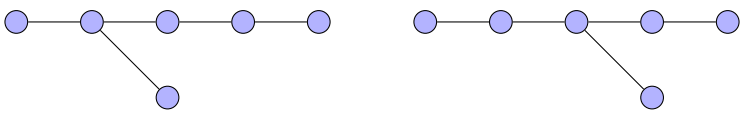


Fig. 3.19 Non-isomorphic graphs with degree sequence $(3, 2, 2, 1, 1, 1)$.



A graph is *bipartite* if the nodes can be divided into two groups, with adjacency only between nodes across groups. If we call the two groups even and odd, then odd nodes are never adjacent to odd nodes, and even nodes are never adjacent to even nodes.

The *complete bipartite* graph is the bipartite graph with maximum number of edges: Every odd node is adjacent to every even node. The complete

bipartite graph with n odd nodes with m even nodes is written $K_{n,m}$. Then the order of $K_{n,m}$ is $n+m$.

Let $a = (1, 1, \ldots, 1, 0, 0, \ldots, 0)$ be the vector with n ones and m zeros, and let $b = \mathbf{1} - a$. Then b has n zeros and m ones, and the adjacency matrix of $K_{n,m}$ is

$$A = A(K_{n,m}) = a \otimes b + b \otimes a.$$

For example, the adjacency matrix of $K_{5,3}$ is

$$\begin{pmatrix}1\\1\\1\\0\\0\\0\\0\\0\end{pmatrix} \otimes \begin{pmatrix}0\\0\\0\\1\\1\\1\\1\\1\end{pmatrix} + \begin{pmatrix}0\\0\\0\\1\\1\\1\\1\\1\end{pmatrix} \otimes \begin{pmatrix}1\\1\\1\\0\\0\\0\\0\\0\end{pmatrix} = \begin{pmatrix}0&0&0&1&1&1&1&1\\0&0&0&1&1&1&1&1\\0&0&0&1&1&1&1&1\\1&1&1&0&0&0&0&0\\1&1&1&0&0&0&0&0\\1&1&1&0&0&0&0&0\\1&1&1&0&0&0&0&0\\1&1&1&0&0&0&0&0\end{pmatrix}.$$

Recall we have

$$(a \otimes b)v = (b \cdot v)a.$$

From this, we see the column space of $A = a \otimes b + b \otimes a$ is $span(a, b)$. Thus the rank of A is 2, and the nullspace of A consists of the orthogonal complement $span(a, b)^{\perp}$ of $span(a, b)$. Using this, we compute the eigenvalues of A.

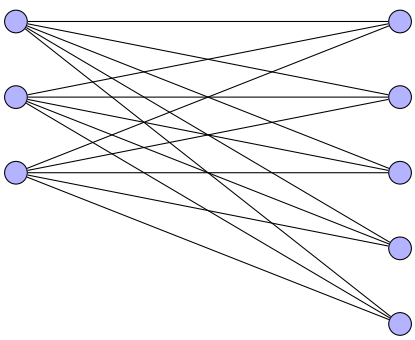


Fig. 3.20 Complete bipartite graph $K_{5,3}$.

Since the nullspace is $span(a, b)^{\perp}$, any vector orthogonal to a and to b is an eigenvector for $\lambda = 0$. Hence the eigenvalue $\lambda = 0$ has multiplicity $n + m - 2$. Since $\text{trace}(A) = 0$, the sum of the eigenvalues is zero, and the remaining two eigenvalues are $\pm\lambda \neq 0$.

Let v be an eigenvector for $\lambda \neq 0$. Because eigenvectors corresponding to distinct eigenvalues of a symmetric matrix are orthogonal (see §3.2), v is orthogonal to the nullspace of A, so v must be a linear combination of a and

b, $v = ra + sb$. Since $a \cdot b = 0$,

$$Aa = nb, \qquad Ab = ma.$$

Hence

$$\lambda v = Av = A(ra + sb) = rnb + sma.$$

Applying A again,

$$\lambda^2 v = A^2 v = A(rnb + sma) = rnma + smnb = nm(ra + sb) = nmv.$$

Hence $\lambda = \sqrt{nm}$. We conclude the eigenvalues of $K_{n,m}$ are

$$\sqrt{nm}, 0, 0, \ldots, 0, -\sqrt{nm}, \qquad \text{(with 0 repeated } n + m - 2 \text{ times)}.$$

For example, for the graph in Figure 3.20, the nonzero eigenvalues are $\lambda = \pm\sqrt{3 \times 5} = \pm\sqrt{15}$.



Let G be a graph with n nodes and m edges. The *incidence matrix* of G is a matrix whose rows are indexed by the edges, and whose columns are indexed by the nodes. Therefore, the incidence matrix has shape $m \times n$.

By placing arrows along the edges, we can make G into a directed graph. In a directed graph, each edge has a *tail node* and a *head node*. Then the incidence matrix is given by

$$B_{ij} = \begin{cases} 1, & \text{if node } j \text{ is the head of edge } i, \\ -1, & \text{if node } j \text{ is the tail of edge } i, \\ 0, & \text{if node } j \text{ is not on edge } i. \end{cases}$$

The *laplacian* of a graph G is the symmetric $n \times n$ matrix

$$L = B^t B.$$

Both the laplacian matrix and the adjacency matrix are $n \times n$. What is the connection between them?

Laplacian

The laplacian satisfies

$$L = D - A,$$

where $D = \text{diag}(d_1, d_2, \ldots, d_n)$ is the diagonal degree matrix.

Note the Laplacian does not depend on how the edges were directed, it only depends on A.

For example, for the cycle graph C_6, the degree matrix is $2I$, and the laplacian is the matrix we saw in §3.2,

$$L = Q(6) = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 & -1 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ -1 & 0 & 0 & 0 & -1 & 2 \end{pmatrix}.$$

In a k-regular graph, the Laplacian is $L = kI - A$, so the eigenvalues of A and L are related by $\lambda \to k - \lambda$.



Let A be the adjacency matrix of the cycle graph C_n. Since C_n is 2-regular, the top eigenvalue of A is 2. Since

$$A = 2I - Q(n),$$

by (3.2.15), the eigenvalues of A are

$$2\cos(2\pi k/n), \qquad k = 0, 1, 2, \ldots, n-1,$$

and the eigenvectors of A are the eigenvectors of $Q(n)$.

Exercises

Exercise 3.3.1 [36] Which of the following degree sequences are possible for a simple graph?

1. (4,4,4,3,3,3,3,2)
2. (5,3,2,2,2,1)
3. (8,7,6,5,5,3,1,1)
4. (6,6,6,5,4,4,4,3,3,3)

Exercise 3.3.2 [36] Consider the graph in Figure 3.21 below. What is the order of the graph? What is the degree of vertex G? What is the degree of vertex C? How many components does the graph have?

Exercise 3.3.3 [36] Construct a tree with five vertices such that the degree of one vertex is 3. How many such (non-isomorphic) graphs are there?

Exercise 3.3.4 [36] Consider the cities F, G, H, I, J, K. The costs of the possible roads between cities are $c(F,G) = 8$, $c(G,H) = 5$, $c(F,H) = 9$, $c(F,K) = 10$, $c(I,K) = 6$, $c(I,J) = 7$. What is the minimum cost to build a road system that connects all the cities?

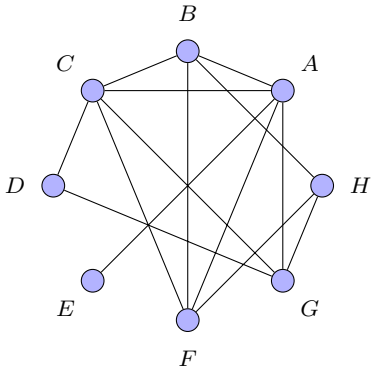


Fig. 3.21 A graph.

Exercise 3.3.5 [36] A graph has vertices M, N, O, P, Q, R, S, and edges MN, MO, MP, MR, MS, NO, OR, PR, PS, QR. What is the degree of S? What is the degree of M? How many components does the graph have?

Exercise 3.3.6 [36] Find the degree sequences of the cycle graph C_6, the complete graph K_8, the complete bipartite graph $K_{3,7}$, and the wheel graph W_4.

Exercise 3.3.7 [36] A directed graph has vertices v_0, v_1, v_2, v_3, v_4, v_5, v_6 with adjacency matrix

$$\begin{pmatrix} 0&1&1&0&1&0&0\\ 0&0&0&1&0&0&0\\ 0&1&0&1&0&0&1\\ 1&0&0&0&0&1&1\\ 0&1&1&0&0&0&0\\ 1&0&1&0&0&0&1\\ 1&1&0&0&1&0&0 \end{pmatrix}.$$

What is the order of the graph? What is the number of edges? What are the in-degree and out-degree of v_3? of v_5?

Exercise 3.3.8 Which of C_n, K_n, W_n is Eulerian? For which n?

Exercise 3.3.9 Find an Eulerian cycle in the graph in Figure 3.17.

3.4 Singular Value Decomposition

In this section, we discuss the singular value decomposition (U, S, V) of a matrix A.

Let A be a matrix. We say a real number σ is a *singular value* of A if there are nonzero vectors v and u satisfying

$$Av = \sigma u \qquad \text{and} \qquad A^t u = \sigma v. \tag{3.4.1}$$

When this happens, v is a *right singular vector* and u is a *left singular vector* associated to σ.

When (3.4.1) holds, so does

$$Av = (-\sigma)(-u), \qquad A^t(-u) = (-\sigma)v.$$

Because of this, to eliminate ambiguity, it is standard to assume $\sigma \geq 0$; henceforth we shall insist singular values are positive or zero.

Contrast singular values with eigenvalues: While eigenvalues may be positive, negative, or zero, singular values are positive or zero, never negative.

The definition immediately implies

Singular Values of A Versus A Transpose

The singular values of A and the singular values of A^t are the same.

Contrast this with the analogous result for eigenvalues in §3.2.

We work out our first example. Let

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

Then $Av = \lambda v$ implies $\lambda = 1$ and $v = (1, 0)$. Thus A has only one eigenvalue equal to 1, and only one eigenvector. Set

$$Q = A^t A = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}.$$

Since Q is symmetric, Q has two eigenvalues λ_1, λ_2 and corresponding eigenvectors v_1, v_2. Moreover, as we saw in §3.2, v_1, v_2 may be chosen orthonormal.

The eigenvalues of Q are given by

$$0 = \det(Q - \lambda I) = \lambda^2 - 3\lambda + 1.$$

By the quadratic formula,

$$\lambda_1 = \frac{3}{2} + \frac{\sqrt{5}}{2} = 2.62, \qquad \lambda_2 = \frac{3}{2} - \frac{\sqrt{5}}{2} = 0.38.$$

Now we turn to singular values. If v and u and σ satisfy (3.4.1), then

$$Qv = A^t A v = A^t(\sigma u) = \sigma^2 v. \tag{3.4.2}$$

Hence $\sigma^2 = \lambda$, and we obtain

$$\sigma_1 = \sqrt{\frac{3}{2} + \frac{\sqrt{5}}{2}} = 1.62, \qquad \sigma_2 = \sqrt{\frac{3}{2} - \frac{\sqrt{5}}{2}} = 0.62.$$

To make (3.4.1) work, we set $u_1 = Av_1/\sigma_1$. Then $Av_1 = \sigma_1 u_1$, and

$$A^t u_1 = A^t A v_1/\sigma_1 = Qv_1/\sigma_1 = \lambda_1 v_1/\sigma_1 = \sigma_1 v_1.$$

Thus v_1, u_1 are right and left singular vectors corresponding to the singular value σ_1 of A. Similarly, if we set $u_2 = Av_2/\sigma_2$, then v_2, u_2 are right and left singular vectors corresponding to the singular value σ_2 of A.

We show v_1, v_2 are orthonormal, and u_1, u_2 are orthonormal. We already know v_1, v_2 are orthonormal, because they are orthonormal eigenvectors of the symmetric matrix Q. Also

$$\lambda_1 v_1 \cdot v_2 = Qv_1 \cdot v_2 = (A^t A v_1) \cdot v_2 = (Av_1) \cdot (Av_2) = \sigma_1 u_1 \cdot \sigma_2 u_2 = \sigma_1 \sigma_2 u_1 \cdot u_2.$$

Since $\sigma_1 \neq 0$, $\sigma_2 \neq 0$, $v_1 \cdot v_2 = 0$, it follows u_1, u_2 are orthogonal. Also

$$\lambda_1 = \lambda_1 v_1 \cdot v_1 = Qv_1 \cdot v_1 = (A^t A v_1) \cdot v_1 = (Av_1) \cdot (Av_1) = \sigma_1^2 u_1 \cdot u_1.$$

Since $\lambda_1 = \sigma_1^2$, $u_1 \cdot u_1 = 1$. Similarly, $u_2 \cdot u_2 = 1$. This shows u_1, u_2 are orthonormal, and completes the first example.



Let A be an $N \times d$ matrix, and suppose $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r$ are positive singular values with corresponding left singular vectors u_1, u_2, …, u_r and right singular vectors v_1, v_2, …, v_r. Then, since $u_k = Av_k/\sigma_k$, the vectors u_1, u_2, …, u_r are in the column space of A.

If u_1, u_2, …, u_r are linearly independent, it follows r is no larger than $rank(A)$, hence r is no larger than $\min(N, d)$. We seek the largest value of r. Below we show the largest r is $\min(N, d)$, the lesser of d and N.



The close connection between singular values σ of A and eigenvalues λ of $Q = A^t A$ carries over in the general case.

A Versus $Q = A^t A$

Let A be any matrix. Then

- the rank of A equals the rank of Q,
- σ is a singular value of A iff $\lambda = \sigma^2$ is an eigenvalue of Q.

Since the rank equals the dimension of the row space, the first part follows from §2.4. If $Av = \sigma u$ and $A^t u = \sigma v$, then

$$Qv = A^t Av = A^t(\sigma u) = \sigma A^t u = \sigma^2 v,$$

so $\lambda = \sigma^2$ is an eigenvalue of Q.

Conversely, If $Qv = \lambda v$, then $\lambda \geq 0$, so there are two cases. If $\lambda > 0$, set $\sigma = \sqrt{\lambda}$ and $u = Av/\sigma$. Then

$$Av = \sigma u, \qquad A^t u = A^t Av/\sigma = Qv/\sigma = \lambda v/\sigma = \sigma v$$

This shows σ is a singular value of A with singular vectors u and v.

If $\lambda = 0$, then we take $\sigma = 0$, and the correct interpretation of the second part is the nullspace of Q equals the nullspace of A, which we already know.

From §3.2, the number of positive eigenvalues (possibly repeated) of Q equals the rank of Q. By the above, we conclude *the rank of A equals the number of positive singular values (possibly repeated) of A.*



The above results may be phrased as

Singular Value Decomposition (SVD)

Let A be any matrix, and let r be the rank of A. Then there are r singular values σ_k, an orthonormal basis u_k of the target space, and an orthonormal basis v_k of the source space, such that

$$Av_k = \sigma_k u_k, \qquad A^t u_k = \sigma_k v_k, \qquad k \leq r, \tag{3.4.3}$$

and

$$Av_k = 0, \qquad A^t u_k = 0 \qquad \text{for } k > r. \tag{3.4.4}$$

Taken together, (3.4.3) and (3.4.4) say the number of nonzero singular values is exactly r. Assume A is $N \times d$, and let $p = \min(N, d)$ be the lesser of N and d.

Since (3.4.4) holds as long as there are vectors u_k and v_k, there are $p - r$ zero singular values. Hence there are $p = \min(N, d)$ singular values altogether.

The proof of the result is very simple once we remember the rank of Q equals the number of positive eigenvalues of Q. By the eigenvalue decomposition, there is an orthonormal basis v_k of the source space and positive eigenvalues λ_k such that $Qv_k = \lambda_k v_k$, $k \leq r$, and $Qv_k = 0$, $k > r$.

Setting $\sigma_k = \sqrt{\lambda_k}$ and $u_k = Av_k/\sigma_k$, $k \leq r$, as in our first example, we have (3.4.3), and, again as in our first example, u_k, $k \leq r$, are orthonormal.

By construction, v_k, $k > r$, is an orthonormal basis for the nullspace of A, and u_k, $k \leq r$, is an orthonormal basis for the column space of A.

Choose u_k, $k > r$, any orthonormal basis for the nullspace of A^t. Since the column space of A is the row space of A^t, the column space of A is the orthogonal complement of the nullspace of A^t (2.7.6). Hence u_k, $k \le r$, and u_k, $k > r$, are orthogonal. From this, u_k, $k \le r$, together with u_k, $k > r$, form an orthonormal basis for the target space.



For our second example, let a and b be nonzero vectors, possibly of different sizes, and let A be the matrix

$$A = a \otimes b, \qquad A^t = b \otimes a.$$

Let v and u be right and left singular vectors corresponding to a positive singular value σ. Then, by (2.2.11),

$$Av = (v \cdot b)a = \sigma u \qquad \text{and} \qquad A^t u = (u \cdot a)b = \sigma v.$$

Since $\sigma > 0$, it follows v is a multiple of b and u is a multiple of a. If we write $v = tb$ and $u = sa$ and plug in, we get

$$v = |a|\, b, \qquad u = |b|\, a, \qquad \sigma = |a|\, |b|.$$

Thus there is only one positive singular value of A, equal to $|a|\,|b|$. All other singular values are zero. This is not surprising since the rank of A is one.

Now think of the vector b as a single-row matrix B. Then, in a similar manner, one sees the only positive singular value of B is $\sigma = |b|$.

Our third example is

$$A = \begin{pmatrix} 0\,0\,0\,0 \\ 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \end{pmatrix}. \tag{3.4.5}$$

Then

$$A^t = \begin{pmatrix} 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \\ 0\,0\,0\,0 \end{pmatrix}, \qquad Q = A^t A = \begin{pmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,0 \end{pmatrix}$$

Since Q is diagonal symmetric, its rank is 3 and its eigenvalues are $\lambda_1 = 1$, $\lambda_2 = 1$, $\lambda_3 = 1$, $\lambda_4 = 0$, and its eigenvectors are

$$v_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, v_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, v_3 \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, v_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Clearly v_1, v_2, v_3, v_4 are orthonormal. By (3.4.2), $\sigma_1 = 1$, $\sigma_2 = 1$, $\sigma_3 = 1$, $\sigma_4 = 0$.

Since we must have $Av = \sigma u$, we can check that

$$u_1 = Av_1 = v_2, \quad u_2 = Av_2 = v_3, \quad u_3 = Av_3 = v_4, \quad u_4 = v_1$$

satisfies (3.4.1). This completes our third example.



Let A be $N \times d$, let U be the matrix with columns $u_1, u_2, \ldots, u_N$, and let V be the matrix with rows $v_1, v_2, \ldots, v_d$. Then V^t has columns $v_1, v_2, \ldots, v_d$. Because the u's and v's are orthonormal, U and V are orthogonal $N \times N$ and $d \times d$ matrices.

To see the implications of (3.4.3), for simplicity, assume A is 2×2. Then we have two singular values σ_1, σ_2, two right singular vectors v_1, v_2, and two left singular vectors u_1, u_2.

If we let $S = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix}$, then matrix-vector multiplication shows

$$AV^t = (Av_1, Av_2) = (\sigma_1 u_1, \sigma_2 u_2), \qquad US = (\sigma_1 u_1, \sigma_2 u_2).$$

Hence $AV^t = US$. Right-multiplying by V and using $V^tV = I$ implies the following result, which remains valid in general.

Diagonalization (SVD)

Let A be any matrix, with singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$. Let $v_1, v_2, \ldots$ be an orthonormal basis of right singular vectors of A in the source space, and let $u_1, u_2, \ldots$ be an orthonormal basis of left singular vectors of A in the target space. Let U be the orthogonal matrix with columns $u_1, u_2, \ldots$, and let V be the orthogonal matrix with rows $v_1, v_2, \ldots$. Let S be the diagonal matrix with the same shape as A and consisting of the singular values. Then

$$A = USV. \tag{3.4.6}$$

In more detail, suppose A is 4×6. Then we have an orthonormal basis v_1, v_2, v_3, v_4, v_5, v_6 of $\mathbf{R}^6$, and an orthonormal basis u_1, u_2, u_3, u_4 satisfying (3.4.3) with $r = 4$. If we set

$$S = \begin{pmatrix} \sigma_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_4 & 0 & 0 \end{pmatrix},$$

then U is 4×4 and V is 6×6, and we can verify directly that $A = USV$.

If A is 6×4, and

$$S = \begin{pmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 \\ 0 & 0 & 0 & \sigma_4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

then U is 6×6 and V is 4×4, and we can verify directly that $A = USV$. In either case, S has the same shape as A.

When $A = Q$ is a variance matrix, $Q \geq 0$, then the eigenvalues are non-negative, and, from (3.2.3), we have $UEU^t = Q$. If we choose $V = U^t$, we see EVD is a special case of SVD.

In general, however, if Q has negative eigenvalues, V is not equal to U^t; instead V is obtained from U^t by multiplying some of the columns of U by minus signs.



In `numpy`, `svd` returns the orthogonal matrices `U` and `V` and a vector `sigma` of singular values. The singular values are arranged in decreasing order. To recover the diagonal matrix `S`, we use `diag`.

```
from  numpy import *
set_printoptions(legacy = "1.25")

from  scipy.linalg import svd

u = array([1,2,3,4,5])
v = array([6,7,8,9,10])
w = array([11,12,13,14,15])

A = column_stack([u,v,w])

U, sigma, V = svd(A)
# sigma is a vector

# build diag matrix S
p = min(A.shape)
S = zeros(A.shape)
S[:p,:p] = diag(sigma)

print(U.shape,S.shape,V.shape)

allclose(A, dot(U, dot(S, V)))
```

This code returns `True`.



Given the relation between the singular values of A and the eigenvalues of $Q = A^t A$, we also can conclude

Right Singular Vectors Are the Same as Eigenvectors

Let A be any matrix and let $Q = A^t A$.

$$v \text{ is an eigenvector of } Q \iff v \text{ is a right singular vector of } A. \tag{3.4.7}$$

For example, if `dataset` is the Iris dataset (ignoring the labels), the code

```
from numpy import *
from scipy.linalg import svd,eigh
from sklearn import datasets

iris = datasets.load_iris()
dataset = iris["data"]

# center dataset
mu = mean(dataset, axis = 0)
A = dataset - mu
# rows of V are right
# singular values and vectors of A
sigma = svd(A)[1]
V = svd(A)[2]

# any of these will work
# because the eigenvectors are the same
Q = dot(A.T,A)
Q = cov(dataset.T,bias = False)
Q = cov(dataset.T,bias = True)

# columns of U are
# eigenvectors of Q
U = eigh(Q)[1]

#  columns of U
# and rows of V
# are the same
# except for order

U, V
```

returns

$$U = \begin{pmatrix} 0.36 & -0.66 & -0.58 & 0.32 \\ -0.08 & -0.73 & 0.6 & -0.32 \\ 0.86 & 0.18 & 0.07 & -0.48 \\ 0.36 & 0.07 & 0.55 & 0.75 \end{pmatrix}, V = \begin{pmatrix} 0.36 & -0.08 & 0.86 & 0.36 \\ -0.66 & -0.73 & 0.18 & 0.07 \\ 0.58 & -0.6 & -0.07 & -0.55 \\ 0.32 & -0.32 & -0.48 & 0.75 \end{pmatrix}$$

This shows the columns of U are identical to the rows of V, except for the third column of U, which is the negative of the third row of V.



Now we turn to the pseudo-inverse.

Pseudo-Inverse: Invert the Nonzero Singular Values

The pseudo-inverse A^+ is obtained by replacing, in the SVD diagonalization of A, the nonzero singular values of A by their reciprocals, and taking the transpose.

More explicitly, we can write

Pseudo-Inverse: Flip the Singular Vectors

Let A have rank r, and let σ_k be the nonzero singular values, and v_k, u_k the singular vectors as above. Then

$$A^+ u_k = \frac{1}{\sigma_k} v_k, \qquad (A^+)^t v_k = \frac{1}{\sigma_k} u_k, \qquad k = 1, 2, \ldots, r,$$

and

$$A^+ u_k = 0, \qquad (A^+)^t v_k = 0 \qquad \text{for } k > r.$$

We illustrate these results in the case of a diagonal matrix

$$S = \begin{pmatrix} a & 0 & 0 & 0 & 0 \\ 0 & b & 0 & 0 & 0 \\ 0 & 0 & c & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} = \left(\begin{array}{ccc|cc} a & 0 & 0 & 0 & 0 \\ 0 & b & 0 & 0 & 0 \\ 0 & 0 & c & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \end{array}\right) = \left(\begin{array}{c|cc} Q & 0 & 0 \\ & 0 & 0 \\ & 0 & 0 \\ \hline 0\ 0\ 0 & 0 & 0 \end{array}\right)$$

Since S is 4×5 and $SS^+S = S$, S^+ must be 5×4. Writing S^+ as blocks and applying the four properties of the pseudo-inverse S^+, leads to

$$S^+ = \left(\begin{array}{c|c} Q^{-1} & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ \hline \begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix} & \begin{matrix} 0 \\ 0 \end{matrix} \end{array}\right) = \begin{pmatrix} 1/a & 0 & 0 & 0 \\ 0 & 1/b & 0 & 0 \\ 0 & 0 & 1/c & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Exercises

Exercise 3.4.1 Let b be a vector and let B be the matrix with the single row b. Show $\sigma = |b|$ is the only positive singular value.

Exercise 3.4.2 Let Q be a symmetric matrix. When are the eigenvalues of Q equal to the singular values of Q? When are they not?

Exercise 3.4.3 In `sympy`, there is `Q.eigenvects()`, which returns the eigendata of a symmetric matrix Q. In `numpy`, `svd(A)` returns the singular data of a matrix A. Write a `sympy` function `svd(A)` that replicates this. (First assume all singular values of A are positive, then adjust the code for when A has zero singular values. Your final product should work for the zero matrix $A = 0$.)

Exercise 3.4.4 Let σ_1 and σ_2 be the greatest and least *nonzero* singular values of a matrix A, and let A^+ be the pseudo-inverse of A. Then

$$\sigma_2 \le |Au| \le \sigma_1, \qquad \frac{1}{\sigma_1} \le \left|(A^+)^t u\right| \le \frac{1}{\sigma_2},$$

for all unit vectors u in the row space of A (Exercise **2.7.12**).

Exercise 3.4.5 Let σ_1 and σ_2 be the greatest and least *nonzero* singular values of

$$A = \begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix}.$$

Using `sympy`, show

$$\frac{1}{2}\left(\frac{\sigma_1}{\sigma_2} + \frac{\sigma_2}{\sigma_1}\right) = \frac{62\sqrt{3}}{15}.$$

(See (3.2.4) and (3.2.5).)

3.5 Principal Component Analysis

Let Q be the variance matrix of a dataset in $\mathbf{R}^d$. Then Q is a $d \times d$ symmetric matrix, and the EVD guarantees an orthonormal basis $v_1, v_2, \ldots, v_d$ in $\mathbf{R}^d$ consisting of eigenvectors of Q,

$$Qv_k = \lambda_k v_k, \qquad k = 1, 2, \ldots, d.$$

These eigenvectors are the *principal components* of the dataset. Principal Component Analysis (PCA) consists of *projecting the dataset onto lower-dimensional spans of some of the eigenvectors.*

Let Q be a symmetric matrix with eigenvalue λ and corresponding eigenvector v, $Qv = \lambda v$. If t is a scalar, then the matrix tQ has eigenvalue $t\lambda$ and corresponding eigenvector v, since

$$(tQ)v = tQv = t\lambda v = (t\lambda)v.$$

Hence *multiplying Q by a scalar does not change the eigenvectors.*

Let A be the dataset matrix of a given dataset with N samples, and d features. If the samples are the rows of A, then A is $N \times d$. If we assume the dataset is centered, then, by (2.2.20), the variance is $Q = A^t A/N$. From the previous paragraph, the eigenvectors of the variance Q equal the eigenvectors of $A^t A$. From (3.4.7), these are the same as the right singular vectors of A.

Thus *the principal components of a dataset are the right singular vectors of the centered dataset matrix.* This shows there are two approaches to the principal components of a dataset: Either through EVD and eigenvectors of the variance matrix, or through SVD and right singular vectors of the centered dataset matrix. We shall do both.

Assuming the eigenvalues are ordered top to bottom,

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d,$$

in PCA one takes the most significant components, those components who eigenvalues are near the top eigenvalue. For example, one can take the top two eigenvalues $\lambda_1 \geq \lambda_2$ and their eigenvectors v_1, v_2, and project the dataset onto $span(v_1, v_2)$. The projected dataset can then be visualized as points in the plane. Similarly, one can take the top three eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$ and their eigenvectors v_1, v_2, v_3 and project the dataset onto $span(v_1, v_2, v_3)$. This can then be visualized as points in three dimensions.

Recall the MNIST dataset consists of $N = 60000$ points in $d = 784$ dimensions. After we download the dataset,

```
from pandas import *
from numpy import *
set_printoptions(legacy = "1.25")

mnist = read_csv("mnist.csv").to_numpy()

dataset = mnist[:,1:]
labels = mnist[:,0]

Q = cov(dataset.T)
totvar = Q.trace()
```

we compute Q, the total variance, and the eigenvalues, as percentages of the total variance. We also name the targets as `labels` for later use.

```
array([[ 9.705,  9.705],
       [ 7.096, 16.801],
       [ 6.169, 22.97 ],
       [ 5.389, 28.359],
       [ 4.869, 33.228],
       [ 4.312, 37.54 ],
       [ 3.272, 40.812],
       [ 2.884, 43.696],
       [ 2.762, 46.458],
       [ 2.357, 48.815],
       [ 2.109, 50.924],
       [ 2.023, 52.947],
       [ 1.716, 54.663],
       [ 1.692, 56.355],
       [ 1.579, 57.934],
       [ 1.483, 59.417],
       [ 1.325, 60.741],
       [ 1.277, 62.018],
       [ 1.187, 63.205],
       [ 1.153, 64.358]])
```

Fig. 3.22 MNIST eigenvalues as a percentage of the total variance.

The left column in Figure 3.22 lists the top twenty eigenvalues as a percentage of their sum. For example, the top eigenvalue λ_1 is around 10% of the total variance. The right column lists the cumulative sums of the eigenvalues, so the third entry in the right column is the sum of the top three eigenvalues, $\lambda_1 + \lambda_2 + \lambda_3 = 22.97\%$.

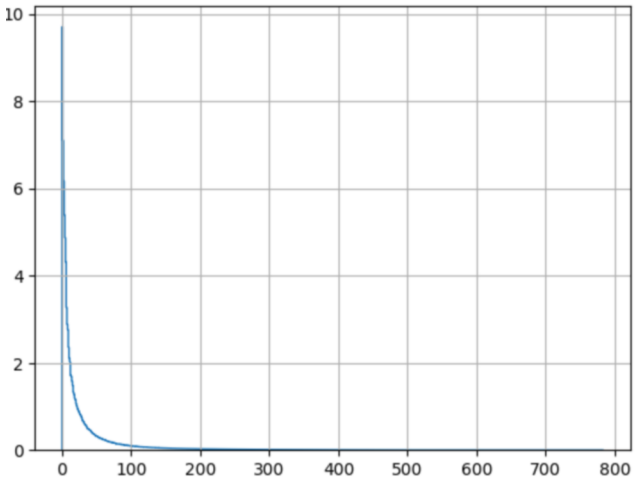


Fig. 3.23 MNIST eigenvalue percentage plot.

This results in Figures 3.22 and 3.23. Here we `sort` the array `eig` in decreasing order, then we `cumsum` the array to obtain the cumulative sums.

Because the rank of the MNIST dataset is 712 (§2.9), the bottom $72 = 784 - 712$ eigenvalues are exactly zero. A full listing shows that many more eigenvalues are near zero, and the second column in Figure 3.22 shows the top ten eigenvalues alone sum to almost 50% of the total variance.

```
from scipy.linalg import eigh

# use eigh for symmetric matrices
lamda, U = eigh(Q)

# sort in ascending order then reverse
sorted = sort(lamda)[::-1]
percent = sorted*100/totvar

# cumulative sums
sums = cumsum(percent)

data = array([percent,sums])
print(data.T[:20].round(decimals = 3))

d = len(lamda)
from matplotlib.pyplot import *

stairs(percent,range(d+1))

grid()
show()
```



A MNIST image is a point in $\mathbf{R}^{784}$. Now we turn to projecting the image from 784 dimensions down to n dimensions, where n is 784, 600, 350, 150, 50, 10, 1. Let Q be any $d \times d$ variance matrix, and let v be in $\mathbf{R}^d$. Let v_1, v_2, ..., v_d be the orthonormal basis of eigenvectors corresponding to the eigenvalues of Q, arranged in decreasing order, and let E be the diagonal matrix of eigenvalues. By EVD diagonalization (§3.2), if v_1, v_2, ..., v_d are the columns of U and the rows of V, then $Q = UEV$.

Here is code that returns the projection matrix P (§2.7) onto the span of the eigenvectors v_1, v_2, ..., v_n corresponding to the top n eigenvalues of Q.

```
from scipy.linalg import eigh

# projection matrix onto top n
# eigenvectors of variance
# of dataset
```

```
def pca(dataset,n):
   Q = cov(dataset.T)
   # columns of U are
   # eigenvectors of Q
   lamda, U = eigh(Q)
   # decreasing eigenvalue sort
   order = lamda.argsort()[::-1]
   # sorted top n columns of U
   # are cols of Uproj
   #  U is dxd Uproj is dxn
   Uproj = U[:,order[:n]]
   P = dot(Uproj,Uproj.T)
   return P
```

In the code, `lamda` is sorted in decreasing order, and the sorting order is saved as `order`. To obtain the top n eigenvectors of U, we sort the first n columns `U[:,order[:n]]` in the same order, resulting in the $d \times n$ matrix U_{proj}. The code then returns the projection matrix $P = U_{\text{proj}} U_{\text{proj}}^t$ (2.7.4).

Instead of working with the variance Q, as discussed at the start of the section, we can work directly with the dataset, using SVD, to obtain the eigenvectors.

```
from scipy.linalg import svd

# projection matrix onto top n
# eigenvectors of variance
# of dataset

def pca_with_svd(dataset,n):
   # center dataset
   mu = mean(dataset,axis = 0)
   vectors = dataset - mu
   # rows of V are
   # right singular vectors
   V = svd(vectors)[2]
   # no need to sort, already decreasing order
   Uproj = V[:n].T # top n rows as columns
   P = dot(Uproj,Uproj.T)
   return P
```

Let `v = dataset[1]` be the second image in the MNIST dataset, and let Q be the variance of the dataset. Then the code below returns the image compressed down to $n = 784, 600, 350, 150, 50, 10, 1$ dimensions, returning Figure 3.24.

```
def display_image(v,row,col,i):
   A = reshape(v,(28,28))
   fig.add_subplot(row, col,i)
   xticks([])
   yticks([])
   imshow(A,cmap = "gray_r")
```

```
fig = figure(figsize=(10,5))
row, col = 2, 4

v = dataset[1] # second image
display_image(v,row,col,1)

for i,n in enumerate([784,600,350,150,50,10,1],start=2):
   # either will work
   P = pca(dataset,n)
   #P = pca_with_svd(dataset[:100],n)
   projv = dot(P,v)
   display_image(projv,row,col,i)
```

If you run out of memory trying this code, cut down the dataset from 60,000 points to 10,000 points or fewer. The code works with `pca` or with `pca_with_svd`.



We show how to project a vector v in the dataset using `sklearn`. The following code sets up the PCA engine using `sklearn`.

```
from sklearn.decomposition import PCA

N = len(dataset)
n = 10
engine = PCA(n_components = n)
```

The following code computes the reduced dataset (§2.7)

```
reduced = engine.fit_transform(dataset)
reduced.shape
```

and returns $(N, n) = (60000, 10)$. The following code computes the projected dataset

```
projected = engine.inverse_transform(reduced)
projected.shape
```

and returns $(N, d) = (60000, 784)$.

Let U_{proj} be the $d \times n$ matrix with columns the top n eigenvectors. Then the projection matrix onto the column space of U_{proj} is $P = U_{\text{proj}} U_{\text{proj}}^t$. In the above code, `reduced` equals $U_{\text{proj}}^t v$ for each image v, and `projected` is $U_{\text{proj}} U_{\text{proj}}^t v$ for each image v.

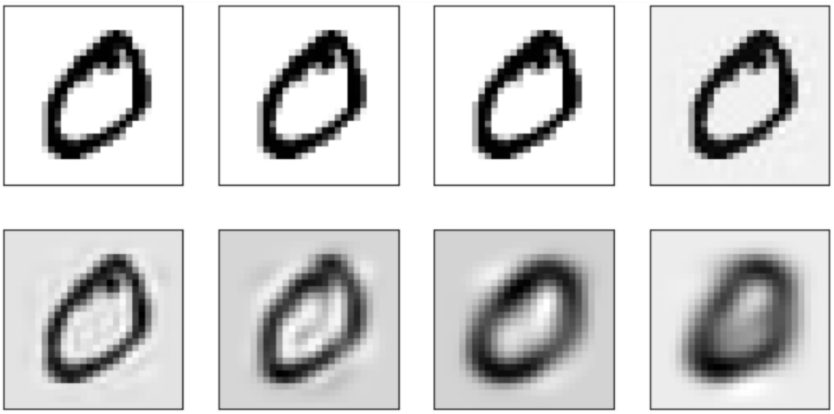


Fig. 3.24 Original and projections: $n = 784, 600, 350, 150, 50, 10, 3$.

Then the code

```
from matplotlib.pyplot import *

fig = figure(figsize=(10,5))
row, col = 2, 4

v = dataset[1] # second image
display_image(v,row,col,1)

for i,n in enumerate([784,600,350,150,50,10,3],start=2):
   engine = PCA(n_components = n)
   reduced = engine.fit_transform(dataset)
   projected = engine.inverse_transform(reduced)
   projv = projected[1] # second image
   display_image(projv,row,col,i)
```

returns Figure 3.24.



Now we project all vectors of the MNIST dataset onto two and three dimensions, those corresponding to the top two or three eigenvalues. To start, we compute `reduced` as above with $n = 3$, the top three components.

In the two-dimensional plotting code below, `reduced` is an array of shape `(60000,3)`, but we use only the top two components 0 and 1. When the rows are plotted as a scatterplot, we obtain Figure 3.25. Note the rows are plotted grouped by color, to match the legend, and each plot point's color is determined by the value of its label.

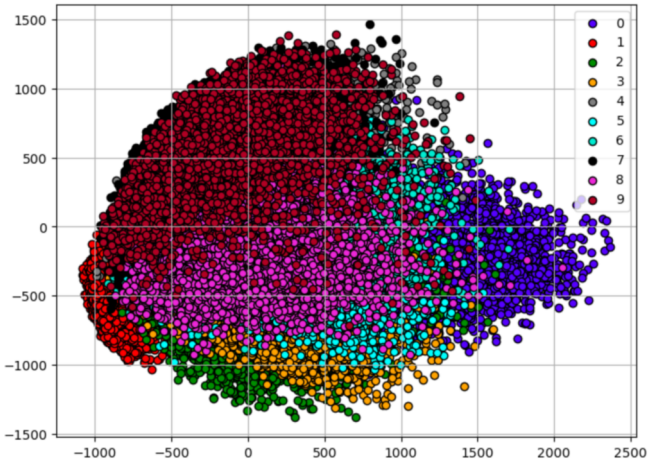


Fig. 3.25 The MNIST dataset (Two-dimensional projection).

```
from scipy.spatial import ConvexHull

Colors = ('blue', 'red', 'green', 'orange',
    ↪ 'gray','cyan','turquoise', 'black', 'orchid', 'brown')

for i,color in enumerate(Colors):
   points = reduced[labels==i,:]
   scatter(points[:,0], points[:,1],label = i, edgecolor =
    ↪ 'black')
   #hull = ConvexHull(points)
   #for simplex in hull.simplices:
   #plot(points[simplex, 0], points[simplex, 1], '-',c = color)

grid()
legend(loc = 'upper right')
show()
```

Code for displaying the convex hulls is included. Now we turn to three dimensional plotting. Here is the code

```
%matplotlib ipympl
from matplotlib.pyplot import *

ax = axes(projection="3d")

Colors = ('blue', 'green', 'black', 'brown', 'gray','cyan' ,
    ↪ 'turquoise', 'orange', 'orchid', 'red')

for i,color in enumerate(Colors):
   ax.scatter(reduced[labels==i,0], reduced[labels==i,1],
    ↪ reduced[labels==i,2], label = i, c = color, edgecolor =
    ↪ 'black')

ax.set_aspect("equal")
ax.set_axis_off()

legend(loc = 'upper right')
show()
```

The three dimensional plot of the complete MNIST dataset is Figure 1.5 in §1.2.



Code for the two-dimensional plot (Figure 3.26) of the Iris dataset is

```
from sklearn import datasets

iris = datasets.load_iris()
dataset = iris["data"]
labels = iris["target"]

engine = PCA(n_components = n)
reduced = engine.fit_transform(dataset)

# close previous plots
close("all")

Colors = ['blue', 'red', 'green']
Classes = ["Iris-setosa", "Iris-virginica", "Iris-versicolor"]
Labels = [0,1,2]

for Label, Class, Color in zip(Labels,Classes,Colors):
   scatter(reduced[labels==Label,0], reduced[labels==Label,1],
    ↪ label = Class, c = Color, edgecolor = 'black')

grid()
legend(loc = 'upper right')
show()
```

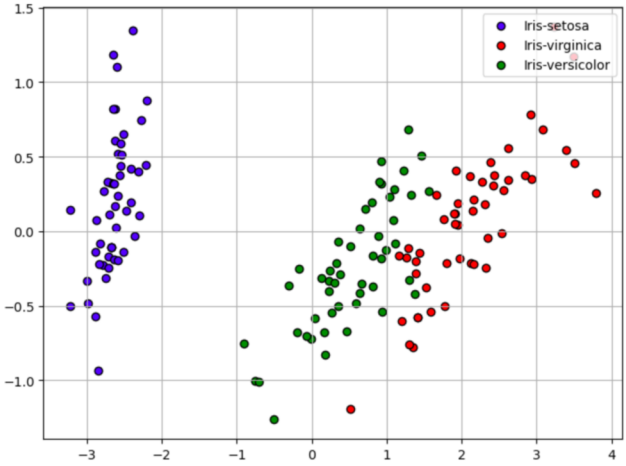


Fig. 3.26 The Iris dataset (Two-dimensional projection).

Exercises

Exercise 3.5.1 Use PCA to reduce the MNIST dataset to four dimensions. Out of the first 1,000 images, reduced to $\mathbf{R}^4$, what is the digit corresponding to the projected image closest to the origin?

3.6 Cluster Analysis

Cluster analysis seeks to partition a dataset into groups or clusters based on selected criteria, such as proximity in distance.

Let $x_1, x_2, \ldots, x_N$ be a dataset in $\mathbf{R}^d$. The simplest algorithm is *k-means clustering*. The algorithm is iterative: We start with k means $m_1, m_2, \ldots, m_k$, not necessarily part of the dataset, and we divide the dataset into k clusters, where the i-th cluster consists of the points x in the dataset for which the i-th mean m_i is nearest to x.

The algorithm is in two parts, the *assignment step* and the *update step.* Initially the means $m_1, m_2, \ldots, m_k$ are chosen at random, or by an educated guess, then clusters $C_1, C_2, \ldots, C_k$ are assigned, then each mean is recomputed as the mean of each cluster.

The `sklearn` package contains clustering routines, but here we write the code from scratch to illustrate the ideas. Here is an animated gif illustrating the convergence of the algorithm.

We assume each mean is a point in $\mathbf{R}^d$, coded as an array of shape `(d,)`, and each cluster is a collection of points in $\mathbf{R}^d$, coded as a list of arrays of shape `(d,)`.

Then the collection of cluster means is coded as an array of shape `(k,d)`, and the collection of clusters is coded as a list of lists of length k.

Given a point x, we select the first mean closest to x:

```
from numpy import *
set_printoptions(legacy = "1.25")

from scipy.linalg import norm

def nearest_index(x,means):
   distances = norm(means-x,axis=1)
   return argmin(distances)
```

Starting with empty clusters, we iterate the assign/update steps until the means no longer change. If any clusters remain empty, we discard them. Here is the assignment step.

```
def assign_clusters(dataset,means):
   clusters = [ [ ] for _ in range(k) ]
   for x in dataset:
      i = nearest_index(x,means)
      clusters[i].append(x)
   return [ c for c in clusters if len(c) ]
```

Here is the update step.

```
def update_means(clusters):
   return array([ sum(c,axis=0)/len(c) for c in clusters ])
```

Here is the iteration.

```
from numpy.random import default_rng
samples = default_rng().random

def kmeans(dataset,k):
   close_enough = False
   (N,d) = dataset.shape
   means = samples((k,d))
   while not close_enough:
      clusters = assign_clusters(dataset,means)
      print([len(c) for c in clusters])
      new_means = update_means(clusters)
      # only check closeness if number of means unchanged
      if len(new_means) == len(means):
```

```
        close_enough = allclose(means,new_means)
      means = new_means
   return means, clusters

d, k, N = 2, 7, 100
dataset = samples((N,d))

means, clusters = kmeans(dataset, k)
```

This code also prints the cluster sizes after each iteration. Here is code that plots a cluster.

```
def plot_cluster(mean,cluster,color,marker):
   scatter(*array(cluster).T, s = 30, c = color, marker = marker)
   scatter(*mean, s=20, c=color, marker='*')
```

Here is code for the entire iteration. `hexcolor` is in §1.3.

```
from matplotlib.pyplot import *

from numpy.random import default_rng
samples = default_rng().random

d = 2
k,N = 7,100

from random import choice

def hexcolor():
   chars = '0123456789abcdef'
   return "#" + ''.join([choice(chars) for _ in range(6)])

dataset = samples((N,d))
means = samples((k,d))
colors = [ hexcolor() for _ in range(k) ]

scatter(*dataset.T,s = 10)
grid()
show()

means, clusters = kmeans(dataset,k)

for i,cluster in enumerate(clusters):
   tex = '$' + str(i) + '$'
   mean = means[i]
   color = colors[i]
   plot_cluster(mean,cluster,color,tex)

grid()
show()
```

Chapter 4
Calculus

The material in this chapter lays the groundwork for Chapter 7. It assumes the reader has some prior exposure, and the first section quickly reviews basic material essential for our purposes.

The overarching role played by convex functions is emphasized repeatedly. To avoid technical difficulties, our convex functions are usually strongly convex, in the sense that their second derivatives are bounded above and below.

The chain rule is treated extensively, combinatorially (back propagation) and geometrically (time-derivatives). Both interpretations are crucial for neural network training in Chapter 7.

Because it is used infrequently in the text, integration is treated separately in an appendix (§A.6).

Even though parts of §4.5 are heavy-going, the material is necessary for Chapter 7. Nevertheless, for a first pass, the reader should feel free to skim this material and come back to it after the need is made clear.

4.1 Single-Variable Calculus

In this section, we focus on single-variable calculus, and in §4.3, we review multi-variable calculus. Recall the *slope* of a line $y = mx + b$ equals m.

Let $y = f(x)$ be a function as in Figure 4.1, and let a be a fixed point. The *derivative of* $f(x)$ *at the point* a is the slope of the line tangent to the graph of $f(x)$ at a. Then the derivative at a point a is a number $f'(a)$ possibly depending on a.

Definition of Derivative

The derivative of $f(x)$ at the point a is the slope of the line tangent to the graph of $f(x)$ at a.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

O. Hijab, *Math for Data Science*, https://doi.org/10.1007/978-3-031-89707-8_4

Since a constant function $f(x) = c$ is a line with slope zero, *the derivative of a constant is zero*. Since $f(x) = mx+b$ is a line with slope m, its derivative is m.

Since the tangent line at a passes through the point $(a, f(a))$, and its slope is $f'(a)$, the equation of the tangent line at a is

$$y = f(a) + f'(a)(x - a).$$

Based on the definition, natural properties of the derivative are

A. The derivative of $f(x) + g(x)$ is $f'(x) + g'(x)$, and $(-f(x))'$ is $-f'(x)$.
B. If $f'(x) \geq 0$ on an interval $[a, b]$, then $f(b) \geq f(a)$.
C. If $f'(x) \leq 0$ on an interval $[a, b]$, then $f(b) \leq f(a)$.

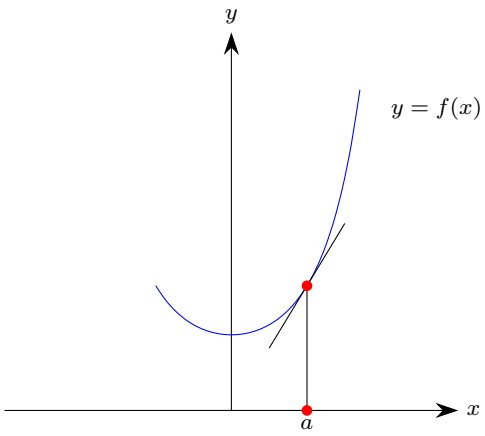


Fig. 4.1 $f'(a)$ is the slope of the tangent line at a.

Using these properties, we determine the formula for $f'(a)$. Suppose the derivative is bounded between two extremes m and L at every point x in an interval $[a, b]$, say

$$m \leq f'(x) \leq L, \qquad a \leq x \leq b.$$

Then by **A**, the derivative of $h(x) = f(x) - mx$ at x equals $h'(x) = f'(x) - m$. By assumption, $h'(x) \geq 0$ on $[a, b]$, so, by **B**, $h(b) \geq h(a)$. Since $h(a) = f(a) - ma$ and $h(b) = f(b) - mb$, this leads to

$$\frac{f(b) - f(a)}{b - a} \geq m.$$

Repeating this same argument with $f(x) - Lx$, and using **C**, leads to

$$\frac{f(b) - f(a)}{b - a} \leq L.$$

We have shown

First Derivative Bounds

If $m \le f'(x) \le L$ for $a \le x \le b$, then

$$m \le \frac{f(b) - f(a)}{b - a} \le L. \tag{4.1.1}$$

As an immediate consequence, when $m = L = 0$, applying this to any subinterval $[a', b']$ in $[a, b]$,

Zero Derivative Implies Constant

For any $f(x)$,

$$f'(x) = 0 \qquad \Longrightarrow \qquad f(x) \text{ is constant.} \tag{4.1.2}$$

When b is close to a, we expect both extremes m and L to be close to $f'(a)$. From (4.1.1), we arrive at the formula for the derivative,

Derivative Formula

$$f'(a) = \lim_{x \to a} \frac{f(x) - f(a)}{x - a}. \tag{4.1.3}$$

From (4.1.3), the derivative of a line $f(x) = mx + b$ equals $f'(a) = m$, agreeing with what we already know. We usually deal with limits as in (4.1.3) in an intuitive manner. When needed, please refer to §A.7 for basic properties of limits.

Below we also write

$$y' = f'(x) = \frac{dy}{dx}$$

or

$$f'(a) = \left.\frac{dy}{dx}\right|_{x=a}$$

When the particular point a is understood from the context, we write y'.



From (4.1.3), the basic properties of the derivative are

- *Sum rule.* $h = f + g$ implies $h' = f' + g'$,
- *Product rule.* $h = fg$ implies $h' = f'g + fg'$,
- *Quotient rule.* $h = f/g$ implies $h' = (f'g - fg')/g^2$.

- *Chain rule.* $u = f(x)$ and $y = g(u)$ implies

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}.$$

To visualize the chain rule, suppose

$$u = f(x) = \sin x,$$
$$y = g(u) = u^2.$$

These are two functions f, g in composition, as in Figure 4.2.

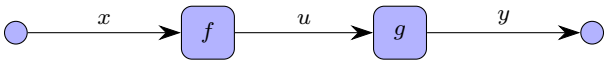


Fig. 4.2 Composition of two functions.

Suppose $x = \pi/4$. Then $u = \sin(\pi/4) = 1/\sqrt{2}$, and $y = u^2 = 1/2$. Since

$$\frac{dy}{du} = 2u = \frac{2}{\sqrt{2}}, \qquad \frac{du}{dx} = \cos x = \frac{1}{\sqrt{2}},$$

by the chain rule,

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} = \frac{2}{\sqrt{2}} \cdot \frac{1}{\sqrt{2}} = 1.$$

Since the chain rule is important for machine learning, it is discussed in detail in §4.4.

By the product rule,

$$(x^2)' = x'x + xx' = 1x + x1 = 2x.$$

Similarly one obtains the *power rule*

$$(x^n)' = nx^{n-1}. \tag{4.1.4}$$

Using the chain rule, the power rule can be derived for any rational number n, positive or negative. For example, since $(\sqrt{x})^2 = x$, we can write $x = f(g(x))$ with $f(x) = x^2$ and $g(x) = \sqrt{x}$. By the chain rule,

$$1 = (x)' = f'(g(x))g'(x) = 2g(x)g'(x) = 2\sqrt{x}(\sqrt{x})'.$$

Solving for $(\sqrt{x})'$ yields

$$(\sqrt{x})' = \frac{1}{2\sqrt{x}},$$

which is (4.1.4) with $n = 1/2$. In this generality, the variable x is restricted to positive values only.

For example, the code

```
from sympy import *

x, a = symbols('x, a')
f = x**a

display(f.diff(x), diff(f,x), f.diff(x).simplify(),
    ↪ simplify(diff(f,x)))
```

returns

$$\frac{ax^a}{x}, \qquad \frac{ax^a}{x}, \qquad ax^{a-1}, \qquad ax^{a-1}.$$



The power rule can be combined with the chain rule. For example, if

$$u = 1 - p + cp, \qquad f(p) = u^n, \qquad g(u) = \frac{u^{n+1}}{(c-1)(n+1)},$$

then

$$F(p) = \frac{(1-p+cp)^{n+1}}{(c-1)(n+1)},$$

and

$$F'(p) = g'(u)u' = u^n,$$

hence

$$F(p) = \frac{(1-p+cp)^{n+1}}{(c-1)(n+1)} \qquad \Longrightarrow \qquad F'(p) = f(p). \tag{4.1.5}$$



The *second derivative* $f''(x)$ of $f(x)$ is the derivative of the derivative,

$$f''(x) = \left(f'(x)\right)'.$$

For example,

$$(x^n)'' = (nx^{n-1})' = n(n-1)x^{n-2} = \frac{n!}{(n-2)!}x^{n-2} = P(n,2)x^{n-2}$$

(for $n!$ and $P(n,k)$ see §A.1).

More generally, the *k-th derivative* $f^{(k)}(x)$ is the derivatives taken k times, so

$$(x^n)^{(k)} = n(n-1)(n-2)\dots(n-k+1)x^{n-k} = \frac{n!}{(n-k)!}x^{n-k} = P(n,k)x^{n-k}.$$

When $k=0$, $f^{(0)}(x) = f(x)$, and, when $k=1$, $f^{(1)}(x) = f'(x)$. The code

```
from sympy import *

x, n = symbols('x, n')

diff(x**n,x,3).simplify()
```

returns the third derivative $nx^{n-3}(n^2-3n+2)$.



Here is an example using derivatives from `sympy`. Given a power n, let $p_n(x) = (x^2-1)^n$. Then $p_n(x)$ is a polynomial of degree $2n$.

The *Legendre polynomial* $P_n(x)$ is the n-th derivative of $p_n(x)$ divided by $n!2^n$. Then $P_n(x)$ is a polynomial of degree n.

For example, when $n=1$, $p_1(x) = x^2-1$, so

$$P_1(x) = \frac{1}{1!2^1}(x^2-1)' = \frac{1}{2}\,2x = x.$$

When $n=2$,

$$P_2(x) = \frac{1}{2!2^2}\left((x^2-1)^2\right)'' = \frac{1}{2}(3x^2-1).$$

The Python code for $P_n(x)$ uses symbolic functions and symbolic derivatives.

```
from sympy import diff, symbols
from scipy.special import factorial

def sym_legendre(n):
   # symbolic variable
   x = symbols('x')
   # symbolic function
   p = (x**2 - 1)**n
   nfact = factorial(n,exact = True)
   # symbolic nth derivative
   return p.diff(x,n)/(nfact * 2**n)
```

For example,

```
from sympy import simplify

for n in range(6):
```

```
display(simplify(sym_legendre(n)))
```

returns the first six Legendre Polynomials, starting from $n = 0$:

$$\left[1, x, \frac{3x^2}{2} - \frac{1}{2}, \frac{x\left(5x^2-3\right)}{2}, \frac{35x^4}{8} - \frac{15x^2}{4} + \frac{3}{8}, \frac{x\left(63x^4-70x^2+15\right)}{8}\right]$$

To compute values such as $P_4(5)$, we have modify `sym_legendre` to a `numpy` function as follows,

```
from sympy import lambdify

def num_legendre(n):
   x = symbols('x')
   f = sym_legendre(n)
   return lambdify(x,f, 'numpy')
```

The function `num_legendre(n)` can be evaluated, plotted, integrated, etc.

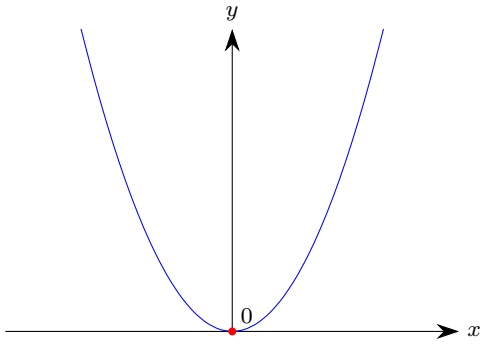


Fig. 4.3 Increasing or decreasing?

For the parabola in Figure 4.3, $y = x^2$ so, by the power rule, $y' = 2x$. Since $y' > 0$ when $x > 0$ and $y' < 0$ when $x < 0$, this agrees with the increase/decrease of the graph. In particular, the minimum of the parabola occurs when $y' = 0$.

For the curve $y = x^4 - 2x^2$ in Figure 4.4,

$$y' = 4x^3 - 4x = 4x(x^2 - 1) = 4x(x-1)(x+1),$$

so y' is a product of the three factors $4x$, $x-1$, $x+1$. Since the zeros of these factors are 0, 1, and -1, and $y' > 0$ when all factors are positive, or two of them are negative, this agrees with the increase/decrease in the figure.

Here $y' = 0$ occurs at the two minima $x = \pm 1$ and at the local maximum 0. Notice 0 is not a global maximum as there is no greatest value for y.

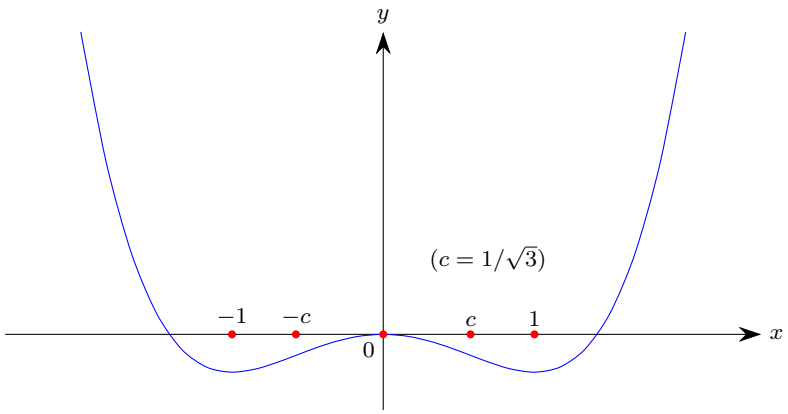


Fig. 4.4 Increasing or decreasing?

A point x^* is a *local maximizer* of $f(x)$ if $f(x^*) \geq f(x)$ for x near x^*. A point x^* is a *global maximizer* if $f(x^*) \geq f(x)$ for all x.

A point x^* is a *local minimizer* of $f(x)$ if $f(x^*) \leq f(x)$ for x near x^*. A point x^* is a *global minimizer* if $f(x^*) \leq f(x)$ for all x.

A *critical point* is a point x^* where the derivative equals zero, $f'(x^*) = 0$. Figures 4.3 and 4.4 exhibit examples of local or global maximizers or minimizers, and show they are critical points.

Let $f(x)$ be any function, and let x^* be a local maximizer of $f(x)$. Then for $x > x^*$ near x^*, we have

$$\frac{f(x) - f(x^*)}{x - x^*} \leq 0,$$

so, by (4.1.1), $f'(x^*) \leq 0$. Similarly, for $x < x^*$ near x^*,

$$\frac{f(x) - f(x^*)}{x - x^*} \geq 0,$$

so, by (4.1.1), $f'(x^*) \leq 0$. Hence at a local maximizer x^*, we must have $f'(x^*) = 0$.

By the same method, a local minimizer x^* satisfies $f'(x^*) = 0$. This shows

A Maximizer or Minimizer is a Critical Point

If x^* is a maximizer or minimizer of $f(x)$ on an interval $[a, b]$, and $a < x^* < b$, then $f'(x^*) = 0$. Thus the maximum of $f(x)$ over the interval equals the maximum over critical points and endpoints,

$$\max_{a \le x \le b} f(x) = \max(f(a), f(x^*), f(b)), \tag{4.1.6}$$

and the minimum of $f(x)$ over the interval equals the minimum over critical points and endpoints,

$$\min_{a \le x \le b} f(x) = \min(f(a), f(x^*), f(b)), \tag{4.1.7}$$

In other words, to find the maximum or minimum of $f(x)$, find the critical points x^*, plug them and the endpoints a, b into $f(x)$, and select whichever yields the maximum value or minimum value.



Suppose $f(x)$ is given by a finite or infinite sum

$$f(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \dots \tag{4.1.8}$$

Then $f(0) = c_0$. Taking derivatives, by the sum, product, and power rules,

$$\begin{aligned} f'(x) &= c_1 + 2c_2 x + 3c_3 x^2 + 4c_4 x^3 + \dots \\ f''(x) &= 2c_2 + 3 \cdot 2c_3 x + 4 \cdot 3c_4 x^2 + \dots \\ f'''(x) &= 3 \cdot 2c_3 + 4 \cdot 3 \cdot 2c_4 x + \dots \\ f^{(4)}(x) &= 4 \cdot 3 \cdot 2c_4 + \dots \end{aligned} \tag{4.1.9}$$

Inserting $x = 0$, we obtain $f'(0) = c_1$, $f''(0) = 2c_2$, $f'''(0) = 3 \cdot 2c_3$, $f^{(4)}(0) = 4 \cdot 3 \cdot 2c_4$. This can be encapsulated by $f^{(n)}(0) = n! c_n$, $n = 0, 1, 2, 3, 4, \dots$, which is best written

$$\frac{f^{(n)}(0)}{n!} = c_n, \qquad n \ge 0.$$

Going back to (4.1.8), we derived

Taylor Series

If $f(x)$ has derivatives of all orders,

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!} x^n$$
$$= f(0) + f'(0)x + f''(0)\frac{x^2}{2} + f'''(0)\frac{x^3}{6} + f^{(4)}(0)\frac{x^4}{24} + \dots \tag{4.1.10}$$



We now compute the derivatives of the exponential function (§A.3). By the compound-interest formula (A.3.9),

$$e^x = \lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n.$$

By the power rule and chain rule,

$$\left(\left(1 + \frac{x}{n}\right)^n\right)' = n\left(1 + \frac{x}{n}\right)^{n-1} \cdot \frac{1}{n} = \left(1 + \frac{x}{n}\right)^{n-1}.$$

From this follows

$$(e^x)' = \lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^{n-1} = \lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n \cdot \frac{1}{1 + x/n} = e^x \cdot 1 = e^x.$$

Since the first derivative is e^x, so is the second derivative. This derives

Derivative of the Exponential Function

The exponential function satisfies

$$(e^x)' = e^x.$$

Since

$$(e^x)^{(n)} = e^x, \qquad n \geq 0,$$

the Taylor series of e^x reduces to the exponential series (A.3.13).



The *logarithm* function is the inverse of the exponential function,

$$y = \log x \qquad \Longleftrightarrow \qquad x = e^y.$$

This is the same as saying

$$\log(e^y) = y, \qquad e^{\log x} = x.$$

From here, we see the logarithm is defined only for $x > 0$ and is strictly increasing (Figure 4.5).

Since $e^0 = 1$,

$$\log 1 = 0.$$

Since $e^{\infty} = \infty$ (Figure A.2),

$$\log \infty = \infty.$$

Since $e^{-\infty} = 1/e^{\infty} = 1/\infty = 0$,

$$\log 0 = -\infty.$$

We also see $\log x$ is negative when $0 < x < 1$, and positive when $x > 1$.

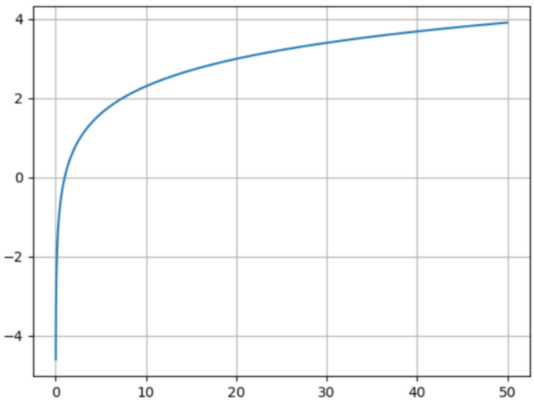


Fig. 4.5 The logarithm function $\log x$.

Moreover, by the law of exponents,

$$\log(ab) = \log a + \log b.$$

For $a > 0$ and b real, define

$$a^b = e^{b \log a}.$$

Then, by definition,

$$\log(a^b) = b \log a,$$

and

$$\left(a^b\right)^c = \left(e^{b \log a}\right)^c = e^{bc \log a} = a^{bc}.$$

By definition of the logarithm, $y = \log x$ is shorthand for $x = e^y$. Use the chain rule to find y':

$$x = e^y \qquad \Longrightarrow \qquad 1 = x' = (e^y)' = e^y y' = xy',$$

so

$$y = \log x \qquad \Longrightarrow \qquad y' = \frac{1}{x}.$$

Derivative of the Logarithm

$$y = \log x \qquad \Longrightarrow \qquad y' = \frac{1}{x}. \tag{4.1.11}$$

Since the derivative of $\log(1+x)$ is $1/(1+x)$, the chain rule implies

$$\frac{d^n}{dx^n} \log(1+x) = (-1)^{n-1} \frac{(n-1)!}{(1+x)^n}, \qquad n \geq 1.$$

From this, the Taylor series of $\log(1+x)$ is

$$\log(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots. \tag{4.1.12}$$



Now we look at the increase/decrease in y', rather than in y. Applying the above logic to y' instead to y, we see y' is increasing when $y'' \geq 0$, and y' is decreasing when $y'' \leq 0$.

In the first case, we say $f(x)$ is *convex*, while in the second case, we say $f(x)$ is *concave*.

Since multiplying by a minus interchanges $y'' \geq 0$ and $y'' \leq 0$, concavity of $f(x)$ is equivalent to convexity of $-f(x)$.

The simplest example of a convex function is a quadratic function,

$$f(x) = ax^2 + 2bx + c, \qquad a > 0.$$

The graph of this function is a shifted and scaled version of the parabola in Figure 4.3.

By completing the square,

$$ax^2 + 2bx + c = a\left(x + \frac{b}{a}\right)^2 + c - \frac{b^2}{a}.$$

Thus the minimizer is at the vertex $x = -b/a$, and the minimum is $c - b^2/a$.

For this quadratic function, the slope at x equals $y' = 2(ax+b)$. Thus as x increases, y' increases. Even though the parabola height y decreases when x is

less than the vertex $-b/a$, and increases when x is greater than the vertex, its slope y' is always increasing: When x is less than the vertex, as x increases, y' is less and less negative, while, when x is greater than the vertex, as x increases, y' is more and more positive.

Since y' increases when its derivative is positive, the parabola's behavior is encapsulated in

$$y'' = (y')' = 2(ax + b)' = 2a > 0.$$

In general,

Second Derivatives and Convexity

$y = f(x)$ is convex if $y'' \geq 0$, and concave if $y'' \leq 0$.

A point where $y'' = 0$ is an *inflection point*. In Figure 4.4, the graph is convex away from 0, and concave near 0. Since

$$y'' = (x^4 - 2x^2)'' = (4x^3 - 4x)' = 12x^2 - 4 = 4(3x^2 - 1),$$

the inflection points are $x = \pm 1/\sqrt{3}$. Hence $f(x)$ is convex when $|x| > 1/\sqrt{3}$, and $f(x)$ is concave when $|x| < 1/\sqrt{3}$. Since $1/\sqrt{3} < 1$, $f(x)$ is convex near $x = \pm 1$.



A function $y = f(x)$ is *strictly convex* if $y'' > 0$, and *strictly concave* if $y'' < 0$. Then $y = f(x)$ is strictly convex when y' is strictly increasing, and $y = f(x)$ is strictly concave when y' is strictly decreasing.

Second Derivatives and Strict Convexity

$y = f(x)$ is strictly convex if $y'' > 0$, and strictly concave if $y'' < 0$.

Since $(x^2)'' = 2 > 0$ and $(e^x)'' = e^x > 0$, x^2 and e^x are strictly convex everywhere, and $x^4 - 2x^2$ is strictly convex for $|x| > 1/\sqrt{3}$.

Let $y = -\log x$.By the power rule,

$$y'' = (-\log x)'' = \left(-\frac{1}{x}\right)' = -\left(x^{-1}\right)' = \frac{1}{x^2}.$$

Since $y'' > 0$, which shows $-\log x$ is strictly convex. This shows $\log x$ is strictly concave on $x > 0$.



Suppose $y = f(x)$ is convex, so y' is increasing. Then $a \le t \le x \le b$ implies $f'(a) \le f'(t) \le f'(x) \le f'(b)$. Taking $m = f'(a)$ and $L = f'(x)$ in (4.1.1),

$$f'(a) \le \frac{f(x) - f(a)}{x - a} \le f'(x), \qquad a \le x \le b.$$

Since the tangent line at a is $y = f'(a)(x - a) + f(a)$, rearranging this last inequality, we obtain

Convex Function Graph Lies Above the Tangent Line

If $f(x)$ is convex on $[a.b]$, then

$$f(a) + f'(a)(x - a) \le f(x), \qquad a \le x \le b.$$

For example, the function in Figure 4.6 is convex near $x = a$, and the graph lies above its tangent line at a.



Let $y = f(x)$ be a strictly convex function on an interval $[a, b]$, so $y'' > 0$ on $[a, b]$. By convexity, the derivative is increasing, so $f'(a) \le f'(b)$. This means either $f'(a) = f'(b)$ or $f'(a) < f'(b)$.

If $f'(a) = f'(b)$, then since the derivative is increasing, we have $f'(a) = f'(x) = f'(b)$ for every x in $[a, b]$, so $f'(x)$ is constant on $[a, b]$. But this implies the derivative of $f'(x)$ equals zero, or $y'' = f''(x) = 0$. Since this contradicts strict convexity, we conclude

First Derivative of Strictly Convex is Strictly Increasing

If $y = f(x)$ is strictly convex on $[a, b]$, then $f'(b) > f'(a)$.

As a consequence, we have

Strictly Convex Has At Most One Minimizer

If $y = f(x)$ is strictly convex, there is at most one minimizer.

A critical point is where $y' = 0$. By the previous result, there is at most one critical point. Since every minimizer is a critical point, there is at most one minimizer.



Let $p_m(x)$ be the parabola

$$p_m(x) = f(a) + f'(a)(x-a) + \frac{m}{2}(x-a)^2. \tag{4.1.13}$$

Then $p_m''(x) = m$. Moreover the graph of $p_m(x)$ is tangent to the graph of $f(x)$ at $x = a$, in the sense $f(a) = p_m(a)$ and $f'(a) = p_m'(a)$. Because of this, we call $p_m(x)$ the *lower tangent parabola*.

Similarly, let $p_L(x)$ be the parabola

$$p_L(x) = f(a) + f'(a)(x-a) + \frac{L}{2}(x-a)^2. \tag{4.1.14}$$

Then $p_L''(x) = L$. Moreover the graph of $p_L(x)$ is tangent to the graph of $f(x)$ at $x = a$, in the sense $f(a) = p_L(a)$ and $f'(a) = p_L'(a)$. Because of this, we call $p_L(x)$ the *upper tangent parabola*.

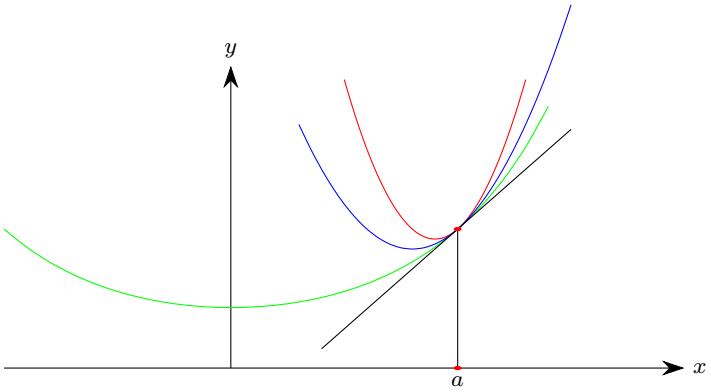


Fig. 4.6 Tangent parabolas $p_m(x)$ (green), $p_L(x)$ (red), $L > m > 0$.

We say $f(x)$ is *strongly convex* on an interval $[a, b]$ if there are positive constants m and L satisfying

$$m \le f''(x) \le L, \qquad a \le x \le b.$$

When the inequalities hold for all x, we say $f(x)$ is *strongly convex*.

There are several levels of convexity. They are, in order of generality,

- quadratic functions, $q = f''(x)$ is constant with $q > 0$,
- strongly convex functions, $L \ge f''(x) \ge m$,
- strictly convex functions, $f''(x) > 0$,
- general convex functions, $f''(x) \ge 0$.

The exponential function e^x is strongly convex on any bounded interval $[a, b]$, since $(e^x)'' = e^x$ and $e^a \le e^x \le e^b$ when $a \le x \le b$. This implies e^x is strictly convex for all x. However, e^x is not strongly convex for all x, since $e^{-\infty} = 0$.



When $f(x)$ is convex, we saw above the graph of $f(x)$ lies above its tangent line. When $f(x)$ is strongly convex, we can specify the size of the difference between the graph and the tangent line. In fact, the graph is constrained to lie above or below the lower or upper tangent parabolas for all x.

Lower and Upper Tangent Parabolas

Let $f(x)$ be strongly convex, with $m \le f''(x) \le L$. Then the graph lies between the lower and upper tangent parabolas $p_m(x)$ and $p_L(x)$,

$$\frac{m}{2}(x-a)^2 \le f(x) - f(a) - f'(a)(x-a) \le \frac{L}{2}(x-a)^2. \qquad (4.1.15)$$

To see this, suppose $f''(x) \ge m$. then $g(x) = f(x) - p_m(x)$ satisfies

$$g''(x) = f''(x) - p_m''(x) = f''(x) - m \ge 0,$$

so $g(x)$ is convex, so $g(x)$ lies above its tangent line at $x = a$. Since $g(a) = 0$ and $g'(a) = 0$, the tangent line is 0, and we conclude $g(x) \ge 0$, which is the left half of (4.1.15). Similarly, if $f''(x) \le L$, then $p_L(x) - f(x)$ is convex, leading to the right half of (4.1.15).

Choosing $a = 0$ in (4.1.15) implies the graph lies between two parabolas. Because of this, a strongly convex function must have at least one minimizer x^*. Because a minimizer is a critical point, and strong convexity implies strict convexity which in turn implies y' is strictly increasing, there is at most one minimizer x^*. This shows

Strongly Convex Implies Unique Global Minimizer

A strongly convex function has exactly one global minimizer x^*.

In particular, choosing $a = x^*$ in (4.1.15) implies

$$\frac{m}{2}(x-x^*)^2 \le f(x) - f(x^*) \le \frac{L}{2}(x-x^*)^2. \qquad (4.1.16)$$



When $m \le f''(x) \le L$, by (4.1.1),

$$t = \frac{f'(b) - f'(a)}{b - a} \implies m \le t \le L. \tag{4.1.17}$$

Replacing b by x implies

Lipschitz Bounds

Let $f(x)$ be strongly convex. If $p = f'(x)$ and $q = f'(a)$, then

$$m|x - a| \le |p - q| \le L|x - a|. \tag{4.1.18}$$

Replacing b by x in (4.1.17), then multiplying by $(x - a)^2$, we obtain

Strongly Convex Gradient Bounds

Let $f(x)$ be strongly convex. If $q = f'(x)$ and $p = f'(a)$, then

$$m(x - a)^2 \le (q - p)(x - a) \le L(x - a)^2. \tag{4.1.19}$$

Moreover (4.1.17) implies

$$t^2 - (m + L)t + mL = (t - m)(t - L) \le 0.$$

Replacing b by x, then multiplying by $(x - a)^2$, yields

Coercivity of the Gradient

Let $f(x)$ be strongly convex. If $q = f'(x)$ and $p = f'(a)$, then

$$\frac{mL}{m + L}(x - a)^2 + \frac{1}{m + L}(q - p)^2 \le (q - p)(x - a). \tag{4.1.20}$$

These are used in gradient descent §7.8.



For gradient descent, we need the relation between a convex function and its convex dual. The *convex dual* $g(p)$ of a function $f(x)$ is

$$g(p) = \max_x (px - f(x)). \tag{4.1.21}$$

Here the maximum is over all x.

When $f(x)$ is strongly convex, $f(x) - px$ is also strongly convex, so $f(x) - px$ has a unique minimizer. It follows that for each p there is a unique maximizer x^* in (4.1.21),

$$g(p) = px - f(x) \qquad \Longleftrightarrow \qquad x = x^*(p).$$

The simplest example is a quadratic

$$f(x) = \frac{1}{2}ax^2 + bx + c, \qquad a > 0.$$

In this case, solving $(px - f(x))' = 0$ leads to

$$0 = \left(px - \left(\frac{1}{2}ax^2 + bx + c\right)\right)' = p - ax - b,$$

or $x^* = (p - b)/a$. Plugging x^* back into (4.1.21) yields

$$g(p) = \frac{1}{2a}(p - b)^2 - c.$$



Let $f(x)$ be a strongly convex function. Going back to the general setting (4.1.21), since a maximizer x^* is a critical point,

$$0 = (px - f(x))' = p - f'(x) \qquad \Longleftrightarrow \qquad x = x^*,$$

hence

$$p = f'(x^*).$$

By the product rule and chain rule, since $x^* = x^*(p)$ depends on p,

$$g'(p) = (px^* - f(x^*))' = x^* + p(x^*)' - f'(x^*)(x^*)' = x^* + (p - f'(x^*))(x^*)' = x^*.$$

We conclude

$$p = f'(x) \qquad \Longleftrightarrow \qquad x = g'(p). \tag{4.1.22}$$

Thus $f'(x)$ is the inverse function of $g'(p)$.

Since $f'(x)$ is the inverse function of $g'(p)$, we have

$$f'(g'(p)) = p.$$

Differentiating with respect to p again yields

$$f''(g'(p))g''(p) = 1.$$

Since $g(p) = px - f(x)$ is equivalent to $f(x) = px - g(p)$, we conclude

Dual of the Dual

Let $f(x)$ be strongly convex. If $g(p)$ is the convex dual of $f(x)$, then $f(x)$ is the convex dual of $g(p)$, $g(p)$ is strongly convex, and

$$g''(p) = \frac{1}{f''(x)} \qquad \Longleftrightarrow \qquad p = f'(x) \qquad \Longleftrightarrow \qquad x = g'(p).$$

It follows that the dual $g(p)$ is strongly convex with

$$\frac{1}{L} \leq g''(p) \leq \frac{1}{m}.$$

Using this, we also have (Exercise **4.1.20**)

Dual Second Derivative Bounds

Let $f(x)$ be strongly convex. If $p = f'(x)$ and $q = f'(a)$, then

$$\frac{1}{2L}(p-q)^2 \leq f(x) - f(a) - f'(a)(x-a) \leq \frac{1}{2m}(p-q)^2. \quad (4.1.23)$$



For the chi-squared distribution §5.5, we need Newton's generalization of the binomial theorem (A.2.7) to general exponents.

Newton's Binomial Theorem

Let n be any real number. For $a > 0$ and $-a < x < a$,

$$(a+x)^n = a^n + na^{n-1}x + \binom{n}{2}a^{n-2}x^2 + \binom{n}{3}a^{n-3}x^3 + \dots.$$

This makes sense because the binomial coefficient $\binom{n}{k}$ is defined for any real number n (A.2.11), (A.2.12).

Using summation notation,

$$(a+x)^n = \sum_{k=0}^{\infty} \binom{n}{k} a^{n-k}x^k. \quad (4.1.24)$$

The only difference between the binomial theorem and (4.1.24) is the upper limit of the summation, which is set to infinity. When n is a whole number, by (A.1.1), we have

$$\binom{n}{k} = 0, \qquad \text{for } k > n,$$

so (4.1.24) is a sum of $n+1$ terms, yielding the binomial theorem exactly. When n is not a whole number, the sum (4.1.24) is an infinite sum.

Actually, in §5.5, we need the special case $a = 1$, which we write in slightly different notation,

$$(1+x)^p = \sum_{n=0}^{\infty} \binom{p}{n} x^n. \tag{4.1.25}$$

Newton's binomial theorem (4.1.24) is a special case of the Taylor series (4.1.10). To see this, set

$$f(x) = (a+x)^n.$$

Then, by the power rule,

$$f^{(k)}(x) = n(n-1)(n-2)\dots(n-k+1)(a+x)^{n-k},$$

so

$$\frac{f^{(k)}(0)}{k!} = \frac{n(n-1)(n-2)\dots(n-k+1)}{k!} a^{n-k} = \binom{n}{k} a^{n-k}.$$

Writing out the Taylor series,

$$(a+x)^n = \sum_{k=0}^{\infty} \frac{f^{(k)}(0)}{k!} = \sum_{k=0}^{\infty} \binom{n}{k} a^{n-k} x^k,$$

which is Newton's binomial theorem.

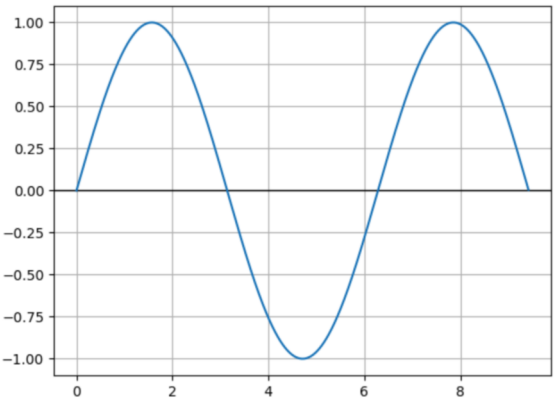


Fig. 4.7 The sine function.

The trigonometric functions sine and cosine were defined in (A.4.3). To plot them, use

```
from matplotlib.pyplot import *
from numpy import *

a, b = 0, 3*pi
theta = arange(a,b,.01)

axhline(0, color = 'black', lw = 1)
# set_pi_ticks(a,b)

plot(theta,sin(theta))

grid()
show()
```

This returns Figure 4.7.

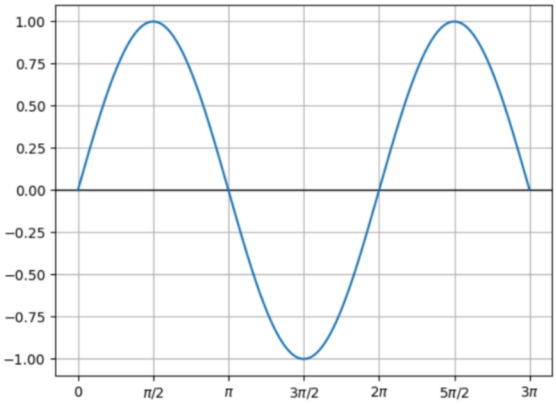


Fig. 4.8 The sine function with $\pi/2$ tick marks.

It is often convenient to set the horizontal axis tick marks at the multiples of $\pi/2$. For this, we use

```
from numpy import *
from matplotlib.pyplot import *

# r'...' is a raw string. In a raw string
# the backslash \ is not an escape character

def label(k):
   if k == 0: return '$0$'
   elif k == 1: return r'$\pi/2$'
   elif k == -1: return r'$-\pi/2$'
   elif k == 2: return r'$\pi$'
```

```
    elif k == -2: return r'$-\pi$'
    elif k%2 == 0: return '$' + str(k//2) +  r'\pi$'
    else: return '$' + str(k) +  r'\pi/2$'


def set_pi_ticks(a,b):
    base = pi/2
    m = floor(b/base)
    n = ceil(a/base)
    k = arange(n,m+1,dtype = int)
    # multiples of base
    return xticks(k*base, map(label,k) )
```

Then inserting `set_pi_ticks(a,b)` in the plot code returns Figure 4.8.



We review the derivative of sine and cosine. This is needed for the arcsine law (3.2.16). Recall the angle θ in radians is the length of the subtended arc (in red) in Figure 4.9. Following the figure, with $P = (x, y)$, we have $x = \cos\theta$, $y = \sin\theta$.

The key idea here is Archimedes' axiom [14], which states:

> If two convex curves share common initial and terminal points, and one lies inside the convex region defined by the other, than that one is shorter.

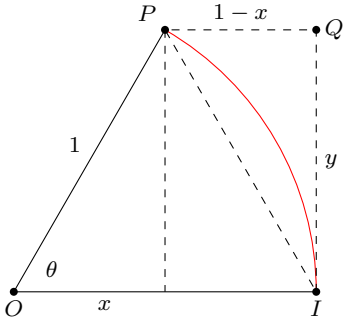


Fig. 4.9 Angle θ in the plane, $P = (x, y)$.

In Figure 4.9, there are three convex curves joining P and I: The line segment PI, the red arc, and the polygonal curve PQI. By Archimedes' axiom, the length of PI is less than the length of the red arc, which in turn is less than the length of PQI. Since the length of PI is greater than y, this implies

$$y < \theta < 1 - x + y,$$

or

$$\sin\theta < \theta < 1 - \cos\theta + \sin\theta.$$

Dividing by θ (here we assume $0 < \theta < \pi/2$),

$$1 - \frac{1-\cos\theta}{\theta} < \frac{\sin\theta}{\theta} < 1. \tag{4.1.26}$$

We use this to show (the definition of limit is in §A.7)

$$\lim_{\theta\to 0} \frac{\sin\theta}{\theta} = 1. \tag{4.1.27}$$

Since $\sin\theta$ is odd, it is enough to verify (4.1.27) for $\theta > 0$.

To this end, since $\sin^2\theta = 1 - \cos^2\theta$, from (4.1.26),

$$0 \le \frac{1-\cos\theta}{\theta} = \frac{1-\cos^2\theta}{\theta(1+\cos\theta)} = \frac{\sin\theta}{\theta}\cdot\frac{\sin\theta}{1+\cos\theta} \le \sin\theta \le \theta,$$

which implies

$$\lim_{\theta\to 0} \frac{1-\cos\theta}{\theta} = 0.$$

Taking the limit $\theta \to 0$ in (4.1.26), we obtain (4.1.27) for $\theta > 0$.

From (A.5.6),

$$\sin(\theta + t) = \sin\theta\cos t + \cos\theta\sin t,$$

so

$$\lim_{t\to 0} \frac{\sin(\theta+t) - \sin\theta}{t} = \lim_{t\to 0} \sin\theta\cdot\frac{\cos t - 1}{t} + \cos\theta\cdot\frac{\sin t}{t} = \cos\theta.$$

Thus the derivative of sine is cosine,

$$(\sin\theta)' = \cos\theta.$$

Similarly,

$$(\cos\theta)' = -\sin\theta.$$

Using the chain rule, we compute the derivative of the inverse $\arcsin x$ of $\sin\theta$. Since

$$\theta = \arcsin x \qquad \Longleftrightarrow \qquad x = \sin\theta,$$

we have

$$1 = x' = (\sin\theta)' = \theta'\cdot\cos\theta = \theta'\cdot\sqrt{1-x^2},$$

or

$$(\arcsin x)' = \theta' = \frac{1}{\sqrt{1-x^2}}.$$

We use this to compute the derivative of the arcsine law (3.2.16). With $x = \sqrt{\lambda}/2$, by the chain rule,

$$\begin{aligned}\left(\frac{2}{\pi}\arcsin\left(\frac{1}{2}\sqrt{\lambda}\right)\right)' &= \frac{2}{\pi}\frac{1}{\sqrt{1-x^2}}\cdot x' \\ &= \frac{2}{\pi}\frac{1}{\sqrt{1-\lambda/4}}\cdot\frac{1}{4\sqrt{\lambda}} = \frac{1}{\pi\sqrt{\lambda(4-\lambda)}}.\end{aligned} \tag{4.1.28}$$

This shows the derivative of the arcsine law is the density in Figure 3.11.

Exercises

Exercise 4.1.1 What is the y-intercept of the line tangent to $f(x) = x^2$ at the point $(1, 1)$?

Exercise 4.1.2 With $\exp x = e^x$, what are the first derivatives of $\exp(\exp x)$ and $\exp(\exp(\exp x))$?

Exercise 4.1.3 With $a > 0$, let $f(x) = \frac{1}{2}ax^2 - e^x$. Where is $f(x)$ convex, and where is it concave?

Exercise 4.1.4 With $P_n(x)$ the Legendre polynomial, use `num_legendre` to find the general formula for $P_n(0)$, $P_n(1)$, $P_n(-1)$, for $n = 1, 2, 3, \ldots$.

Exercise 4.1.5 For fixed $\beta > \alpha > 0$, find the maximizer $\hat{p}$ of

$$p^\alpha(1-p)^{\beta-\alpha}, \qquad 0 \le p \le 1.$$

Exercise 4.1.6 Compute the maximum and minimum of the second derivative of $\cos\theta$ over the interval $[a, b] = [-\pi/4, \pi/4]$. Use that to compute the upper and lower tangent parabolas at $\theta = 0$. Plots these parabolas against $\cos\theta$. Repeat everything with $[a, b] = [-\pi/2, \pi/2]$.

Exercise 4.1.7 Suppose $f''(x) \le 1/2$ and $f(x) \ge 0$ for all x. Write down the upper tangent parabola $p(x)$ at a point a. By (4.1.15), $0 \le f(x) \le p(x)$ for all x, so $p(x)$ is a nonnegative parabola, so its bottom is nonnegative. Calculate the bottom of the parabola to conclude

$$\left(f'(a)\right)^2 \le f(a).$$

Exercise 4.1.8 Use the Taylor series for $\log(1 + x)$ to show

$$\log 2 = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots.$$

Exercise 4.1.9 Compute the Taylor series for $\sin\theta$ and $\cos\theta$.

Exercise 4.1.10 Let $W = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$. Compute e^W (Exercise **2.2.16**).

Exercise 4.1.11 Using Newton's binomial theorem, show

$$\frac{1}{\sqrt{1-2u}} = 1 + u + \frac{1\cdot 3}{2!}u^2 + \frac{1\cdot 3\cdot 5}{3!}u^3 + \frac{1\cdot 3\cdot 5\cdot 7}{4!}u^4 + \dots$$

Exercise 4.1.12 If the convex dual of $f(x)$ is $g(p)$, and t is a constant, what is the convex dual of $f(x) + t$?

Exercise 4.1.13 If the convex dual of $f(x)$ is $g(p)$, and t is a constant, what is the convex dual of $f(x + t)$?

Exercise 4.1.14 If the convex dual of $f(x)$ is $g(p)$, and $t \neq 0$ is a constant, what is the convex dual of $f(tx)$?

Exercise 4.1.15 If the convex dual of $f(x)$ is $g(p)$, and $t \neq 0$ is a constant, what is the convex dual of $tf(x)$?

Exercise 4.1.16 Show $f(x)$ convex implies $e^{f(x)}$ convex.

Exercise 4.1.17 Let $0 < m < L$ be positive scalars, and let $r = m/L$. Show

$$\frac{m}{L} + \frac{L}{m} = r + \frac{1}{r} > 2$$

by searching for critical points over $r > 0$ (4.1.6).

Exercise 4.1.18 Let $0 < m < L$ be positive scalars. Show

$$q(t) = \Big(tm + (1-t)L\Big) \times \left(\frac{t}{m} + \frac{1-t}{L}\right)$$

satisfies

$$1 \leq q(t) \leq \frac{(m+L)^2}{4mL}, \qquad 0 \leq t \leq 1,$$

by searching for critical points over $0 < t < 1$ (4.1.6).

Exercise 4.1.19 If Q is a symmetric 2×2 matrix with positive eigenvalues L and m, show

$$1 \leq (u \cdot Qu)(u \cdot Q^{-1}u) \leq \frac{(m+L)^2}{4mL} \tag{4.1.29}$$

for every unit vector u. Use EVD and Exercise **4.1.18**.

Exercise 4.1.20 Derive (4.1.23) by applying (4.1.15) with f, x, a replaced by g, q, p, then inserting $q = f'(a)$ and $p = f'(x)$.

4.2 Entropy and Information

Let p be a probability, i.e. a number between 0 and 1. The *entropy* of p is

$$H(p) = -p\log p - (1-p)\log(1-p), \qquad 0 \le p \le 1. \tag{4.2.1}$$

This is also called *absolute entropy* to contrast with relative entropy which we see below.

To graph $H(p)$, we compute its first and second derivatives. Here the independent variable is p. By the product rule,

$$H'(p) = (-p\log p - (1-p)\log(1-p))' = -\log p + \log(1-p) = \log\left(\frac{1-p}{p}\right).$$

Thus $H'(p) = 0$ when $p = 1/2$, $H'(p) > 0$ on $p < 1/2$, and $H'(p) < 0$ on $p > 1/2$. Since this implies $H(p)$ is increasing on $p < 1/2$, and decreasing on $p > 1/2$, $p = 1/2$ is a global maximizer of the graph.

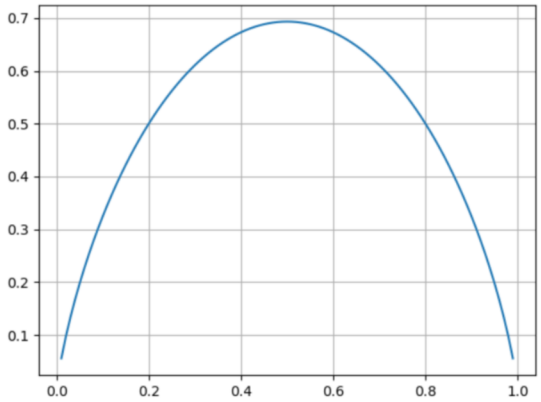


Fig. 4.10 The absolute entropy function $H(p)$.

As p increases, $1-p$ decreases, so $(1-p)/p$ decreases. Since log is increasing, as p increases, $H'(p)$ decreases. Thus $H(p)$ is concave.

Taking the second derivative, by the chain rule and the quotient rule,

$$H''(p) = \left(\log\left(\frac{1-p}{p}\right)\right)' = -\frac{1}{p(1-p)},$$

which is negative, leading to the strict concavity of $H(p)$.

A crucial aspect of Figure 4.10 is its limiting values at the edges $p = 0$ and $p = 1$,

$$H(0) = \lim_{p\to 0} H(p) \qquad \text{and} \qquad H(1) = \lim_{p\to 1} H(p).$$

Inserting $p = 0$ into $p\log p$ yields $0 \times (-\infty)$, so it is not at all clear what $H(0)$ should be. On the other hand, Figure 4.10 suggests $H(0) = 0$.

For the first limit, since $H(p)$ is increasing near $p = 0$, it is clear there is a definite value $H(0)$. The entropy is the sum of two terms, $-p \log p$, and $-(1-p)\log(1-p)$. When $p \to 0$, the second term approaches $-\log 1 = 0$, so $H(0)$ is the limit of the first term,

$$H(0) = -\lim_{p \to 0} p \log p.$$

When $p \to 0$, also $2p \to 0$. Replacing p by $2p$,

$$\begin{aligned} H(0) &= -\lim_{p \to 0} p \log p = -\lim_{p \to 0} 2p \log(2p) \\ &= \lim_{p \to 0} -2p \log 2 + 2H(0) = 2H(0). \end{aligned}$$

Thus $H(0) = 0$. Since $H(p)$ is symmetric, $H(1-p) = H(p)$, we also have $H(1) = 0$.



To explain the meaning of the entropy function $H(p)$, suppose a coin has heads-bias or heads-probability p. If p is near 1, then we have confidence the outcome of tossing the coin is heads, and, if p is near 0, we have confidence the outcome of tossing the coin is tails. If $p = 1/2$, then we have least information. Thus we can view the entropy as measuring a lack of information.

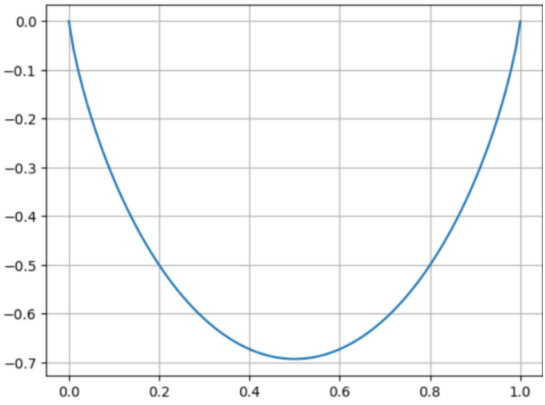


Fig. 4.11 The absolute information $I(p)$.

To formalize this, we define the *information* or *absolute information*

$$I(p) = p \log p + (1-p)\log(1-p), \qquad 0 \le p \le 1. \tag{4.2.2}$$

Then we have

Entropy and Information

Entropy equals negative information.

The clearest explanation of $H(p)$ is in terms of repeated coin-tossing, where it is shown $H(p)$ is the log of the number of outcomes with heads-proportion p. This is explained in §5.2.



The *logistic function* is

$$p = \sigma(x) = \frac{e^x}{1+e^x} = \frac{1}{1+e^{-x}}, \qquad -\infty < x < \infty. \tag{4.2.3}$$

By the quotient and chain rules, its derivative is

$$p' = -\frac{-e^{-x}}{(1+e^{-x})^2} = \sigma(x)(1-\sigma(x)) = p(1-p). \tag{4.2.4}$$

The logistic function, also called the *expit function* and the *sigmoid function*, is studied further in §5.2, where it used in coin-tossing and Bayes theorem.

The inverse of the logistic function is the *logit* function. The logit function is found by solving $p = \sigma(x)$ for x, obtaining

$$p = \sigma(x) \qquad \Longleftrightarrow \qquad x = \log\left(\frac{p}{1-p}\right). \tag{4.2.5}$$

The logit function is also called the *log-odds* function. Its derivative is

$$x' = \frac{1-p}{p} \cdot \left(\frac{p}{1-p}\right)' = \frac{1-p}{p} \cdot \frac{1}{(1-p)^2} = \frac{1}{p(1-p)}.$$



Let

$$Z(x) = \log\left(1+e^x\right). \tag{4.2.6}$$

Then $Z'(x) = \sigma(x)$ and $Z''(x) = \sigma'(x) = \sigma(1-\sigma) > 0$. This shows $Z(x)$ is strictly convex. We call $Z(x)$ the *cumulant-generating function*, to be consistent with random variable terminology (§5.3).

We compute the convex dual (§4.1) of $Z(x)$. By (4.1.6), the maximum

$$\max_x(px - Z(x))$$

is attained when $(px - Z(x))' = 0$, which happens when $p = Z'(x) = \sigma(x)$. Therefore the maximizer is the log-odds function $x = \sigma^{-1}(p)$. Inserting this into $px - Z(x)$, we obtain

$$\max_x(px - Z(x)) = p \log\left(\frac{p}{1-p}\right) - Z\left(\log\left(\frac{p}{1-p}\right)\right), \tag{4.2.7}$$

which simplifies to $I(p)$ (4.2.2).

Dual of Cumulant-Generating Function is Information

The convex dual of the cumulant-generating function is the information.

The derivative of $I(p)$ is

$$I'(p) = \log\left(\frac{p}{1-p}\right). \tag{4.2.8}$$

Then $I'(p)$ is the inverse of $Z'(x) = \sigma(x)$, as it should be (4.1.22).

From (4.2.8),

$$I''(p) = \frac{1}{p(1-p)}.$$

The multinomial extension of $I(p)$ is in §5.6.

Let p and q be two probabilities,

$$0 \le p \le 1, \qquad \text{and} \qquad 0 \le q \le 1.$$

When do we consider p and q close to each other? If p and q were just numbers, p and q are considered close if the distance $|p - q|$ is small or the distance squared $|p - q|^2$ is small. But here p and q are probabilities, so it makes sense to consider them close if their information content is close.

To this end, we define the *relative information* $I(p, q)$ by

$$I(p, q) = p \log\left(\frac{p}{q}\right) + (1-p) \log\left(\frac{1-p}{1-q}\right). \tag{4.2.9}$$

Then

$$I(q, q) = 0,$$

which agrees with our design goal of $I(p, q)$ measuring the divergence between the information in p and the information in q. Because $I(p, q)$ is not symmetric in p, q, we think of q as a base or reference probability, against which we compare p.

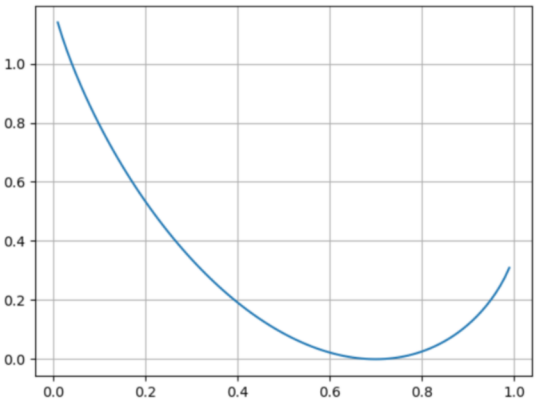


Fig. 4.12 The relative information $I(p, q)$ with $q = .7$.

Equivalently, instead of measuring relative information, we can measure the *relative entropy*,

$$H(p, q) = -I(p, q).$$

Since $-\log(x)$ is strictly convex,

$$\begin{aligned} I(p, q) &= -p \log\left(\frac{q}{p}\right) - (1-p)\log\left(\frac{1-q}{1-p}\right) \\ &> -\log\left(p \cdot \frac{q}{p} + (1-p)\cdot\frac{1-q}{1-p}\right) = -\log 1 = 0. \end{aligned}$$

This shows $I(p, q)$ is positive and $H(p, q)$ is negative, when $p \neq q$.

Since

$$I(p, q) = I(p) - p\log(q) - (1-p)\log(1-q),$$

the second derivatives of $I(p)$ and $I(p, q)$ agree, and $I(0) = 0 = I(1)$, $I(p, q)$ is well-defined for $p = 0$, and $p = 1$,

$$I(1, q) = -\log q, \qquad I(0, q) = -\log(1-q).$$

Taking derivatives (with independent variable p),

$$\frac{d^2}{dp^2} I(p, q) = I''(p) = \frac{1}{p(1-p)},$$

hence I is strictly convex in p. Thus q is a global minimizer of the graph of $I(p, q)$ (Figure 4.12). Also

$$\frac{d^2}{dq^2} I(p, q) = \frac{p}{q^2} + \frac{1-p}{(1-q)^2},$$

so $I(p,q)$ is strictly convex in q as well. In Exercise **4.5.4**, it is shown $I(p,q)$ is convex *in all directions* in the (p,q)-plane (Figure 4.13).

The clearest explanation of $H(p,q)$ is in terms of coin-tossing, where it is shown $H(p,q)$ is the log of the probability of a coin with heads-bias q having outcomes with heads-proportion p. This also is explained in §5.2.



Figure 4.13 is the surface plot of $I(p,q)$ as a function of two variables (p,q). This clearly exhibits the trough $p=q$ where $I(p,q)=0$, and the edges $q=0,1$ where $I(p,q)=\infty$.

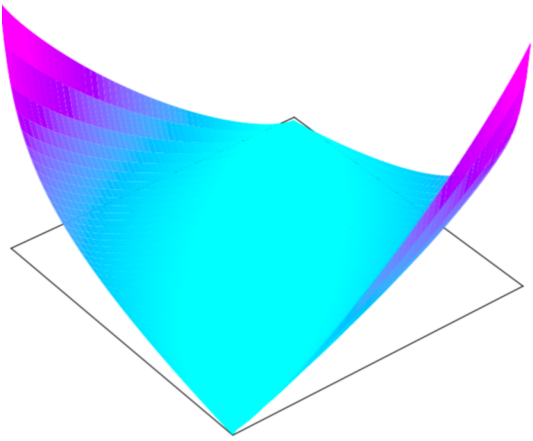


Fig. 4.13 Surface plot of $I(p,q)$ over the square $0 \le p \le 1$, $0 \le q \le 1$.

In `scipy`, $I(p,q)$ is incorrectly called `entropy`. For more on this terminology confusion, see the remarks at the end of §5.6. The code is as follows.

```
%matplotlib ipympl
from numpy import *
from  matplotlib.pyplot import *
from scipy.stats import entropy

def I(p,q): return entropy([p,1-p],[q,1-q])

ax = axes(projection = '3d')
ax.set_axis_off()

p = arange(0,1,.01)
q = arange(0,1,.01)
p,q = meshgrid(p,q)
```

```
# surface
ax.plot_surface(p,q,I(p,q), cmap = 'cool')

# square
ax.plot([0,1,1,0,0],[0,0,1,1,0],linewidth = .5,c = "k")

show()
```

Exercises

Exercise 4.2.1 Check (4.2.7) simplifies to the information (4.2.2).

Exercise 4.2.2 Compute

$$\min_{0 \le p \le 1} I''(p).$$

Exercise 4.2.3 Let $0 < q < 1$ be a constant. What is the convex dual of

$$Z(x, q) = \log\left(qe^x + 1 - q\right)?$$

Exercise 4.2.4 Use Python to plot the entropy $H(p)$ and $\sqrt{p(1-p)}$. Use `scipy.optimize.newton` to find where they are equal.

Exercise 4.2.5 The relative information $I(p, q)$ has minimum zero when $p = q$. Use the lower tangent parabola (4.1.13) of $I(x, q)$ at q and Exercise **4.2.2** to show

$$I(p, q) \ge 2(p - q)^2.$$

For $q = 0.7$, plot both $I(p, q)$ and $2(p - q)^2$ as functions of $0 < p < 1$.

4.3 Multi-Variable Calculus

Let

$$f(x) = f(x_1, x_2, \ldots, x_d)$$

be a scalar function of a point $x = (x_1, x_2, \ldots, x_d)$ in $\mathbf{R}^d$, and suppose v is a unit vector in $\mathbf{R}^d$. Then, along the line $x(t) = x + tv$, $g(t) = f(x + tv)$ is a function of the single variable t. Hence its derivative $g'(0)$ at $t = 0$ is well-defined. This rate of change is the directional derivative of $f(x)$ at x in the direction v.

More explicitly, the *directional derivative* of $f(x)$ *at* x *in the direction* v is

$$\left.\frac{d}{dt}\right|_{t=0} f(x + tv). \tag{4.3.1}$$

When we select specific directions, the directional derivatives have specific names. Let e_1, e_2, ..., e_d be the standard basis in $\mathbf{R}^d$. The *partial derivative* in the k-th direction, $k = 1, \ldots, d$, is

$$\frac{\partial f}{\partial x_k}(x) = \left.\frac{d}{dt}\right|_{t=0} f(x + te_k).$$

The partial derivative in the k-th direction is just the one-dimensional derivative considering x_k as the independent variable, with all other features x_j's held constant.

The multi-variable chain rule, below, expresses the directional derivative (4.3.1) in terms of the partial derivatives of $f(x)$.



Below we exhibit the multi-variable chain rule in two ways. The first interpretation is geometric, and involves motion in time and directional derivatives. This interpretation is relevant to gradient descent, §7.3.

The second interpretation is combinatorial, and involves repeated compositions of functions. This interpretation is relevant to computing gradients in networks, specifically back propagation §4.4, §7.2.

These two interpretations work together when training neural networks, §7.4.



For the first interpretation of the chain rule, suppose the features x_1, x_2, ..., x_d of a point x are functions of a single variable t (usually time), so we have

$$x_1 = x_1(t), \qquad x_2 = x_2(t), \qquad \ldots, \qquad x_d = x_d(t).$$

Inserting these into $f(x_1, x_2, \ldots, x_d)$, we obtain a function

$$f(t) = f(x_1(t), x_2(t), \ldots, x_d(t))$$

of a single variable t. Then we have

Multi-Variable Chain Rule

With $f(t) = f(x_1(t), x_2(t), \ldots, x_d(t))$,

$$\frac{df}{dt} = \frac{\partial f}{\partial x_1} \cdot \frac{dx_1}{dt} + \frac{\partial f}{\partial x_2} \cdot \frac{dx_2}{dt} + \cdots + \frac{\partial f}{\partial x_d} \cdot \frac{dx_d}{dt}.$$

The *gradient* of $f(x)$ is the vector

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_d} \right). \tag{4.3.2}$$

The $\mathbf{R}^d$-valued function $x(t) = (x_1(t), x_2(t), \ldots, x_d(t))$ represents a curve or path in $\mathbf{R}^d$, and the vector

$$x'(t) = (x_1'(t), x_2'(t), \ldots, x_d'(t))$$

represents its *velocity* at time t.

With this notation, the chain rule may be written

$$\frac{df}{dt} = \nabla f(x(t)) \cdot x'(t).$$

Let $v = (v_1, v_2, \ldots, v_d)$. The simplest application of the multi-variable chain rule is to select $x(t) = x + tv$. Then the chain rule becomes

Directional Derivative Formula

The directional derivative of $f(x)$ in the direction v is the dot product of the gradient $\nabla f(x + tv)$ and v,

$$\frac{d}{dt} f(x + tv) = \nabla f(x + tv) \cdot v. \tag{4.3.3}$$

In particular,

$$\left. \frac{d}{dt} \right|_{t=0} f(x + tv) = \nabla f(x) \cdot v. \tag{4.3.4}$$

Let $f(x)$ be a quadratic function,

$$f(x) = \frac{1}{2} x \cdot Qx - b \cdot x,$$

where Q is a $d \times d$ symmetric matrix and b is a vector. Then

$$f(x) = \frac{1}{2} \sum_{i,j=1}^{d} q_{ij} x_i x_j - \sum_{j=1}^{d} b_j x_j,$$

so

$$\frac{\partial f}{\partial x_i} = \frac{1}{2} \sum_{j=1}^{d} q_{ij} x_j + \frac{1}{2} \sum_{j=1}^{d} q_{ji} x_j - b_i = (Qx - b)_i.$$

Here we used $Q = Q^t$. This shows

Gradient of a Quadratic

$$f(x) = \frac{1}{2} x \cdot Qx - b \cdot x \qquad \Longrightarrow \qquad \nabla f(x) = Qx - b. \tag{4.3.5}$$



In §7.7, we will need to compute the gradient of a function $f(W)$ of matrices W. Towards this, recall the collection of matrices with a fixed shape may be added and scaled. It follows if W and V are matrices with the same shape, then $W + sV$ also has the same shape, for any scalar s.

If G and V are two matrices with the same shape, we think of $\text{trace}(V^t G)$ as a dot product between G and V. This is consistent with the definition of norm squared (2.2.17). By analogy with (4.3.4), we say

Directional Derivative Matrix Formula

A matrix G is the gradient of $f(W)$ at W if

$$\left.\frac{d}{ds}\right|_{s=0} f(W + sV) = \text{trace}(V^t G). \qquad \text{for all } V. \tag{4.3.6}$$

Then the gradient G has the same shape as W.

We use this result in Chapter 7.



Here is an example of the second interpretation of the chain rule. Suppose

$$r = f(x) = \sin x, \qquad s = g(x) = \frac{1}{1 + e^{-x}},$$
$$t = h(x) = x^2, \qquad u = r + s + t, \qquad y = k(u) = \cos u.$$

These are multiple functions in composition, as in Figure 4.14.

The input variable is x and the output variable is y. The intermediate variables are r, s, t, u. Suppose $x = \pi/4$. Then

$$x, r, s, t, u, y = 0.79, 0.71, 0.69, 0.62, 2.01, -0.43.$$

To compute derivatives, start with

$$\frac{dy}{du} = k'(u) = -\sin u = -0.90.$$

Next, to compute dy/dr, the chain rule says

$$\frac{dy}{dr} = \frac{dy}{du}\frac{du}{dr} = -0.90 * 1 = -0.90,$$

and similarly,

$$\frac{dy}{ds} = \frac{dy}{dt} = -0.90.$$

By the chain rule,

$$\frac{dy}{dx} = \frac{dy}{dr} \cdot \frac{dr}{dx} + \frac{dy}{ds} \cdot \frac{ds}{dx} + \frac{dy}{dt} \cdot \frac{dt}{dx}.$$

By (4.2.4), $s' = s(1-s) = 0.22$, so

$$\frac{dr}{dx} = \cos x = 0.71, \qquad \frac{ds}{dx} = s(1-s) = 0.22, \qquad \frac{dt}{dx} = 2x = 1.57.$$

We obtain

$$\frac{dy}{dx} = -0.90 * 0.71 - 0.90 * 0.22 - 0.90 * 1.57 = -2.25.$$

The chain rule is discussed in further detail in §4.4.

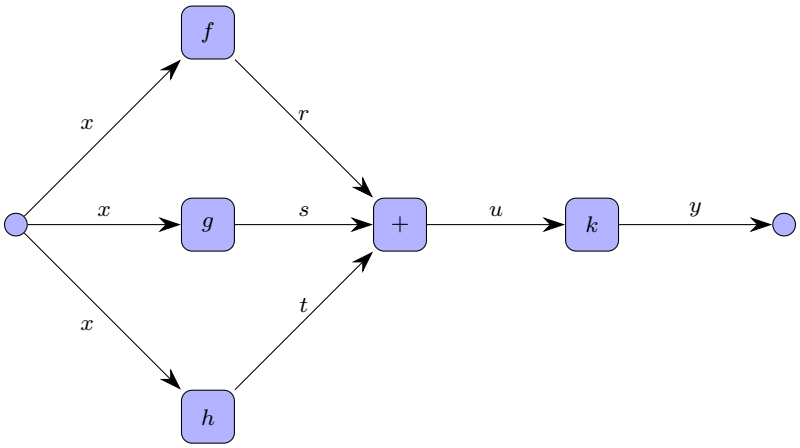


Fig. 4.14 Composition of multiple functions.



By (2.2.2),

$$\nabla f(x) \cdot v = |\nabla f(x)|\,|v| \cos\theta,$$

where θ is the angle between v and $\nabla f(x)$. Since $-1 \leq \cos\theta \leq 1$, we conclude

Gradient is Direction of Greatest Increase

Let v be a unit vector and let a be a point. As the direction v varies, the directional derivative varies between two extremes

$$-|\nabla f(a)| \leq \nabla f(a) \cdot v \leq |\nabla f(a)|.$$

The directional derivative achieves its greatest value when v points in the direction of $\nabla f(a)$, and achieves its least value in the opposite direction, when v points in the direction of $-\nabla f(a)$.



A *critical point* is a point x^* satisfying

$$\nabla f(x^*) = 0.$$

A *local minimizer* is a point x^* with $f(x) \geq f(x^*)$ for x near x^*. If the inequality holds for all x, x^* is a *global minimizer*. A *local maximizer* is a point x^* with $f(x) \leq f(x^*)$ for x near x^*. If the inequality holds for all x, x^* is a *global maximizer*. As in the single-variable case,

A Maximizer or Minimizer is a Critical Point

A minimizer is a critical point, and a maximizer is a critical point.



We now address the existence of a global minimizer of a function $f(x)$. Intuitively, if $f(x)$ goes up to $+\infty$ when x is far away, then its graph must have a minimizer at some point x^*. This notion is made precise as follows.

Let $f(x)$ be a function. A *level* is a scalar c determining a *sublevel set* $f(x) \leq c$. A *bound* is a scalar C determining a bounded set $|x| \leq C$.

A function $f(x)$ is *proper* if the sublevel set $f(x) \leq c$ is bounded for every level c: For every level c, there is a bound C so that

$$f(x) \leq c \qquad \Longrightarrow \qquad |x| \leq C. \tag{4.3.7}$$

In other words, $f(x)$ is proper if *every sublevel set is bounded*. The functions in Figure 4.3 and 4.4 are proper.

This is same as saying $f(x)$ rises to $+\infty$ as $|x| \to \infty$. The exact formula for the bound C, which depends on the level c and the function $f(x)$, is not

important for our purposes. What matters is the existence of *some* bound C for *each* level c.

More vividly, suppose x is scalar, and think of the graph of $y = f(x)$ as the cross-section of a river. Then $f(x)$ is proper if the river never floods its banks, no matter how much it rains. So $y = \sin x$ is not proper, but $y = x^2 + \sin x$ is proper.

Existence of Global Minimizer

A continuous proper function has at least one global minimizer.

To see this, pick any point a. Then, by properness, the sublevel set $f(x) \leq f(a)$ is bounded. Since $f(x)$ is continuous, the sublevel set is closed (§A.8). Since a continuous function on a bounded and closed set has a minimizer (§A.8), there is a minimizer x^*. Since for all x outside this sublevel set, we have $f(x) > f(a)$, x^* is a global minimizer. This completes the proof.

From §4.1, we already know a strongly convex function has a global minimizer. In fact, strongly convex functions are proper (Exercise **4.5.22**).

In §4.5, we establish a converse to this result: *If a function is strictly convex and has a minimizer, then it is proper.*



What does it mean for $f(x)$ to not be proper? Unpacking the definition, $f(x)$ is not proper if there is *some* level c with no corresponding bound C. This means there is some level c and a sequence $x_1, x_2, \ldots$ with $f(x_n) \leq c$ and $|x_n| \to \infty$.

Using properness, we can establish the existence of residual minimizers, as promised in §2.6.

Properness of Residual on Row Space

Let A be a matrix, and b a vector with dimensions so that the residual

$$f(x) = |Ax - b|^2 \tag{4.3.8}$$

is defined. Then $f(x)$ is proper on the row space of A, and there is a global minimizer.

To see this, suppose $f(x)$ is not proper. In this case, by (4.3.7), there would be a level c and a sequence $x_1, x_2, \ldots$ in the row space of A satisfying $|x_n| \to \infty$ and $f(x_n) \leq c$ for $n \geq 1$.

Let $x'_n = x_n/|x_n|$. Then x'_n are unit vectors in the row space of A, hence x'_n is a bounded sequence. From §A.8, this implies x'_n subconverges to some x^*, necessarily a unit vector in the row space of A.

By the triangle inequality (2.2.4),

$$|Ax_n'| = \frac{1}{|x_n|}|Ax_n| \le \frac{1}{|x_n|}(|Ax_n - b| + |b|) \le \frac{1}{|x_n|}(\sqrt{c} + |b|).$$

Moreover Ax_n' subconverges to Ax^*. Since $|x_n| \to \infty$, taking the limit $n \to \infty$,

$$|Ax^*| = \lim_{n\to\infty} |Ax_n'| \le \frac{1}{\infty}(\sqrt{c} + |b|) = 0.$$

Thus x^* is both in the row space of A and in the nullspace of A. Since the row space and the nullspace are orthogonal, this implies $x^* = 0$. But we can't have $1 = |x^*| = |0| = 0$. This contradiction shows there is no such sequence x_n, and we conclude $f(x)$ is proper.

When the row space is the source space,

Properness of Residual

When the $N \times d$ matrix A has rank d,

$$f(x) = |Ax - b|^2 \tag{4.3.9}$$

is proper on $\mathbf{R}^d$.

As a consequence,

Existence of Residual Minimizer

Let A be a matrix and b a vector so that the residual

$$|Ax - b|^2 \tag{4.3.10}$$

is well-defined. Then there is a residual minimizer x^* in the row space of A,

$$|Ax^* - b|^2 \le |Ax - b|^2 \tag{4.3.11}$$

for all x.



Let $f(x)$ be a function of $x = (x_1, x_2, \ldots, x_d)$. The *second partial derivatives* are

$$\frac{\partial^2 f}{\partial x_i \partial x_j}, \qquad 1 \le i, j \le d,$$

and the *second derivative* of $f(x)$ is the $d \times d$ symmetric matrix

$$D^2 f(x) = \begin{pmatrix} \dfrac{\partial^2 f}{\partial x_1 \partial x_1} & \dfrac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots \\ \dfrac{\partial^2 f}{\partial x_2 \partial x_1} & \dfrac{\partial^2 f}{\partial x_2 \partial x_2} & \cdots \\ \cdots & \cdots & \cdots \\ \dfrac{\partial^2 f}{\partial x_d \partial x_1} & \dfrac{\partial^2 f}{\partial x_d \partial x_2} & \cdots \end{pmatrix}$$

Differentiating (4.3.3) and using the chain rule again,

Second Directional Derivative

Let $Q(t) = D^2 f(x + tv)$. Then

$$\frac{d^2}{dt^2} f(x + tv) = v \cdot Q(t) v. \tag{4.3.12}$$

Exercises

Exercise 4.3.1 Let $I(p, q)$ be the relative information (4.2.9), and let I_{pp}, I_{pq}, I_{qp}, I_{qq} be the second partial derivatives. If Q is the second derivative matrix

$$Q = \begin{pmatrix} I_{pp} & I_{pq} \\ I_{qp} & I_{qq} \end{pmatrix},$$

show

$$\det(Q) = \frac{(p-q)^2}{p(1-p)q^2(1-q)^2}.$$

Exercise 4.3.2 Let $J(x) = J(x_1, x_2, \ldots, x_d)$ equal

$$J(x) = \frac{1}{2}(x_1 - x_2)^2 + \frac{1}{2}(x_2 - x_3)^2 + \cdots + \frac{1}{2}(x_{d-1} - x_d)^2 + \frac{1}{2}(x_d - x_1)^2.$$

Compute $Q = D^2 J$.

Exercise 4.3.3 Show

$$f(x) = \frac{1}{2}|Ax - b|^2 \qquad \Longrightarrow \qquad \nabla f(x) = A^t A x - A^t b.$$

Conclude $\nabla f(x)$ is in the row space of A.

Exercise 4.3.4 Let $J(M, b)$ be a function of two variables M and b. If $J(M, b)$ is proper, then $J(M, 0)$ is proper.

4.4 Back Propagation

In this section, we compute outputs and derivatives on a graph. We already did this for the graph (4.14), but now we systematize things so that we can write code.

We consider two cases, when the graph is a chain, or the graph is a network of neurons. The derivatives are taken with respect to the outputs at each node of the graph. In §7.2, we consider a third case, and compute outputs and derivatives on a neural network.

To compute node outputs, we do forward propagation. To compute derivatives, we do back propagation. Corresponding to the three cases, we will code three versions of forward and back propagation. In all cases, back propagation depends on the chain rule.

The chain rule (§4.1) states

$$r = f(x), y = g(r) \qquad \Longrightarrow \qquad \frac{dy}{dx} = \frac{dy}{dr} \cdot \frac{dr}{dx}.$$

In this section, we work out the implications of the chain rule on repeated compositions of functions.

Suppose

$$r = f(x) = \sin x, \qquad s = g(r) = \frac{1}{1 + e^{-r}}, \qquad y = h(s) = s^2.$$

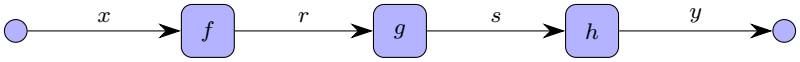


Fig. 4.15 Composition of three functions in a chain.

These are three functions f, g, h composed in a *chain* (Figure 4.15). The chain in Figure 4.15 has five nodes and four edges. There is one input node ● (no incoming edge from another node) and one output node ● (no outgoing edge to another node). The outgoing signals at the first four nodes are x, r, s, y. The incoming signals at the last four nodes are x, r, s, y.

Start with $x = \pi/4$. Evaluating the functions in order,

$$x = 0.785, \qquad r = 0.707, \qquad s = 0.670, \qquad y = 0.448.$$

Notice these values are evaluated in the forward direction: x then r then s then y. This is *forward propagation.*

Now we evaluate the derivatives of the output y with respect to x, r, s,

$$\frac{dy}{dx}, \qquad \frac{dy}{dr}, \qquad \frac{dy}{ds}.$$

With the above values for x, r, s, we have

$$\frac{dy}{ds} = 2s = 2 * 0.670 = 1.340.$$

Since g is the logistic function, by (4.2.4),

$$g'(r) = g(r)(1 - g(r)) = s(1 - s) = 0.670 * (1 - 0.670) = 0.221.$$

From this,

$$\frac{dy}{dr} = \frac{dy}{ds} \cdot \frac{ds}{dr} = 1.340 * g'(r) = 1.340 * 0.221 = 0.296.$$

Repeating one more time,

$$\frac{dy}{dx} = \frac{dy}{dr} \cdot \frac{dr}{dx} = 0.296 * \cos x = 0.296 * 0.707 = 0.209.$$

Thus the derivatives are

$$\frac{dy}{dx} = 0.209, \qquad \frac{dy}{dr} = 0.296, \qquad \frac{dy}{ds} = 1.340.$$

Notice the derivatives are evaluated in the backward direction: First $dy/dy = 1$, then dy/ds, then dy/dr, then dy/dx. This is *back propagation*.



Here is another example. Let

$$\begin{aligned} r &= x^2, \\ s &= r^2 = x^4, \\ y &= s^2 = x^8. \end{aligned}$$

This is the same function $h(x) = x^2$ composed with itself three times. With $x = 5$, we have

$$x = 5, \qquad r = 25, \qquad s = 625, \qquad y = 390625.$$

Applying the chain rule as above, check that

$$\frac{dy}{dx} = 625000, \qquad \frac{dy}{dr} = 62500, \qquad \frac{dy}{ds} = 1250.$$



To evaluate x, r, s, y in Figure 4.15, first we built the list of functions and the list of derivatives

```
from numpy import *
set_printoptions(legacy = "1.25")

def f(x): return sin(x)
def g(r): return 1/(1+ exp(-r))
def h(s): return s**2
# this for next example
def k(t): return cos(t)

func_chain = [f,g,h]

def df(x): return cos(x)
def dg(r): return g(r)*(1-g(r))
def dh(s): return 2*s
# this for next example
def dk(t): return -sin(t)

der_chain = [df,dg,dh]
```

Then we evaluate the output vector $x = (x, r, s, y)$, leading to the first version of forward propagation,

```
# first version: chains

def forward_prop(x_in,func_chain):
   x = [x_in]
   while func_chain:
      f = func_chain.pop(0)  # first func
      x_out = f(x_in)
      x.append(x_out) # insert at end
      x_in = x_out
   return x

from numpy import *
x_in = pi/4
x = forward_prop(x_in,func_chain)
x
```

Now we evaluate the gradient vector $\delta = (dy/dx, dy/dr, dy/ds, dy/dy)$. Since $dy/dy = 1$, we set

```
# dy/dy = 1
delta_out = 1
```

The code for the first version of back propagation is

```
# first version: chains

def backward_prop(delta_out,x,der_chain):
   delta = [delta_out]
   while der_chain:
      # discard last output
      x.pop(-1)
      df = der_chain.pop(-1) # last der
      der = df(x[-1])
      # chain rule -- multiply by previous der
      der = der * delta[0]
      delta.insert(0,der) # insert at start
   return delta

delta = backward_prop(delta_out,x,der_chain)
delta
```

Note forward propagation must be run prior to back propagation.



To apply this code to the second example, use

```
d = 3
func_chain, der_chain = [h]*d, [dh]*d
x_in, delta_out = 5, 1

x = forward_prop(x_in,func_chain)
delta = backward_prop(delta_out,x,der_chain)
x, delta
```



Now we work with the network in Figure 4.16, using the multi-variable chain rule (§4.3). The functions are

$$\begin{aligned} a &= f(x,y) = x+y, \\ b &= g(y,z) = \max(y,z), \\ J &= h(a,b) = ab. \end{aligned}$$

The composite function is

$$J = (x+y)\max(y,z),$$

Here there are three input nodes, labeled 0, 1, 2, three hidden nodes, 3, 4, 5, and an output node, 6. Starting with inputs $(x, y, z) = (1, 2, 0)$, and plugging in, we obtain the outgoing signals at the nodes

$$(x_0, x_1, x_2, x_3, x_4, x_5, x_6) = (x, y, z, a, b, J, \texttt{None}) = (1, 2, 0, 3, 2, 6, \texttt{None})$$

(Figure 4.18). This is forward propagation. At the output node, the output is `None`.

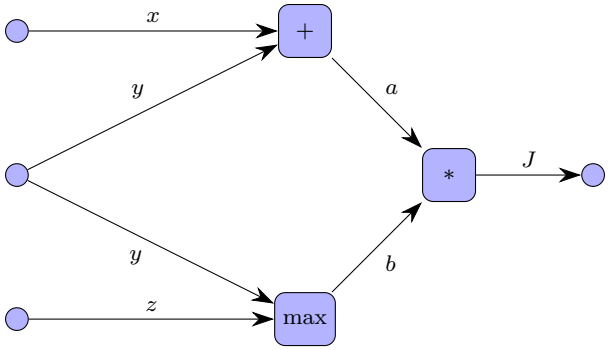


Fig. 4.16 A network composition [33].



Now we compute the derivatives

$$\frac{\partial J}{\partial x}, \quad \frac{\partial J}{\partial y}, \quad \frac{\partial J}{\partial z}, \quad \frac{\partial J}{\partial a}, \quad \frac{\partial J}{\partial b}, \quad \frac{\partial J}{\partial J}.$$

This we do in reverse order. It's clear $\partial J/\partial J = 1$. We compute

$$\frac{\partial J}{\partial a} = b = 2, \qquad \frac{\partial J}{\partial b} = a = 3.$$

Then

$$\frac{\partial a}{\partial x} = 1, \qquad \frac{\partial a}{\partial y} = 1.$$

Let

$$\mathbf{1}(y > z) = \begin{cases} 1, & y > z, \\ 0, & y < z. \end{cases}$$

By Figure 4.17, since $y = 2$ and $z = 0$,

$$\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1, \qquad \frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0.$$

By the chain rule,

$$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial a}\frac{\partial a}{\partial x} = 2 * 1 = 2,$$
$$\frac{\partial J}{\partial y} = \frac{\partial J}{\partial a}\frac{\partial a}{\partial y} + \frac{\partial J}{\partial b}\frac{\partial b}{\partial y} = 2 * 1 + 3 * 1 = 5,$$
$$\frac{\partial J}{\partial z} = \frac{\partial J}{\partial b}\frac{\partial b}{\partial z} = 3 * 0 = 0.$$

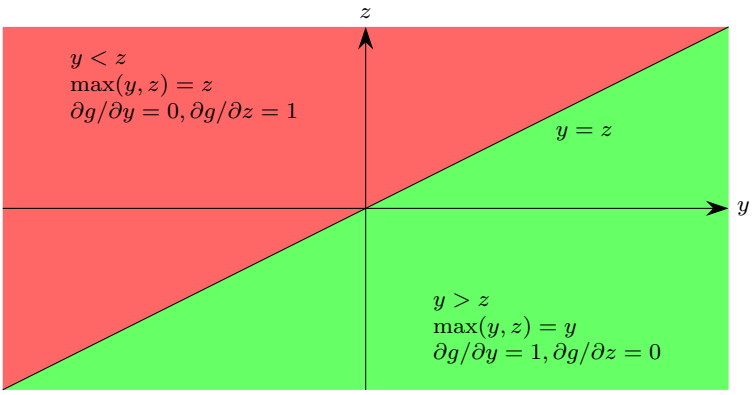


Fig. 4.17 The function $g = \max(y, z)$.

Hence we have

$$\left(\frac{\partial J}{\partial x}, \frac{\partial J}{\partial y}, \frac{\partial J}{\partial z}, \frac{\partial J}{\partial a}, \frac{\partial J}{\partial b}, \frac{\partial J}{\partial J}\right) = (2, 5, 0, 2, 3, 1).$$

The outputs (blue) and the derivatives (red) are displayed in Figure 4.18.

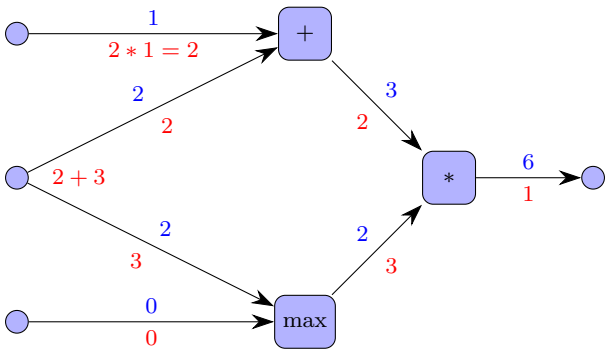


Fig. 4.18 Forward and backward propagation [33].

Summarizing, by the chain rule,

- derivatives are computed backward,
- derivatives along successive edges are multiplied,
- derivatives along several outgoing edges are added.



To label these derivatives systematically, look at a specific node, say in Figure 4.16 look at the node  . This node has outgoing signal a and incoming signals x and y. Corresponding to these, we have the *upstream derivative* $\partial J/\partial a$ and *downstream derivatives* $\partial J/\partial x$ and $\partial J/\partial y$. But there is a problem here, since $\partial J/\partial y$ can be considered a downstream derivative on two separate edges. Because of this, and since there is only one outgoing signal, we label the derivative at this node to be the upstream derivative,

$$\delta_3 = \frac{\partial J}{\partial x_3} = \frac{\partial J}{\partial a}.$$

With this notation, we have

$$\left(\frac{\partial J}{\partial x_0}, \frac{\partial J}{\partial x_1}, \frac{\partial J}{\partial x_2}, \frac{\partial J}{\partial x_3}, \frac{\partial J}{\partial x_4}, \frac{\partial J}{\partial x_5}\right) = \left(\frac{\partial J}{\partial x}, \frac{\partial J}{\partial y}, \frac{\partial J}{\partial z}, \frac{\partial J}{\partial a}, \frac{\partial J}{\partial b}, \frac{\partial J}{\partial J}\right),$$

and $\partial J/\partial x_6 =$ `None` since $x_6 =$ `None`.



To do this in general, recall a directed graph (§3.3) as in Figure 4.16 has a weight matrix $W = (w_{ij})$ with w_{ij} nonzero when (j, i) is an edge.

Suppose a directed graph has d nodes, labeled 0, 1, 2, ..., $d-1$, and, for each node i, let x_i be the outgoing signal. Then $x = (x_0, x_1, x_2, \ldots, x_{d-1})$ is the *outgoing vector*.

In the case of Figure 4.16, $d = 7$ and

$$x = (x_0, x_1, x_2, x_3, x_4, x_5, x_6) = (x, y, z, a, b, J, \texttt{None}).$$

With this order, the adjacency matrix is

$$W = \begin{pmatrix} 0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0 \\ 1&1&0&0&0&0&0 \\ 0&1&1&0&0&0&0 \\ 0&0&0&1&1&0&0 \\ 0&0&0&0&0&1&0 \end{pmatrix}.$$

This we code as a list of lists,

```
d = 7
w = [ [None]*d for _ in range(d) ]

w[3][0] = w[3][1] = w[4][1] = w[4][2] = 1
w[5][3] = w[5][4] = w[6][5] = 1
```

In a weighed directed graph (§3.3), we code the weights w_{ij} as numeric scalars or `None`.

Once we have the outgoing vector x, for each node i, let

$$x_i^- = (w_{i0}x_0, w_{i1}x_1, w_{i2}x_2, \ldots, w_{i,d-1}x_{d-1}). \tag{4.4.1}$$

This is the *incoming signal list* at node i. Here we adopt the convention that `None` times anything is `None`, and any resulting `None` entry in the list is to be discarded.

An *activation function* at node i is a function f_i of the incoming signal list x_i^-. Then the outgoing signal at node i is

$$x_i = f_i(x_i^-). \tag{4.4.2}$$

By the chain rule,

$$\frac{\partial x_i}{\partial x_j} = \begin{cases} \dfrac{\partial f_i}{\partial x_j} \cdot w_{ij}, & \text{if } (j,i) \text{ is an edge,} \\ \texttt{None}, & \text{if } (j,i) \text{ is not an edge.} \end{cases} \tag{4.4.3}$$

The *incoming vector* is

$$x^- = (x_0^-, x_1^-, x_2^-, \ldots, x_{d-1}^-).$$

Then x^- is a list of lists. In the case of Figure 4.16,

$$x^- = (x_0^-, x_1^-, x_2^-, x_3^-, x_4^-, x_5^-, x_6^-),$$

where

```
x0minus =  [None,None,None,None,None,None,None]
x1minus =  [None,None,None,None,None,None,None]
x2minus =  [None,None,None,None,None,None,None]
x3minus =  ['x','y',None,None,None,None,None]
x4minus =  [None,'y','z',None,None,None,None]
x5minus =  [None,None,None,'a','b',None,None]
x6minus =  [None,None,None,None,None,None,'J']
```

This is before discarding. After discarding,

```
x0minus =  [ ]
x1minus =  [ ]
x2minus =  [ ]
x3minus =  ['x','y']
x4minus =  ['y','z']
x5minus =  ['a','b']
x6minus =  ['J']
```

The activation functions are

```
activate = [None]*d

activate[3] = lambda x,y: x+y
activate[4] = lambda y,z: max(y,z)
activate[5] = lambda a,b: a*b
```

To compute the outgoing signal x_i at node i, we collect the incoming signals x_i^- following (4.4.1)

```
def incoming(x,w,i):
   return [ w[i][j] * outgoing(x,w,j) for j in range(d) if
    ↪ w[i][j] ]
```

then plug them into the activation function,

```
def outgoing(x,w,i):
   if x[i] != None: return x[i]
   elif activate[i] != None: return activate[i](*incoming(x,w,i))
   else: return None
```

Here `*` is the unpacking operator.

Summarizing, at each node i, we have the outgoing signal x_i, and a *list* x_i^- of incoming signals.



Now we can define what is meant by a network. A node with an attached activation function is a *neuron*. A *network* is a directed weighed graph where nodes are neurons. The code in this section works for any network without cycles. In §7.2, we specialize to neural networks. Neural networks are networks with a restricted class of activation functions.

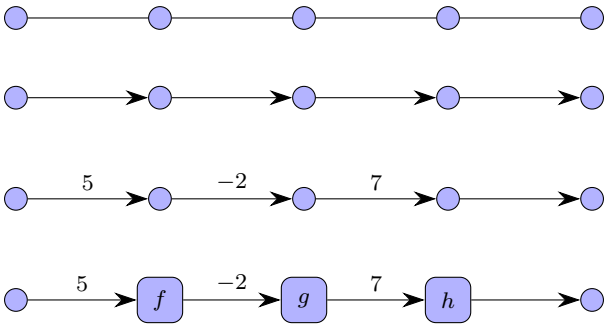


Fig. 4.19 Graph, directed graph, weighed directed graph, network.



Let x_{in} be the outgoing vector over the input nodes. If there are m input nodes, and d nodes in total, then the length of x_{in} is m, and the length of x is d. In the example above, $x_{\text{in}} = (x, y, z)$.

We assume the list of nodes is ordered so that the initial portion of the list of nodes is the list of input nodes.

```
# m = len(x_in)
# x[:m] = x_in
```

Here is the second version of forward propagation.

```
# second version: networks

def forward_prop(x_in,w):
   d = len(w)
   x = [None]*d
   m = len(x_in)
   x[:m] = x_in
   for i in range(m,d): x[i] = outgoing(x,w,i)
   return x

x_in = [1,2,0]
x = forward_prop(x_in,w)
x
```

For this code to work, we assume there are no cycles in the graph: All backward paths end at input nodes, and all forward paths end at output nodes.



The output function J is a function of all node outputs. For Figure 4.16, this means J is a function of x, y, z, a, b.

Therefore, at each node i, we have the derivatives

$$\delta_j = \frac{\partial J}{\partial x_j}(x), \qquad j = 0, 1, 2, \ldots, d-1.$$

Then $\delta = (\delta_0, \delta_1, \delta_2, \ldots, \delta_{d-1})$ is the *gradient vector*. We first compute the derivatives of J with respect to the signals incoming to the output nodes, and we assume these derivatives are assembled into a vector δ_{out}.

In Figure 4.16, $x_6 = \texttt{None}$, so $\delta_6 = \texttt{None}$. Next, at node with output J,

$$\delta_5 = \frac{\partial J}{\partial x_5} = \frac{\partial J}{\partial J} = 1.$$

Hence $\delta_{\text{out}} = (1, \texttt{None})$.

We assume the list of nodes is ordered so that the terminal portion of the list of nodes is the list of output nodes, together with the nodes incoming to the output nodes

```
# m = len(delta_out)
# delta[-m:] = delta_out
```

For each i, j, let

$$g_{ij} = \frac{\partial f_i}{\partial x_j}.$$

Then we have a $d \times d$ *gradient matrix* $g = (g_{ij})$. When i is not a neuron and (j, i) is not an edge, we set g_{ij} equal `None`.

These are the *local* derivatives, not the derivatives obtained by the chain rule: Even though we saw above $\partial J/\partial y = 1$, here the local derivative is zero, since J does not depend directly on y.

For the example above, with $(x_0, x_1, x_2, x_3, x_4, x_5) = (x, y, z, a, b, J)$ and $x_6 = \texttt{None}$,

```
g = [ [None]*d for _ in range(d) ]

# g specified only at neurons

g[3][0] = lambda x,y: 1
g[3][1] = lambda x,y: 1
g[4][1] = lambda y,z: 1 if y >= z else 0
g[4][2] = lambda y,z: 1 if z > y else 0
g[5][3] = lambda a,b: b
g[5][4] = lambda a,b: a
```

By the chain rule and (4.4.3),

$$\frac{\partial J}{\partial x_j} = \sum_{j \to i} \frac{\partial J}{\partial x_i} \cdot \frac{\partial x_i}{\partial x_j} = \sum_{j \to i} \frac{\partial J}{\partial x_i} \cdot \frac{\partial f_i}{\partial x_j} \cdot w_{ij},$$

so

$$\delta_j = \sum_{j \to i} \delta_i \cdot g_{ij} \cdot w_{ij}.$$

Here j is fixed and the sum is over i. The code is

```
def derivative(x,delta,g,j):
   if delta[j] != None: return delta[j]
   else:
      return sum([ derivative(x,delta,g,i) *
    ↪ g[i][j](*incoming(x,w,i)) * w[i][j] for i in range(d) if
    ↪ w[i][j] ])
```

This leads to our second version of back propagation,

```
# second version: networks

def backward_prop(x,delta_out,g):
   d = len(g)
   delta = [None]*d
   m = len(delta_out)
   delta[-m:] = delta_out
   for j in range(d-m): delta[j] = derivative(x,delta,g,j)
   return delta

delta_out = [1,None]
delta = backward_prop(x,delta_out,g)
delta
```

Exercises

Exercise 4.4.1 For the network in Figure 4.15, use the second version of the propagation code and $x = \pi/4$ to compute the output vector and the gradient vector

$$x = (x, r, s, y), \qquad \delta = \left(\frac{dy}{dx}, \frac{dy}{dr}, \frac{dy}{ds}, \frac{dy}{dy}\right).$$

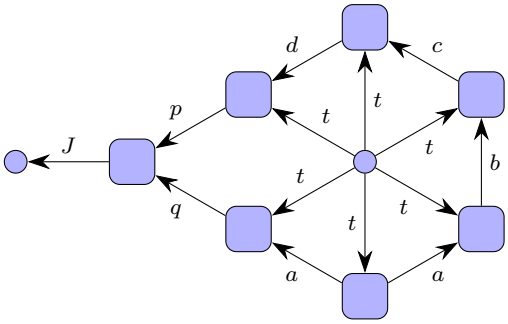


Fig. 4.20 A network with outgoing signals.

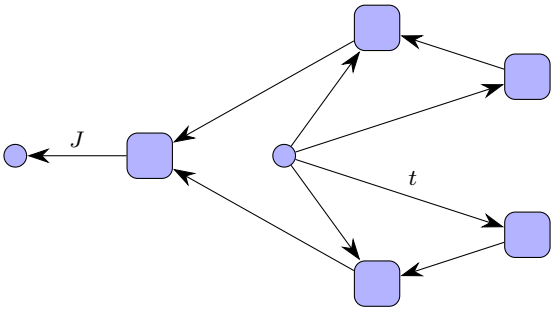


Fig. 4.21 Another network.

Exercise 4.4.2 In Figure 4.20, the activation function at each neuron is the sum of the squares of the incoming signals to that neuron. Starting with $t = 1$, compute

$$x = (x_0, x_1, x_2, x_3, x_4, x_6, x_7, x_8) = (t, a, b, c, d, p, q, J, \texttt{None}),$$

and the corresponding derivatives of J,

$$\delta = \left(\frac{\partial J}{\partial x_0}, \frac{\partial J}{\partial x_1}, \frac{\partial J}{\partial x_2}, \frac{\partial J}{\partial x_3}, \frac{\partial J}{\partial x_4}, \frac{\partial J}{\partial x_5}, \frac{\partial J}{\partial x_6}, \frac{\partial J}{\partial x_7}, \frac{\partial J}{\partial x_8} \right).$$

Do this by hand and by coding. You should get

$$x = (1, 1, 2, 5, 26, 677, 2, 458333, \texttt{None}),$$
$$\delta = (18449628, 5632648, 2816320, 704080, 70408, 1354, 4, 1, \texttt{None}).$$

Exercise 4.4.3 Compute the outgoing vector x and gradient vector δ for the network in Figure 4.21. The outgoing signal at each neuron is the sum

of the squares of the incoming signals at that neuron. Here the input node signal is the variable t, so both x and δ will have powers of t in them.

4.5 Convexity

Let $f(x)$ be a scalar function of points $x = (x_1, \ldots, x_d)$ in $\mathbf{R}^d$. For example, in two dimensions,

$$f(x) = f(x_1, x_2) = \max(|x_1|, |x_2|), \qquad f(x) = f(x_1, x_2) = \frac{x_1^2}{4} + x_2^2$$

are scalar functions of points in $\mathbf{R}^2$.

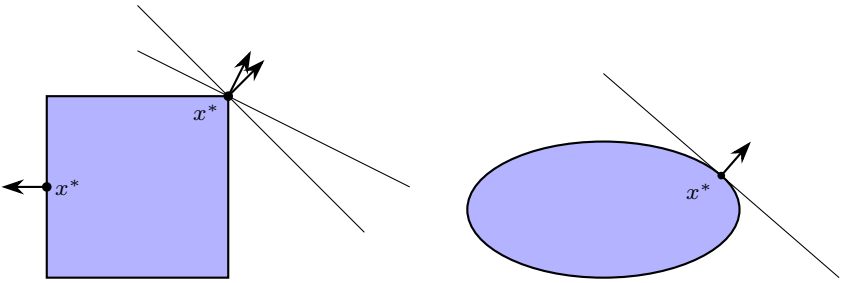


Fig. 4.22 Level sets and sublevel sets in two dimensions.

A *level set* of $f(x)$ is the set of points x satisfying $f(x) = 1$. This is the level set corresponding to level 1. One can have level sets corresponding to any level c, $f(x) = c$. In two dimensions, level sets are also called *level curves*.

For example, the variance ellipse $x \cdot Qx = 1$ is a level set. The perimeters (not their interiors) of the square and ellipse in Figure 4.22 are level sets

$$\max(|x_1|, |x_2|) = 1, \qquad \frac{x_1^2}{16} + \frac{x_2^2}{4} = 1.$$

The level curves of

$$f(x) = f(x_1, x_2) = \frac{x_1^2}{16} + \frac{x_2^2}{4}$$

are in Figure 4.23.

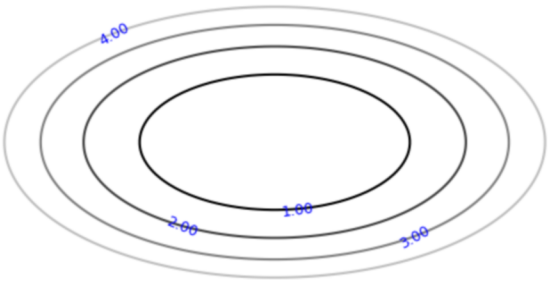


Fig. 4.23 Level curves in two dimensions.

A *sublevel set* of $f(x)$ is the set of points x satisfying $f(x) \leq 1$. This is the sublevel set corresponding to level 1. One can have sublevel sets corresponding to any level c, $f(x) \leq c$. Sublevel sets were used in the definition of proper functions (4.3.7).

In Figure 4.22, the (blue) interior of the square, together with the boundary of the square, is a sublevel set. Similarly, the interior of the ellipse, together with the ellipse itself, is a sublevel set.

The level set $f(x) = 1$ is the boundary of the sublevel set $f(x) \leq 1$. Thus the square and the ellipse in Figure 4.22 are boundaries of their respective sublevel sets, and the unit variance ellipsoid $x \cdot Qx = 1$ is the boundary of the sublevel set $x \cdot Qx \leq 1$.

Interiors and boundaries are discussed in more detail in §A.8.

Given points x_0 and x_1 in $\mathbf{R}^d$, let $v = x_1 - x_0$ be the vector joining them. Then

$$(1-t)x_0 + tx_1 = x_0 + tv, \qquad 0 \leq t \leq 1.$$

The *line segment* $[x_0, x_1]$ *joining* x_0, x_1 consists of linear combinations

$$x = (1-t)x_0 + tx_1 = x_0 + tv, \qquad 0 \leq t \leq 1$$

(Figure 4.24).

More generally, given points x_1, x_2, ..., x_N, a linear combination

$$t_1x_1 + t_2x_2 + \cdots + t_Nx_N$$

is a *convex combination* if $t_1, t_2, \ldots, t_N$ are nonnegative, and

$$t_1 + t_2 + \cdots + t_N = 1.$$

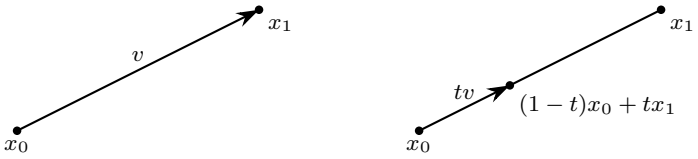


Fig. 4.24 Line segment $[x_0, x_1]$.

A *convex set* is a set K that contains the line segment joining any two points in it: *If x_0 and x_1 are in K, then the line segment $[x_0, x_1]$ is in K.* To be consistent with sublevel sets, we only consider[1] convex sets that contain their boundaries.

Assume x, y, z are three points in a convex set K, and suppose $r+s+t = 1$, with r, s, t positive. Then

$$w = \frac{rx}{r+s} + \frac{sy}{r+s}$$

is in the line segment $[x, y]$. Since K is convex, w is in K. Repeating the same logic with w and z,

$$rx + sy + tz = (r+s)w + tz$$

is also in K.

Since this argument may be repeated with any number of points, we conclude

Convex Sets Contain Any Convex Combination

If a dataset $x_1, x_2, \ldots, x_N$ is in a convex set K, then any convex combination

$$x = t_1 x_1 + t_2 x_2 + \cdots + t_N x_N$$

is also in K.

[1] We only consider *closed* convex sets, see §A.8 for more on closed sets.

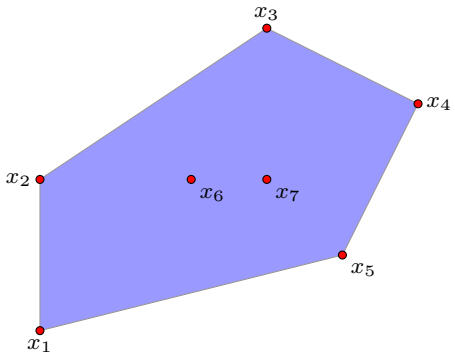


Fig. 4.25 Convex hull of x_1, x_2, x_3, x_4, x_5, x_6, x_7.

The set of all convex combinations of a dataset $x_1, x_2, \ldots, x_N$ is the *convex hull* of $x_1, x_2, \ldots, x_N$ (Figure 4.25).

The convex hull of a dataset is a convex set. Conversely, if K is convex and contains a dataset $x_1, x_2, \ldots, x_N$, then K contains the convex hull of the dataset. Thus the convex hull of a dataset is also the smallest convex set containing the dataset.

The interiors of the square and the ellipse in Figure 4.22, together with their boundaries, are convex sets.

The following code generates convex hulls,

```
from scipy.spatial import ConvexHull
from numpy import *

from numpy.random import default_rng
samples = default_rng().random

# 30 random points in the plane
points = samples((30, 2))

hull = ConvexHull(points)
```

and this plots the facets of the convex hull

```
from matplotlib.pyplot import *

scatter(points[:,0], points[:,1])

for facet in hull.simplices:
   plot(points[facet,0], points[facet,1], 'k-')

facet = hull.simplices[0]
plot(points[facet, 0], points[facet, 1], 'r--')
```

```
grid()
show()
```

resulting in Figure 4.26.

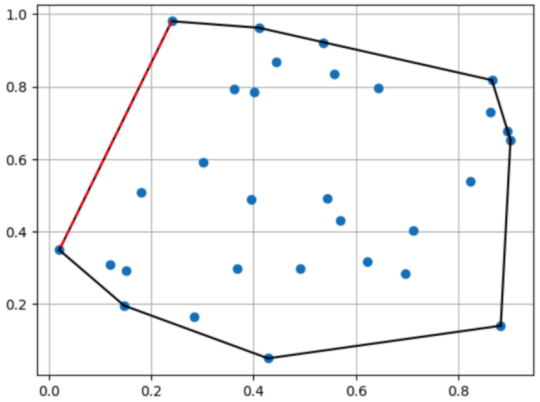


Fig. 4.26 A convex hull with one facet highlighted.

Let K be a convex set, and let x_0 be a point. We search for a point x^* in K that is nearest to x_0 (Figure 4.27). Since $|x - x_0|$ is the distance between x and x_0, the nearest point x^* satisfies

$$|x^* - x_0|^2 = \min_{x \text{ in } K} |x - x_0|^2.$$

Here we minimize the distance squared, since the distance minimizer is the same as the distance squared minimizer.

Unique Nearest Point in Convex Set

Given any point x_0 and any convex set K, there is a unique point x^* in K nearest to x_0.

The results in §A.8 guarantee the existence of a distance minimizer x^*. We show there is exactly one distance minimizer x^* in K (Figure 4.27).

Let $\delta = |x^* - x_0|^2$ equal the minimum distance squared, and let x_1^* be another point in K at the same distance from x_0 as x^*. Then

$$|x^* - x_0|^2 = \delta = |x_1^* - x_0|^2.$$

If $x_a = (x^* + x_1^*)/2$ is the average of x^* and x_1^*, since K is convex, x_a is in K, hence $|x_a - x_0|^2 \geq \delta$. By expanding the squares, check that

$$4|x_a - x_0|^2 + |x^* - x_1^*|^2 = 2|x^* - x_0|^2 + 2|x_1^* - x_0|^2.$$

Since x_a is in K, the left side is no less than $4\delta + |x^* - x_1^*|^2$. On the other hand, the right side equals 4δ. This implies $|x^* - x_1^*|^2 = 0$, or $x^* = x_1^*$, completing the proof.

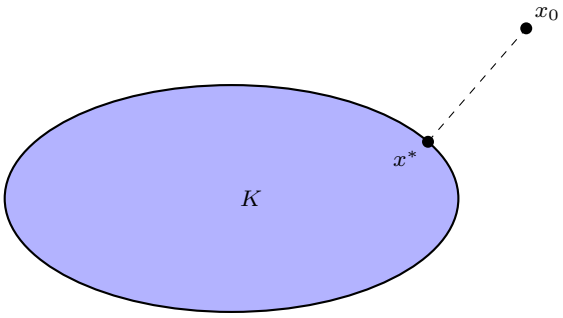


Fig. 4.27 A convex set K has a unique nearest point to any x_0.

Hyperplanes were introduced in §2.5, but we re-introduce them here for convenience. Let m be a nonzero vector, and let b be a scalar.

A *hyperplane* (m, b) consists of all x satisfying

$$m \cdot x + b = 0. \tag{4.5.1}$$

This hyperplane is *orthogonal to* m.

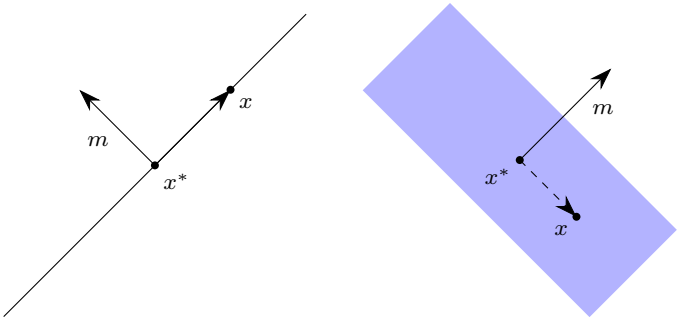


Fig. 4.28 Hyperplanes in two and three dimensions.

A hyperplane (m, b) *passes through* a point x^* if $m \cdot x^* + b = 0$. When this happens,

$$m \cdot (x - x^*) = m \cdot x - m \cdot x^* = m \cdot x + b = 0,$$

so the hyperplane orthogonal to m and passing through x^* may be written

$$m \cdot (x - x^*) = 0. \tag{4.5.2}$$

When $x^* = 0$, this hyperplane is the orthogonal complement $m^\perp$ (2.4.9). In general, x is in the hyperplane orthogonal to m and passing through x^* iff $x - x^*$ is in $m^\perp$.

Rescaling m does not change the hyperplane: the hyperplanes (m, b) and (tm, tb) are equal (Exercise **4.5.2**). Because of this, it is often convenient to assume m is a unit vector.

In two dimensions, a hyperplane is a line; in three dimensions, a hyperplane is a plane (Figure 4.28). In one dimension, (4.5.2) implies a hyperplane is a single point, a threshold $x = x^*$.

The *level* of a point x corresponding to a hyperplane (m, b) is the scalar

$$y = m \cdot x + b. \tag{4.5.3}$$

The level depends on the hyperplane, and the hyperplane consists of points at level zero.

A hyperplane *separates* the sample space into negative, zero, and positive levels

$$y < 0, \qquad y = 0, \qquad y > 0.$$

Let $x_1, x_2, \ldots, x_N$ be a dataset to one side of a hyperplane,

$$m \cdot x_k + b \le 0, \qquad k = 1, 2, \ldots, N,$$

and let $x = \sum_{k=1}^N t_k x_k$ be in the convex hull of the dataset. Then

$$m \cdot x + b = m \cdot \left(\sum_{k=1}^N t_k x_k \right) + \left(\sum_{k=1}^N t_k \right) b = \sum_{k=1}^N t_k (m \cdot x_k + b) \le 0,$$

so x lies on the same side of the hyperplane. Summarizing,

Convex Hulls and Levels of Hyperplanes

If a dataset lies to one side of a hyperplane, then its convex hull lies on the same side of the hyperplane.



Let K be a convex set containing a point x^*, and x_0 a point. We say a hyperplane (m, b) passing through x^* *separates* K *and* x_0 if

$$m \cdot (x_0 - x^*) > 0$$

and

$$m \cdot (x - x^*) \leq 0, \qquad \text{for every } x \text{ in } K. \tag{4.5.4}$$

This can only happen when x_0 is not in K, and x_0 and K are on opposite sides of the hyperplane (Figure 4.29).

Separating Hyperplane I

Let K be a convex set, let x_0 be a point not in K, and let x^* be the point in K nearest to x_0. If $m = x_0 - x^*$, then the hyperplane passing through x^* and orthogonal to m separates x_0 and K.

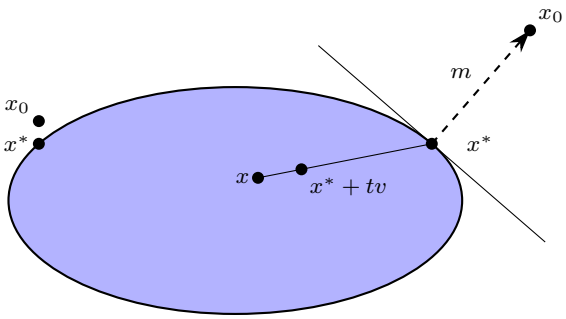


Fig. 4.29 Separating hyperplane I.

A diagram of the proof is Figure 4.29. If x is in K, then by convexity, the line segment $[x^*, x]$ is in K, hence $x^* + tv$, $v = x - x^*$, is in K for $0 \leq t \leq 1$. Since x^* is the point of K nearest to x_0,

$$|x^* - x_0|^2 \leq |x^* + tv - x_0|^2 \qquad \text{for} \qquad 0 \leq t \leq 1.$$

Expanding, we have

$$|x^* - x_0|^2 \leq |x^* - x_0|^2 + 2t(x^* - x_0) \cdot v + t^2|v|^2, \qquad 0 \leq t \leq 1.$$

Canceling $|x^* - x_0|^2$ then canceling t, we obtain

$$0 \leq 2(x^* - x_0) \cdot v + t|v|^2, \qquad 0 \leq t \leq 1.$$

Since this is true for small positive t, sending $t \to 0$ results in $v \cdot (x^* - x_0) \geq 0$. Since $m = x_0 - x^*$, $v = x - x^*$, we obtain

$$m \cdot (x - x^*) \leq 0 \qquad \text{and} \qquad m \cdot (x_0 - x^*) > 0.$$



Now suppose x^* is a point in the boundary of a convex set K. Since x^* is in K, we cannot find a hyperplane separating K and x^*. In this case, the best we can hope for is a hyperplane passing through x^*, with K to one side of the hyperplane:

$$x \text{ in } K \qquad \Longrightarrow \qquad m \cdot (x - x^*) \leq 0. \tag{4.5.5}$$

Such a hyperplane is a *supporting hyperplane for K at x^**. Examples of supporting hyperplanes are in Figure 4.22.

At a given x^*, there may be more than one supporting hyperplane: for example, the corner point of the square in Figure 4.22. Also, the hyperplane in Figure 4.29 separating K and x_0 is a supporting hyperplane for K at x^*.

Here is the basic result relating convex sets and supporting hyperplanes.

Supporting Hyperplane

Let K be a convex set and let x^* be a point on the boundary of K. Then there is a supporting hyperplane for K at x^*.

If x^* is in the boundary of K, there are points x_0 not in K and close to x^* (Figure 4.29). Applying the separating hyperplane theorem to x_0, and taking the limit $x_0 \to x^*$, leads to a supporting hyperplane at x^*. We skip the details.

As a consequence of this result, we have (Exercise **4.5.14**)

Mean of Dataset is in Interior of Convex Hull

Suppose a dataset $x_1, x_2, \ldots, x_N$ does not lie in a hyperplane, and let K be the convex hull of the dataset. Then the mean μ lies in the interior of K.

The interior of K is the part of K that is not on the boundary of K. To be precise, remember a *ball with center a and radius $r > 0$* is the set of points x satisfying $|x - a| \leq r$. Then we say a point a is in the *interior of K* if there is a ball centered at a and wholly contained in K. For example, in Figure 4.30, μ is in the interior of K and x is in the boundary of K.

Not every set K has an interior. For example, a hyperplane has no interior , and the convex hull of a dataset has interior iff the dataset does not lie in

a hyperplane. Also, if a set K lies on one side of a hyperplane and x is in the interior of K, then x is not in the hyperplane (Exercises **4.5.16**), **4.5.18**, **4.5.17**).

These ideas can be used to show supporting hyperplanes characterize convex sets in the following sense: If K is a closed set with interior, and through every point x^* in the boundary of K, there is a supporting hyperplane for K at x^*, then K is convex.

For more on interiors and boundaries, see §A.8.



We summarize the above results together with a result from §2.9 to obtain the main convexity result for datasets.

Full-Rank Dataset

For any dataset, the following are equivalent.

1. The centered dataset matrix is full-rank.
2. The corresponding centered dataset spans the sample space.
3. The dataset does not lie in a hyperplane in sample space.
4. The convex hull of the dataset has interior.

Whenever a dataset satisfies[2] any of these equivalent conditions, we say the dataset is *full-rank*. This result is an extension of a previous result with the same name in §2.9.

Since the rows of the centered dataset matrix constitute the centered dataset, 1 and 2 are equivalent. That 2 and 3 are equivalent was derived in §2.5. Above we saw 3 implies the mean is in the interior of the convex hull, so 3 $\implies$ 4. Conversely, if the convex hull has interior, then it cannot lie in a hyperplane, because a hyperplane has no interior (Exercise **4.5.16**). This establishes the equivalence of 3 and 4.

This result is the basis for the analysis of regression in Chapter 7.



Given a dataset $x_1, x_2, \ldots, x_N$ with convex hull K, we can *scale* K *towards the mean* μ as follows. If $0 \leq t \leq 1$, let

$$tK = \{\mu + t(x - \mu) \mid x \text{ in } K\}.$$

When $t = 1$, $tK = K$. When $t = 0$, tK is the mean μ.

[2] For the equivalence 1 $\iff$ 2, we assume more samples than features: the dataset matrix is $N \times d$ with $N \geq d$.

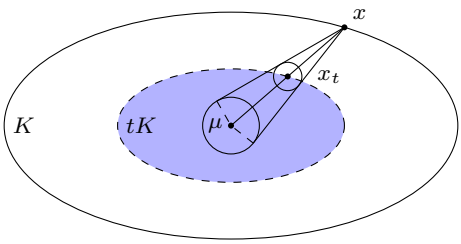


Fig. 4.30 tK lies in the interior of K: $x_t = (1-t)\mu + tx$ is in the interior of K.

More generally, if K is a convex set and μ is any point in K, the same definition *scales* K *towards* μ.

tK is in the Interior of K

Let $0 < t < 1$, and let K be a convex set with a point μ in the interior of K. If tK is K scaled towards μ, then tK is in the interior of K. Conversely, if x is in the interior of K, then x is in tK for some $0 < t < 1$.

In particular, if a dataset $x_1, x_2, \ldots, x_N$ does not lie in a hyperplane, and K is its convex hull, then the mean μ lies in the interior of K, so this result applies. The proof uses Figure 4.30 (Exercise **4.5.19**).

Recall a *bit* is either zero or one. A dataset $x_1, x_2, \ldots, x_N$ is a *two-class* dataset if there are bits $p_1, p_2, \ldots, p_N$ corresponding to each sample. Then the two classes correspond to $p = 1$ and $p = 0$ respectively.

Recall the level of a sample x relative to a hyperplane (m, b) is the scalar $y = m \cdot x + b$. A hyperplane is *separating* if the levels satisfy

$$\begin{array}{ll} y \le 0, & \text{if } p = 0, \\ y \ge 0, & \text{if } p = 1, \end{array} \qquad \text{for every sample } x. \tag{4.5.6}$$

When there is a separating hyperplane, we say the two-class dataset is *separable*. When a two-class dataset is not separable, we say it is *inseparable*. Because samples are separated by level, positive, negative, or zero, a separating hyperplane is also called a *decision boundary*.

The dataset $x_1, x_2, \ldots, x_N$ *lies in the hyperplane* (m, b) if

$$m \cdot x_k + b = 0, \qquad k = 1, 2, \ldots, N. \tag{4.5.7}$$

When a two-class dataset lies in a hyperplane, the hyperplane is separating, so the question of separability is only interesting when the dataset does not lie in a hyperplane.

In a two-class dataset, not only do we have the mean μ of the dataset, we also have the means μ_0, μ_1 of the two classes.

If a two-class dataset does not lie in a hyperplane and the dataset is separable, then the means μ_0, μ_1 of the two classes are distinct (Exercise **4.5.1**).

Hyperplane Separation Theorem

Let $x_1, x_2, \ldots, x_N$ be a two-class dataset and assume neither class lies in a hyperplane. Let K_0 and K_1 be the convex hulls of the two classes. Then

$$\text{the dataset is separable} \iff K_0 \cap K_1 \text{ has no interior.} \tag{4.5.8}$$

In one direction, suppose there is a hyperplane (m, b) separating K_0 and K_1. If x is any point in $K_0 \cap K_1$, then we have $m \cdot x + b \leq 0$ and $m \cdot x + b \geq 0$, so $m \cdot x + b = 0$. Hence $K_0 \cap K_1$ lies in the hyperplane. But by Exercise **4.5.16**, a hyperplane has no interior. Hence $K_0 \cap K_1$ has no interior.

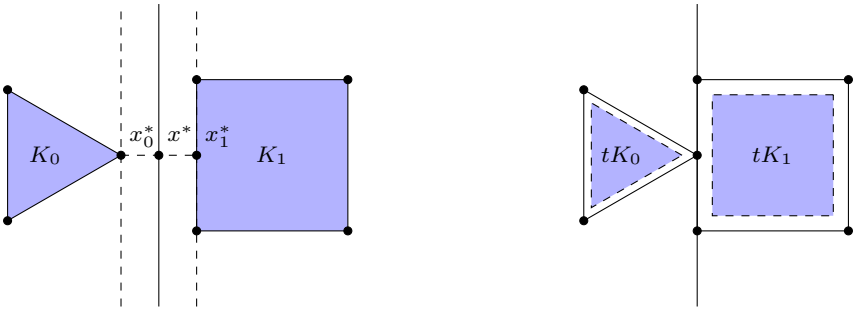


Fig. 4.31 Hyperplane separation theorem.

For the reverse direction, let x_0^*, x_1^* be points in K_0, K_1 closest to each other, so

$$|x_1^* - x_0^*|^2 = \min_{\substack{x_0 \text{ in } K_0 \\ x_1 \text{ in } K_1}} |x_1 - x_0|^2. \tag{4.5.9}$$

Since K_0 and K_1 are closed and bounded, the results in §A.8 guarantee the existence of minimizers x_0^* and x_1^* in K_0 and K_1 respectively. If x_1^* is in the interior of K_1, there is a point x_1 on the line segment $[x_0^*, x_1^*]$ closer to x_0^* and in K_1, contradicting (4.5.9). Hence x_1^* must be on the boundary of K_1. Similarly, x_0^* is on the boundary of K_0.

There are two cases, either $x_0^* \neq x_1^*$ or $x_0^* = x_1^*$. In the first case, since the minimum distance between K_0 and K_1 is positive, x_1^* is not in K_0, and x_0^* is not in K_1. Let

$$m = x_1^* - x_0^*, \qquad x^* = \frac{1}{2}(x_0^* + x_1^*).$$

Since x_0^* is the point in K_0 closest to x_1^*, by separating hyperplane I, $m \cdot (x - x_0^*) \leq 0$ for x in K_0. Since x_1^* is the point in K_1 closest to x_0^*, by separating hyperplane I, $(-m) \cdot (x - x_1^*) \leq 0$ for x in K_1, or $m \cdot (x - x_1^*) \geq 0$ for x in K_1. Since

$$\begin{aligned} m \cdot (x - x^*) &= m \cdot (x - x_0^*) - \frac{1}{2} m \cdot m \leq 0, \qquad \text{for } x \text{ in } K_0, \\ m \cdot (x - x^*) &= m \cdot (x - x_1^*) + \frac{1}{2} m \cdot m \geq 0, \qquad \text{for } x \text{ in } K_1, \end{aligned}$$

the hyperplane passing through x^* and orthogonal to m separates K_0 and K_1 (Figure 4.31). This hyperplane is the *maximum-margin hyperplane*. This completes the first case.

In the second case, neither class lies in a hyperplane, so both classes are full-rank, hence both K_0 and K_1 have interiors. Let μ_0 and μ_1 be any points in the interiors of K_0 and K_1, and let tK_0 and tK_1 be K_0 and K_1 scaled towards μ_0 and μ_1. By Exercise **4.5.19**, when $0 < t < 1$, tK_0 and tK_1 are in the interiors of K_0 and K_1 respectively.

If we assume $K_0 \cap K_1$ has no interior, then tK_0 and tK_1 do not intersect, and the first case applies to tK_0 and tK_1. Hence there is a unit vector m_t and a scalar b_t with the hyperplane (m_t, b_t) separating tK_0 and tK_1,

$$m_t \cdot x_0 + b_t \leq 0, \qquad m_t \cdot x_1 + b_t \geq 0, \qquad \text{for } x_0 \text{ in } tK_0 \text{ and } x_1 \text{ in } tK_1.$$

Since the sequence m_t is bounded, m_t subconverges to some m, as $t \to 1$. Since μ_0 and μ_1 are in tK_0 and tK_1 for $0 < t < 1$,

$$m_t \cdot \mu_0 \leq -b_t \leq m_t \cdot \mu_1,$$

for all $0 < t < 1$. From this, the sequence b_t is bounded, so b_t subconverges to some b as $t \to 1$ (§A.8). It follows (m, b) separates the interior of K_0 and the interior of K_1. Since any points in the boundaries of K_0 and K_1 are limits of points in the corresponding interiors, (m, b) separates K_0 and K_1,

$$m \cdot x_0 + b \leq 0, \qquad m \cdot x_1 + b \geq 0, \qquad \text{for } x_0 \text{ in } K_0 \text{ and } x_1 \text{ in } K_1.$$



More generally, let K_0 and K_1 be any two convex sets. We say K_0 and K_1 are *separable* if there is a hyperplane with the levels satisfying

$$\begin{aligned} y &\leq 0, \qquad \text{for } x \text{ in } K_0, \\ y &\geq 0, \qquad \text{for } x \text{ in } K_1. \end{aligned} \tag{4.5.10}$$

When K_0 and K_1 are the convex hulls of the classes of a two-class dataset, by taking[3] convex combinations, (4.5.6) implies (4.5.10).

Then the statement of the *hyperplane separation theorem* remains valid: *If K_0 and K_1 are closed convex sets with interior, then*

$$K_0,\ K_1 \text{ are separable} \iff K_0 \cap K_1 \text{ has no interior.} \tag{4.5.11}$$

Moreover, the above proof remains valid with one modification: tK_0 must be replaced by $tK_0 \cap B_t$, where B_t is a ball whose radius tends to infinity as $t \to 1$. We skip the details.



The above completes our analysis of datasets and convex sets. As mentioned before, this analysis is the basis for regression training (§7.5, §7.6).

Turning to convex functions, in the multi-variable setting convexity is defined by reducing to the single-variable case.

Let $f(x)$ be a function of d features $x = (x_1, x_2, \dots, x_d)$. If we fix a point a and a direction v, then $g(t) = f(a + tv)$ is a function of the single variable t.

A function $f(x)$ is *convex* if $g(t) = f(a+tv)$ is convex in t, for every point a and direction v. A function $f(x)$ is *strictly convex* if $g(t) = f(a + tv)$ is strictly convex, for every point a and direction v.

As in the scalar case, a function is *concave* or *strictly concave* if $-f$ is convex or strictly convex respectively.

It can be shown $f(x)$ is convex iff

$$f(t_1x_1 + \cdots + t_Nx_N) \le t_1f(x_1) + \cdots + t_Nf(x_N), \tag{4.5.12}$$

for any convex combination of points x_1, x_2, …, x_N in $\mathbf{R}^d$.

Since multi-variable convexity is defined in terms of single-variable convexity, we have the following.

Second Directional Derivatives and Convexity

A function $f(x)$ is convex when

$$\left.\frac{d^2}{dt^2}\right|_{t=0} f(x + tv) \ge 0, \qquad \text{for all } v, \tag{4.5.13}$$

and $f(x)$ is strictly convex when

$$\left.\frac{d^2}{dt^2}\right|_{t=0} f(x + tv) > 0, \qquad \text{for all } v \ne 0. \tag{4.5.14}$$

[3] Use Exercise **4.5.24** with the hyperplane (m, b), and the level zero.

In terms of derivatives, by (4.3.12), $f(x)$ is convex when $D^2 f(x) \geq 0$, and $f(x)$ is strictly convex when $D^2 f(x) > 0$.



Recall $m \leq Q \leq L$ means the eigenvalues of the symmetric matrix Q are between L and m. We say $f(x)$ is *strongly convex* there are positive constants m and L satisfying $m \leq D^2 f(x) \leq L$ for every x.

As in the scalar case, in order of breadth, we have

- quadratic functions, $Q = D^2 f(x)$ is constant with $Q > 0$,
- strongly convex functions, $L \geq D^2 f(x) \geq m > 0$ (eigenvalues between m and L)
- strictly convex functions, $D^2 f(x) > 0$ (positive eigenvalues),
- general convex functions, $D^2 f(x) \geq 0$ (nonnegative eigenvalues).



Let $f(x)$ be strongly convex with bounds $m \leq L$. Given two points a and x, let $v = x - a$ and $g(t) = f(x + tv)$. By (4.3.12), $g(t)$ is a strongly convex function of the scalar variable t with bounds $m|x-a|^2 \leq L|x-a|^2$. Moreover

$$g(1) = f(x), \quad g(0) = f(a), \quad g'(0) = \nabla f(a)\cdot(x-a), \quad g'(1) = \nabla f(x)\cdot(x-a).$$

Applying the scalar strongly convex results in §4.1 to $g(t)$ and the points $t = 0$, $t = 1$, we obtain the following vector versions of those results.

Lower and Upper Tangent Paraboloids

Let $f(x)$ be strongly convex, with $m \leq D^2 f(x) \leq L$. Then

$$\frac{m}{2}|x-a|^2 \leq f(x) - f(a) - \nabla f(a)\cdot(x-a) \leq \frac{L}{2}|x-a|^2. \quad (4.5.15)$$

Strongly Convex Implies Unique Global Minimizer

A strongly convex function has exactly one global minimizer x^*.

Strongly Convex Gradient Bounds

Let $f(x)$ be strongly convex. If $q = \nabla(x)$ and $p = \nabla f(a)$, then

$$m|x-a|^2 \leq (q-p)\cdot(x-a) \leq L|x-a|^2. \quad (4.5.16)$$

Coercivity of the Gradient

Let $f(x)$ be strongly convex. If $q = \nabla f(x)$ and $p = \nabla f(a)$, then

$$\frac{mL}{m+L}|x-a|^2 + \frac{1}{m+L}|q-p|^2 \le (q-p)\cdot(x-a). \qquad (4.5.17)$$

If $a = x^*$ is a global minimizer, then $\nabla f(x^*) = 0$, and $f(x)$ lies between two quadratics globally,

$$\frac{m}{2}|x-x^*|^2 \le f(x) - f(x^*) \le \frac{L}{2}|x-x^*|^2. \qquad (4.5.18)$$

These results are used in gradient descent §7.8.



We describe the convex dual in the multi-variable setting (the single-variable case was done in §4.1). If $f(x)$ is a scalar convex function of x, and x has d features, the *convex dual* is

$$g(p) = \max_x \left(p\cdot x - f(x)\right). \qquad (4.5.19)$$

Here the maximum is over all vectors x, and $p = (p_1, p_2, \ldots, p_d)$, the *dual variable*, also has d features.

As in the scalar case, when $f(x)$ is strongly convex, the maximizer in (4.5.19) is well-defined, and we have

Dual of the Dual

Let $f(x)$ be strongly convex. If $g(p)$ is the convex dual of $f(x)$, then $f(x)$ is the convex dual of $g(p)$, $g(p)$ is strongly convex, and

$$D^2 g(p) = D^2 f(x)^{-1} \quad \Longleftrightarrow \quad p = \nabla f(x) \quad \Longleftrightarrow \quad x = \nabla g(p).$$

It follows that the convex dual $g(p)$ is strongly convex with

$$\frac{1}{L} \le D^2 g(p) \le \frac{1}{m}.$$

Using this, we also have (Exercise **4.1.20**)

Dual Second Derivative Bounds

Let $f(x)$ be strongly convex. If $p = \nabla f(x)$ and $q = \nabla f(a)$, then

$$\frac{1}{2L}|p-q|^2 \le f(x) - f(a) - \nabla f(a)\cdot(x-a) \le \frac{1}{2m}|p-q|^2. \quad (4.5.20)$$

Combining (4.5.15) and (4.5.20), we have

Lipschitz Bounds

Let $f(x)$ be strongly convex. If $p = \nabla f(x)$ and $q = \nabla f(a)$, then

$$m|x-a| \le |p-q| \le L|x-a|. \quad (4.5.21)$$

These are used in gradient descent §7.8.



In §4.3, we showed how the existence of minimizers follows from properness. Now we establish a converse, needed in §7.6.

To set the stage, let $f(x)$ be a strictly convex function on $\mathbf{R}^d$ with a minimizer x^*, and assume the gradient $\nabla f(x)$ is continuous.

Fix another point x, and let $g(t)$ be the restriction of $f(x)$ to the line passing through x^* and x,

$$g(t) = f(x^* + t(x - x^*)).$$

Then $g(t)$ is a strictly convex function of a single variable t, so we may apply the results of §4.1.

Suppose x is also a minimizer. Since $g(0) = f(x^*)$ and $g(1) = f(x)$, both $t = 0$ and $t = 1$ are minimizers of $g(t)$. Since a strictly convex function of a single variable has at most one minimizer, there can be no such x. Summarizing,

Strictly Convex Has at Most One Minimizer

Let $f(x)$ be strictly convex on $\mathbf{R}^d$. Then $f(x)$ has at most one minimizer.

If we set $v = x - x^*$, then $g(t) = f(x^* + tv)$. By the chain rule,

$$g'(t) = \nabla f(x^* + tv)\cdot v.$$

Since x^* is a minimizer, from §4.3, x^* is a critical point, $\nabla f(x^*) = 0$, hence $g'(0) = 0$. By strict convexity, $g'(t)$ is strictly increasing, hence $g'(1) > 0$. This implies $g'(1) = \nabla f(x^* + v)\cdot v > 0$, for every unit vector v.

Since $v = x - x^*$, the unit sphere S centered at x^* is the set of x satisfying $|v| = 1$. Let ϵ be the minimum of $\nabla f(x^* + v)\cdot v$ over S,

$$\epsilon = \min_{|v|=1} g'(1) = \min_{|v|=1} \nabla f(x^* + v) \cdot v.$$

Since S is a closed and bounded set, and $g'(1)$ depends continuously on v, there is a minimizer in S (§A.8), hence ϵ is positive.

From §4.1, a convex function lies above its tangent line, so

$$g(t) \geq g(1) + g'(1)(t-1).$$

Since $g'(1) \geq \epsilon$ and $g(1) = f(x^* + v) \geq f(x^*)$ (x^* is a minimizer), this implies

$$f(x^* + tv) \geq f(x^*) + \epsilon(t-1).$$

Let x be any point not equal to x^*, and let $t = |x - x^*|$, $v = (x - x^*)/|x - x^*|$. Then $x^* + tv = x$. Inserting these values,

$$f(x) \geq f(x^*) + \epsilon(|x - x^*| - 1), \qquad x \neq x^*.$$

Then $f(x) \leq c$ implies

$$|x - x^*| \leq \frac{c - f(x^*)}{\epsilon} + 1. \tag{4.5.22}$$

Let c be any level greater than the minimum, $c \geq f(x^*)$, and let

$$C = 1 + |x^*| + \frac{c - f(x^*)}{\epsilon}.$$

Since $|x| \leq |x - x^*| + |x^*|$, from (4.5.22), we obtain

$$f(x) \leq c \qquad \text{implies} \qquad |x| \leq C.$$

By definition of proper (4.3.7), we have derived

Strictly Convex and Has Minimizer Implies Proper

Let $f(x)$ be strictly convex on $\mathbf{R}^d$, and suppose $f(x)$ has a minimizer. If the gradient $\nabla f(x)$ is continuous, then $f(x)$ is proper.

Exercises

Exercise 4.5.1 If a two-class dataset does not lie in a hyperplane and is separable, then the means of the two classes are distinct. (Argue by contradiction: Assume the means are the same, and look at levels of samples.)

Exercise 4.5.2 Suppose (m_1, b_1) and (m_2, b_2) result in the same hyperplane. Show there is a nonzero scalar t with $(m_2, b_2) = (tm_1, tb_1)$ (Exercise **2.7.10**).

Exercise 4.5.3 Let $e_0 = 0$ and let e_1, e_2, …, e_d be the one-hot encoded basis in $\mathbf{R}^d$. The *d-simplex* Σ_d is the convex hull of e_0, e_1, e_2, …, e_d. Draw pictures of Σ_1, Σ_2, and Σ_3. Show Σ_d is the suspension (§1.5) of Σ_{d-1} from e_d. Conclude

$$\mathrm{Vol}(\Sigma_d) = \frac{1}{d!}, \qquad d = 0, 1, 2, 3, \ldots$$

(Since Σ_0 is one point, we start with $\mathrm{Vol}(\Sigma_0) = 1$.)

Exercise 4.5.4 Let $I(p, q)$ be the relative information (4.2.9). With $x = (p, q)$ and $v = (ap(1-p), bq(1-q))$, show

$$\left.\frac{d^2}{dt^2}\right|_{t=0} I(x + tv) = p(1-p)(a-b)^2 + b^2(p-q)^2.$$

Conclude that $I(p, q)$ is a convex function of (p, q). Where is it not strictly convex?

Exercise 4.5.5 If $Q > 0$ be a positive symmetric matrix, then

$$\frac{1}{2}(p - Qx) \cdot Q^{-1}(p - Qx) = \frac{1}{2}p \cdot Q^{-1}p - p \cdot x + \frac{1}{2}x \cdot Qx. \tag{4.5.23}$$

Conclude the convex dual of $f(x) = x \cdot Qx/2$ is $g(p) = p \cdot Q^{-1}p/2$.

Exercise 4.5.6 Let $0 < \lambda_1 < \lambda_2 < \lambda_3 < \lambda_4$ be positive scalars, and let $P_k = (\lambda_k, 1/\lambda_k)$, $k = 1, 2, 3, 4$, be the four green points on the graph of $y = 1/x$ in Figure 4.32. If

$$(\alpha, \beta) = t_1P_1 + t_2P_2 + t_3P_3 + t_4P_4$$

is a convex combination of the four points, then (α, β) *is located in the blue region* in Figure 4.32. Why? Conclude $1 \le \alpha\beta \le \alpha\gamma$.

Exercise 4.5.7 With $m = \lambda_1$ and $L = \lambda_4$ and γ as in Figure 4.32, show

$$\gamma = \frac{m + L - \alpha}{mL} \qquad \text{and} \qquad 1 \le \alpha\beta \le \alpha\gamma \le \frac{(m+L)^2}{4mL}.$$

(maximize $f(\alpha) = \alpha\gamma$ over $m < \alpha < L$).

Exercise 4.5.8 Use the previous two exercises to obtain

$$1 \le \left(t_1\lambda_1 + t_2\lambda_2 + t_3\lambda_3 + t_4\lambda_4\right) \times \left(\frac{t_1}{\lambda_1} + \frac{t_2}{\lambda_2} + \frac{t_3}{\lambda_3} + \frac{t_4}{\lambda_4}\right) \le \frac{(m+L)^2}{4mL}.$$

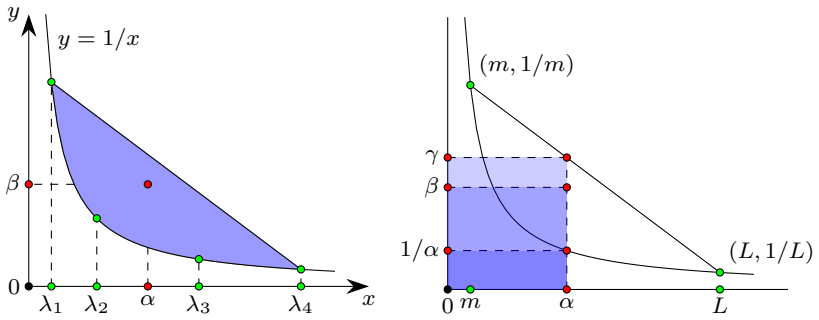


Fig. 4.32 $y = 1/x$ is a convex function.

Exercise 4.5.9 Let Q be a nonzero symmetric matrix with nonnegative eigenvalues, and let L and m be the greatest and least *positive* eigenvalues of Q. If Q^+ is the pseudo-inverse, show

$$1 \le (u \cdot Qu)(u \cdot Q^+ u) \le \frac{(m+L)^2}{4mL} \tag{4.5.24}$$

for every unit vector u in the row space of Q. Use EVD and Exercise **4.5.8**. See Exercise **4.1.19** for the 2×2 case.

Exercise 4.5.10 Let A be any nonzero matrix and let σ_1 and σ_2 be the greatest and least *nonzero* singular values of A. If A^+ is the pseudo-inverse, show

$$1 \le |Au| \times \left|(A^+)^t u\right| \le \frac{1}{2}\left(\frac{\sigma_1}{\sigma_2} + \frac{\sigma_2}{\sigma_1}\right)$$

for every unit vector u in the row space of A. Use Exercises **4.5.9** and **2.6.14**.

Exercise 4.5.11 Let $x_1, x_2, \ldots, x_d$ be positive scalars. Use convexity of $x = e^t$ to show

$$\frac{1}{d}\sum_{i=1}^{d} x_i \ge (x_1 x_2 \ldots x_d)^{1/d}.$$

Exercise 4.5.12 Derive (4.5.17) by applying (4.1.20) to $f(a+tv)$, $v = b-a$, $0 \le t \le 1$.

Exercise 4.5.13 By (4.5.21), strong convexity implies Lipschitz bounds. Show the converse: If

$$m|x-a| \le |\nabla f(x) - \nabla f(a)| \le L|x-a|$$

for all x, then $m \le D^2 f(a) \le L$. (Insert $x = a + tv$ with v a unit eigenvector of $D^2 f(a)$, and let $t \to 0$.)

Exercise 4.5.14 Let K be the convex hull of a dataset, and suppose the mean μ of the dataset lies on the boundary of K. Then the dataset lies in any supporting hyperplane at μ.

Exercise 4.5.15 Let K be the convex hull of a dataset, and suppose the dataset does not lie in a hyperplane. Then the mean μ of the dataset is in the interior of K.

Exercise 4.5.16 Show a hyperplane in $\mathbf{R}^d$ cannot contain a ball, hence a hyperplane has no interior.

Exercise 4.5.17 If a set K lies to one side of a hyperplane, and x_0 is a point in the interior of K, then x_0 is not in the hyperplane.

Exercise 4.5.18 Let K be the convex hull of a dataset. Then the dataset lies in a hyperplane iff K has no interior. Equivalently, the dataset does not lie in a hyperplane iff K has interior.

Exercise 4.5.19 Let K be a convex set, let x be in K, and let μ be in the interior of K. Then, apart from x, the line segment joining μ and x lies in the interior of K. (See Figure 4.30.)

Exercise 4.5.20 Let $f(Q) = \log\det(Q)$ be the log of the determinant of a positive 2×2 matrix Q (Exercise **3.2.6**), and let V be a symmetric 2×2 matrix. Using Exercise **A.4.12**, compute the second derivative of $f(Q+tV)$ at $t = 0$ as in (4.3.12). Using Exercise **A.4.10**, conclude $\log\det(Q)$ is a concave function of Q.

Exercise 4.5.21 Let $f(b)$ be strictly convex and proper, and let A be a matrix. Use the properness of the residual (4.3.8) to show $f(Ax)$ is strictly convex and proper on the row space of A. Here the shapes of x, A and b are such that $Ax = b$ is defined.

Exercise 4.5.22 Show a strongly convex function, either single-variable §4.1 or multi-variable §4.5, is proper.

Exercise 4.5.23 Let $J(M, b)$ be a function of two variables M and b. If $J(M, b)$ is convex, then $J(M, 0)$ is convex. If $J(M, b)$ is strictly convex, then $J(M, 0)$ is strictly convex.

Exercise 4.5.24 Let y be the level (4.5.3) of a point x corresponding to a hyperplane. If the levels of samples x in a dataset are below a certain level, then the levels of all x in the convex hull of the dataset are below that same level.

Chapter 5
Probability

Many concepts of probability are already present in a coin-tossing context. Because of this, we first start with a section on basic notions of probability, then we dive into coin-tossing, and discuss binomial probability. Here we show how, even in this simplest setting, entropy is an inescapable feature, a basic measure of randomness.

We also show how Bayes theorem allows us to flip things and gain inference. For this, we need the fundamental theorem of calculus §A.6.

After this, random variables and the normal and chi-squared distributions are covered. The presentation is layered so that a reader with only minimal prior exposure will come away with an appreciation of probabilistic reasoning.

5.1 Probability

Let us start with an experiment. An *experiment* is a procedure that yields an *outcome*, one of a set of possible outcomes. Each time we run the experiment, the result, possibly an aggregate of several interrelated samples, is considered a single outcome.

Tossing a coin once yields one of two outcomes, heads or tails, H or T, which we also write as 1 or 0. Rolling a six-sided die yields outcomes 1, 2, 3, 4, 5, 6. Rolling two six-sided dice yields 36 outcomes $(1,1)$, $(1,2)$,.... Tossing a coin three times yields outcomes TTT, TTH, THT, THH, HTT, HTH, HHT, HHH, which we also write as 111, 110, 101, 100, 011, 010, 000.

The *outcome space* of an experiment is the set S of all possible outcomes. If $\#(S)$ is the number of outcomes, then for the four experiments above, $\#(S)$ equals 2, 6, 36, and 8. The outcome space S is also called the *sample space*, and the *population*.

An *event* is a specific subset A of S. For example, when rolling two dice, let A_1 can be the subset of outcomes where the sum of the dice equals 7.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

O. Hijab, *Math for Data Science*, https://doi.org/10.1007/978-3-031-89707-8_5

Then the event A_1 consists of the outcomes $(1,6)$, $(2,5)$, $(3,4)$, $(4,3)$, $(5,2)$, $(6,1)$. Here $\#(S) = 36$ and $\#(A_1) = 6$.

Let A_2 be the event of obtaining three heads when tossing a coin seven times. Here $\#(S) = 2^7 = 128$ and $\#(A_2) = 35$, which is the number of ways you can choose three things out of seven things (§A.1):

$$\#(A_2) = \text{7-choose-3} = \binom{7}{3} = \frac{7 \cdot 6 \cdot 5}{1 \cdot 2 \cdot 3} = 35.$$

Suppose A is an event in an experiment, and suppose the outcome of the experiment is in A. Then we say A *has occured*. For example, if rolling two dice results in $(2,5)$, the event A_1 has occured. If tossing a coin seven times results in four heads and three tails, the event A_2 has not occured.

If an event always occurs, then it is a *certain event*. For example, when tossing a coin seven times, obtaining fewer than ten heads is a certain event. Every outcome is in a certain event. There is only one certain event, the whole outcome space. Even so, we often say "a certain event" because certainty can be presented in many ways.

If an event never occurs, it is an *impossible event*. For example, when tossing a coin seven times, obtaining eight heads is an impossible event. There are no outcomes in an impossible event. There is only one impossible event, the event with no outcomes. Even so, we often say "an impossible event" because impossibility can be presented in many ways.



The *intersection* of events A and B is the event $(A \text{ and } B) = A \cap B$ of outcomes common to both events. If A_3 is the event of obtaining at least one 5 when rolling two dice, then the outcomes in $(A_1 \text{ and } A_3)$ are $(2,5)$, $(5,2)$. Here $\#(A_3) = 11$ and $\#(A_1 \text{ and } A_3) = 2$.

Events A and B are *exclusive* if there is no outcome common to both events. Two events are exclusive if their intersection is impossible.

The *union* of events A and B is the event $(A \text{ or } B) = A \cup B$ of outcomes that are in A or in B. If A_3 is the event of obtaining at least one 5 when rolling two dice, then check that $\#(A_1 \text{ or } A_3) = 15$.

Events A and B are *exhaustive* if every outcome is either in A or in B. Two events are exhaustive if their union is certain.

If A is an event, the *complementary event* is the event A^c of outcomes not in A. Thus A occurs exactly when A^c does not occur: the events A and A^c are exclusive and exhaustive.

The *difference* of A minus B are the outcomes in A but not in B: The outcomes of $A_1 - A_3$ are $(1,6)$, $(3,4)$, $(4,3)$, $(6,1)$. Similarly, the outcomes of $A_3 - A_1$ are $(5,1)$, $(5,3)$, $(5,4)$, $(5,5)$, $(5,6)$, $(1,5)$, $(3,5)$, $(4,5)$, $(5,5)$, $(6,5)$.

An event A is *part* of event B if every outcome in A is also in B. If A is part of B, then $A \cap B$ is A, $A \cup B$ is B, and $A - B$ is impossible.

Moreover, for any A, B, $A - B$ is part of A and $B - A$ is part of B, $A \cap B$ is part of both, and all are part of $A \cup B$.

These concepts carry over to several events. Events A, B, C are *exclusive* if $A \cap B$, $A \cap C$, and $B \cap C$ are impossible. Events A, B, C are *exhaustive* if $A \cup B \cup C$ is certain.



A *probability* on an outcome space S is an assignment of a number $Prob(A)$ to every event A in S, satisfying three axioms.

Axioms for Probability

A probability on S satisfies

1. $0 \leq Prob(A) \leq 1$ for every event A in S,
2. $Prob(S) = 1$,
3. (Additivity) If A, B, ... are exclusive events in S,

$$Prob(A \cup B \cup \dots) = Prob(A) + Prob(B) + \dots. \tag{5.1.1}$$

Suppose the number of outcomes $\#(S)$ is finite. Then the simplest probability is the *discrete uniform distribution*, assigning to each event A the *proportion of outcomes in* A,

$$Prob(A) = \frac{\#(A)}{\#(S)}. \tag{5.1.2}$$

When this is so and $\#(S) = N$, each outcome has probability $1/N$, and we say the outcomes are *equally likely*.

Here are examples of discrete uniform distributions.

1. A coin is *fair* if, after one toss, the two outcomes are equally likely. Then $Prob(heads) = Prob(tails) = 1/2$.
2. A 6-sided die is *fair* if, after one roll, the outcomes are equally likely. Let A be the event that the outcome is less than 3. Since the outcome is then 1 or 2, $Prob(A) = 2/6 = 1/3$.
3. With A_1 as above, assuming the dice are fair, leads to $Prob(A_1) = 6/36 = 1/6$.
4. With A_2 as above, assuming the coin is fair, leads to $Prob(A_2) = 35/128$.
5. With A_3 as above, assuming the dice are fair, leads to $Prob(A_3) = 11/36$.



Here are some consequences of the probability axioms. Since A and A^c are exclusive and exhaustive, the first consequence is

Complementary Probabilities

For any event A,

$$Prob(A^c) = 1 - Prob(A).$$

An event A is *sure* if $Prob(A) = 1$. An event A is *null* if $Prob(A) = 0$. Then A is sure iff A^c is null. A certain event is sure, and an impossible event is null. However, as we see below, *sure* and *certain* are not the same, nor are *null* and *impossible*.

The second consequence of additivity is

Monotonicity of Probabilities

Let A be an event, and let B be part of A. Then

$$Prob(B) \leq Prob(A). \tag{5.1.3}$$

This follows from Exercise **5.1.7**. A useful variation of additivity (5.1.1) is

Additivity of Probabilities

If A_1, A_2, ... are exclusive and exhaustive events, and B is any event, then

$$Prob(B) = Prob(B \text{ and } A_1) + Prob(B \text{ and } A_2) + \dots. \tag{5.1.4}$$

In particular, if A and B are any events,

$$Prob(B) = Prob(B \text{ and } A) + Prob(B \text{ and } A^c). \tag{5.1.5}$$

Also, when the events are not exclusive, we have

Sub-Additivity of Probabilities

If the event A is part of the union of events A_1, A_2, ..., then

$$Prob(A) \leq Prob(A_1) + Prob(A_2) + \dots. \tag{5.1.6}$$

This follows from additivity (Exercise **5.1.10**).



Suppose we sample numbers X at random from the interval $[0, 1]$ in a uniform manner. This means the outcome space is $S = [0, 1]$, and the probability of sampling from a sub-interval $[a, b]$ equals its length,

$$Prob(a < X < b) = b - a, \qquad 0 \le a < b \le 1.$$

Since a single number a is a sub-interval $[a, a]$ with zero length, the event A of sampling X exactly equal to 0.5 is a null event. Since A is possible, A is not impossible. Moreover A^c is a sure event, but A^c is not certain.

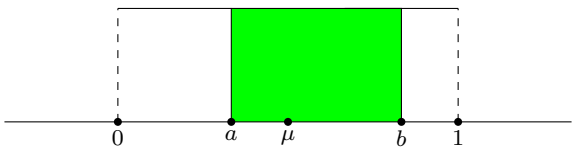


Fig. 5.1 Uniform probability density function.



Here is a more sophisticated example of a null event that is not impossible. Because this example relates to the LLN (see below and Exercise **5.1.11**), we go over it carefully.

Toss a fair coin n times. Then the outcome space S_n consists of all *n-tuples* $x = (t_1, t_2, \dots, t_n)$ of 0's and 1's. Here the number of outcomes is 2^n, and the probability of each outcome is 2^{-n}.

Now toss a coin *infinitely* many times. Then the outcome space S_∞ consists of all *infinite tuples* $x = (t_1, t_2, \dots)$ of 0's and 1's. Here the number of tosses in each outcome is infinite, and it turns out the probability of each outcome is zero.

Why is the probability of each outcome x in S_∞ zero? To understand why, we first have to discuss how to measure probabilities of events in S_∞.

If an event A in S_∞ depends only on the first n tosses, this is easy. We consider A as part of S_n, and $Prob(A)$ is taken from the discrete uniform distribution (5.1.2) on S_n. For example, if A consists of all outcomes in S_∞ with 3 heads in the first 7 tosses, then $Prob(A) = 35/128$. Events that depends only on finitely many tosses are called *finite-toss* events.

Of course, the interesting events in S_∞ are those that are *not* finite-toss events, see the LLN below. Even so, it turns out there is one and only one way to specify probabilities $Prob$ on all events in S_∞, finite-toss or otherwise, once $Prob$ is specified on finite-toss events as above.

Now let $x = (t_1, t_2, \dots)$ be a specific outcome, and let A_n be the event of all outcomes in S_∞ whose first n tosses are exactly $(t_1, t_2, \dots, t_n)$. Since A_n depends only on the first n tosses, $Prob(A_n) = 2^{-n}$. Since x is in A_n for every $n \ge 1$, by monotonicity (5.1.3), $Prob(x) \le 2^{-n}$ for every $n \ge 1$. In this inequality, let n increase without bound. Since the right side approaches zero, we obtain $Prob(x) = 0$. This shows each outcome has probability zero.

Now, for $k < n$, let $A_{n,k}$ be the event of outcomes in S_∞ with k heads in the first n tosses. Since there are $\binom{n}{k}$ ways of obtaining k heads in n tosses (§A.1),

$$Prob\,(A_{n,k}) = 2^{-n}\binom{n}{k}.$$

Let $A_{\infty,k}$ be the event of outcomes with exactly k heads in infinitely many tosses. Then each outcome x in $A_{\infty,k}$ is in $A_{n,k}$ for *some* n. This does not mean $A_{\infty,k}$ is part of $A_{n,k}$, because here n depends on the outcome x, and the tosses beyond the first n tosses are not specified in $A_{n,k}$.

However, we can say each outcome x in $A_{\infty,k}$ is necessarily in $A_{n,k}$ for all sufficiently large n. This means the following: if x is in $A_{\infty,k}$, then for *some* $N \geq 1$, x is in $A_{n,k}$ for *every* n greater or equal than N. Therefore, for each outcome x in $A_{\infty,k}$, there is some N, *depending on* x, with x in

$$\bigcap_{n\geq N} A_{n,k} = A_{N,k} \cap A_{N+1,k} \cap A_{N+2,k} \cap \ldots.$$

This shows the event $A_{\infty,k}$ is part of the union of the events $\bigcap_{n\geq N} A_{n,k}$ over $N = 1, 2, \ldots$.

Since $\bigcap_{n\geq N} A_{n,k}$ is part of $A_{m,k}$ for every $m \geq N$, by monotonicity (5.1.3),

$$Prob\left(\bigcap_{n\geq N} A_{n,k}\right) \leq 2^{-m}\binom{m}{k}, \qquad \text{for every } m \geq N.$$

With k fixed, let m increase without bound in this inequality. Since the right side approaches zero (Exercise **A.7.1**), $\bigcap_{n\geq N} A_{n,k}$ is a null event.

Since $A_{\infty,k}$ is part of the union of $\bigcap_{n\geq N} A_{n,k}$ over $N \geq 1$, by subadditivity (5.1.6), $A_{\infty,k}$ is also a null event. However, since there definitely are outcomes in S_∞ with k heads, $A_{\infty,k}$ is not impossible.

Finally, let A_∞ be the event of all outcomes with a finite, but unspecified, number of heads. Then A_∞ equals

$$A_{\infty,0} \cup A_{\infty,1} \cup A_{\infty,2} \cup \ldots.$$

By subadditivity again, A_∞ is a null event.

The moral here is *if you try to specify a pattern of infinitely many tosses of a coin, you end up with a null event.* This is why coin-tossing is called random.



The *conditional probability* of A *given* B is

$$Prob(A \mid B) = \frac{Prob(A \text{ and } B)}{Prob(B)}. \tag{5.1.7}$$

The definition (5.1.7) is equivalent to the *chain rule*

$$Prob(A \text{ and } B) = Prob(A \mid B)\, Prob(B). \tag{5.1.8}$$

Events A and B are *independent* if

$$Prob(A \text{ and } B) = Prob(A) \times Prob(B). \tag{5.1.9}$$

When A and B are independent,

$$Prob(A \mid B) = \frac{Prob(A \text{ and } B)}{Prob(B)} = \frac{Prob(A) \times Prob(B)}{Prob(B)} = Prob(A).$$

When A and B are independent, the conditional probability equals the unconditional probability.

Are A_1 and A_3 above independent? Since $Prob(A_1) = 6/36 = 1/6$ and

$$Prob(A_1 \mid A_3) = \frac{Prob(A_1 \text{ and } A_3)}{Prob(A_3)} = \frac{2/36}{11/36} = \frac{2}{11},$$

A_1 and A_3 are not independent.

Additivity of probabilities and the chain rule can be combined into

Law of Total Probability

If $A_1, A_2, \ldots$ are exclusive and exhaustive events, and B is any event, then

$$Prob(B) = Prob(B \mid A_1)\, Prob(A_1) + Prob(B \mid A_2)\, Prob(A_2) + \ldots. \tag{5.1.10}$$

In particular, if A and B are any events,

$$Prob(B) = Prob(B \mid A)\, Prob(A) + Prob(B \mid A^c)\, Prob(A^c). \tag{5.1.11}$$



Suppose in a certain community 15% of families have no children, 20% have one child, 35% have two children, and 30% have three children. Suppose also each child is equally likely to be a boy or a girl. Let B and G be the number of boys and girls in a randomly selected family. Then

$$Prob(B = 0 \text{ and } G = 0) = Prob(\text{no children}) = 0.15,$$

and

$$Prob(B = 0 \text{ and } G = 1) = Prob(G = 1 \mid 1 \text{ child})\, Prob(1 \text{ child}) = 0.1,$$

and

$$Prob(B = 1 \text{ and } G = 2) = Prob(G = 2 \mid 3 \text{ children})\, Prob(3 \text{ children})$$

which equals .1125. Continuing in this manner, the complete table is

$Prob((B, G) = (i, j))$	$G = 0$	$G = 1$	$G = 2$	$G = 3$
$B = 0$	0.15	0.10	.0875	.0375
$B = 1$	0.10	.175	.1125	0
$B = 2$	.0875	.1125	0	0
$B = 3$	.0375	0	0	0

Table 5.1 Joint distribution of boys and girls [30].



Now suppose we conduct an experiment by tossing a coin (always assumed fair unless otherwise mentioned) 10 times. Because the coin is fair, we expect to obtain heads around 5 times. Will we obtain heads exactly 5 times? Let's run the experiment with Python. In fact, we will run the experiment 20 times. If we count the number of heads after each run of the experiment, we obtain a digit between 0 and 10 inclusive.

To simulate this, we use `binomial(n,p,N)`. When $N = 1$, this returns the number of heads obtained after a single experiment, consisting of tossing a coin n times, where the probability of obtaining heads in each toss is p.

More generally, `binomial(n,p,N)` runs this experiment N times, returning a vector v with N components. For example, the code

```
from numpy import *
set_printoptions(legacy = "1.25")

from numpy.random import default_rng
samples = default_rng().binomial

p, n, N = .5, 10, 20

v = samples(n,p,N)
print(v)
```

returns

$$[9, 6, 7, 4, 4, 4, 3, 3, 7, 5, 6, 4, 6, 9, 4, 5, 4, 7, 6, 7].$$

The outcome space S corresponding to (p, n, N) consists of all vectors $v = (v_1, v_2, \ldots, v_N)$ with N components, with each component equal to to 0, 1, $\ldots$, n. So here $\#(S) = (n + 1)^N$.

Now we conduct three experiments: tossing a coin 5 times, then 50 times, then 500 times. The code

```
p = .5
for n in [5,50,500]: print(samples(n,p,1))
```

This returns the count of heads after 5 tosses, 50 tosses, and 500 tosses,

$$3, 28, 266.$$

The *proportions* are the count divided by the total number of tosses in the experiment. For the above three experiments, the proportions after 5 tosses, 50 tosses, and 500 tosses, are

$$3/5 = .600, 28/50 = .560, 266/500 = .532.$$

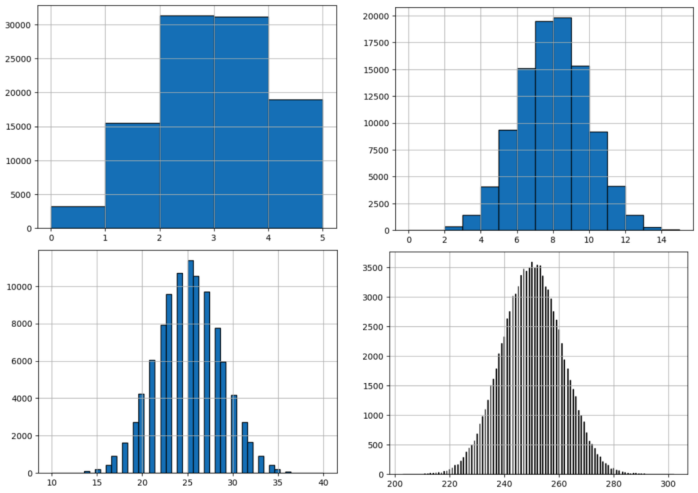


Fig. 5.2 100,000 sessions, with 5, 15, 50, and 500 tosses per session.

Now we repeat each experiment 100,000 times and we plot the results in a histogram.

```
from matplotlib.pyplot import *
from numpy.random import default_rng
samples = default_rng().binomial

N, p = 100000, .5

for n in [5,50,500]:
```

```
data = samples(n,p,N)
hist(data,bins = n,edgecolor = 'Black')
grid()
show()
```

This results in Figure 5.2.

The takeaway from these graphs are the two fundamental results of probability:

Law of Large Numbers (LLN)

The proportion in a repeated experiment is the *sample proportion*. The sample proportion tends to be near the underlying probability p. The underlying probability is the *population proportion*. The larger the sample size in the experiment, the closer the proportion is to p. Another way of saying this is: For large sample size, the sample mean is approximately equal to the population mean.

Central Limit Theorem (CLT)

For large sample size, the shape of the graph of the proportions or counts is approximately normal. The normal distribution is studied in §5.4. Another way of saying this is: For large sample size, the shape of the sample mean histogram is approximately normal.

The law of large numbers is *qualitative* and the central limit theorem is *quantitative*. While the law of large numbers says one thing is close to another, it does not say how close. The central limit theorem provides a numerical measure of closeness, using the normal distribution.

One may think that the LLN and the CLT above depend on some aspect of the binomial distribution. After all, the binomial is a specific formula and something about this formula may lead to the LLN and the CLT. To show that this is not at all the case, to show that the LLN and the CLT are universal, we bring in the petal lengths of the Iris dataset. This time the experiment is not something we invent, it is a result of something arising in nature, Iris petal lengths.

We begin by loading the Iris dataset,

```
from sklearn import datasets

iris = datasets.load_iris()

dataset = iris["data"]
iris["feature_names"]
```

This code shows the petal lengths are the third feature in the dataset, and we compute the mean of the petal lengths using

```
petal_lengths = dataset[:,2]
mean(petal_lengths)
```

This returns the petal length *population mean* $\mu = 3.758$. If we plot the petal lengths in a histogram with 50 bins using the code

```
from matplotlib.pyplot import *

grid()
hist(petal_lengths,bins = 50)
show()
```

we obtain Figure 5.3.

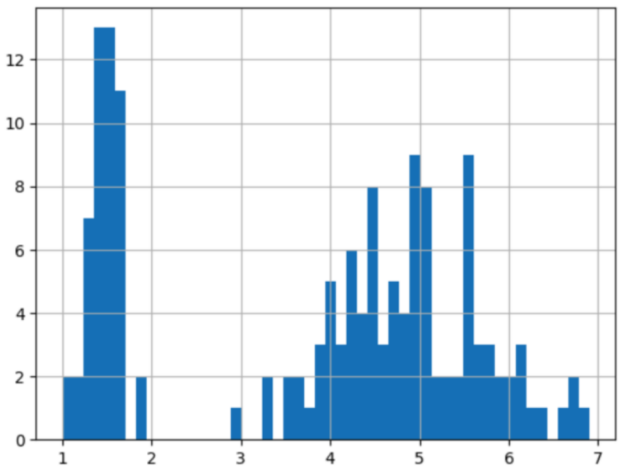


Fig. 5.3 The histogram of Iris petal lengths.

Now we sample the Iris dataset randomly. More generally, we take a random *batch* of samples of size n and take the mean of the samples in the batch. For example, the following code grabs a batch of $n = 5$ petals lengths X_1,

X_2, X_3, X_4, X_5 at random and takes their mean,

$$\frac{X_1 + X_2 + X_3 + X_4 + X_5}{5}.$$

The code is

```
from numpy import *

from numpy.random import default_rng
rng = default_rng()

# n = batch_size

def random_batch_mean(n):
   rng.shuffle(petal_lengths)
   return mean(petal_lengths[:n])

random_batch_mean(5)
```

This code shuffles the dataset, then selects the first n petal lengths, then returns their mean.

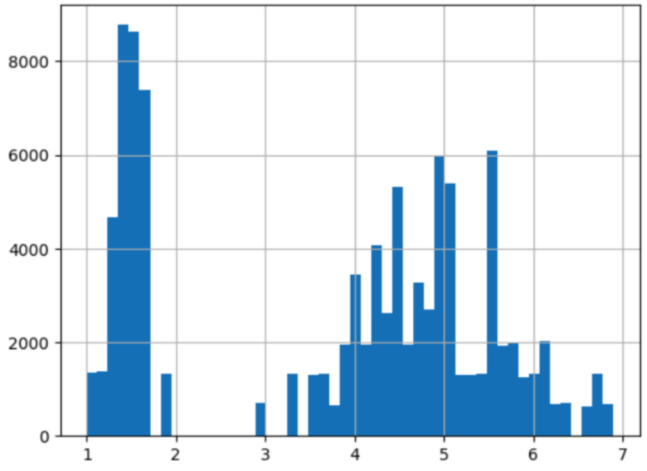


Fig. 5.4 Iris petal lengths sampled 100,000 times.

To sample a single petal length randomly 100,000 times, we run the code

```
N = 100000
n = 1
```

```
Xbar = [ random_batch_mean(n) for _ in range(N)]
hist(Xbar,bins = 50)
grid()
show()
```

Since we are sampling single petal lengths, here we take $n = 1$. This code returns the histogram in Figure 5.4.

In Figure 5.3, the bin heights add up to 150. In Figure 5.4, the bin heights add up to 100,000. Moreover, while the shapes of the histograms are almost identical, a careful examination shows the histograms are not identical. Nevertheless, there is no essential difference between the two figures.

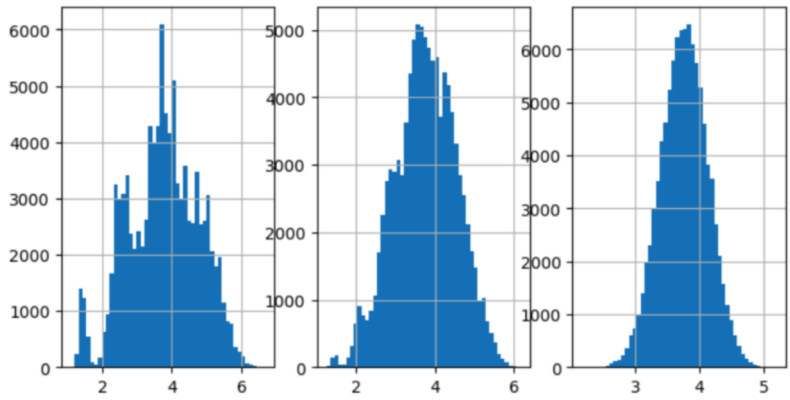


Fig. 5.5 Iris petal lengths batch means sampled 100,000 times, batch sizes 3, 5, 20.

Now repeat the same experiment, but with batches of various sizes, and plot the resulting histograms. If we do this with batches of size $n = 3$, $n = 5$, $n = 20$ using

```
from matplotlib.pyplot import *

figure(figsize=(8,4))
# three subplots
rows, cols = 1, 3

N = 100000

for i,n in enumerate([3,5,20],start=1):
   Xbar = [ random_batch_mean(n) for _ in range(N)]
   subplot(rows,cols,i)
   grid()
   hist(Xbar,bins = 50)
```

```
show()
```

we obtain Figure 5.5.

This shows the CLT is universal, since here it arises from sampling the petal lengths of Irises, whose dataset has the histogram in Figure 5.3. Of course, we also have the LLN, which says the peak of each of the bell-shaped curves is near $\mu = 3.758$.

Exercises

Exercise 5.1.1 [30] A communications channel transmits bits 0 and 1. Because of noise, the probability of transmitting a bit incorrectly is 0.2. To reduce error probabilities, each bit is repeated five times: 1 is sent as 11111 and 0 is sent as 00000. If the recipient uses majority decoding, what is the probability of mis-reading a message consisting of one bit? Majority decoding means five consecutive bits will be read as 1 if at least three of the bits are 1, and similarly for 0.

Exercise 5.1.2 Check the values in Table 5.1.

Exercise 5.1.3 A fair coin is tossed infinitely many times. Show that the event of outcomes with both infinitely many heads and infinitely many tails is sure.

Exercise 5.1.4 [30] Approximately 80,000 marriages took place in New York last year. Assuming any day is equally likely, what is the probability that for at least one of these couples, both partners were born on January 1? Both partners celebrate their birthdays on the same day of the year?

Exercise 5.1.5 This problem has nothing to do with calculus or probability or data science, and just uses addition of numbers, so can be presented to grade school students. Let `dataset` be any five numbers, for example `[-11.2,sqrt(2),1.4,11, 23.4]`, and run the code

```
from matplotlib.pyplot import *
from numpy import *

def sums(dataset,k):
   if k == 1: return dataset
   else:
      s = sums(dataset,k-1)
      return array([ a+b for a in dataset for b in s ])

dataset = [-11.2,sqrt(2),1.4,11, 23.4]
```

```
for k in range(1,10):
   s = sums(dataset,k)
   grid()
   hist(s,bins = 50,edgecolor = "k")
   show()
```

for $k = 1, 2, 3, 4, \ldots$. What does this code do? What does it return? What pattern do you see? What if `dataset` were changed? What if the samples in the dataset were vectors?

Exercise 5.1.6 [30] At least one-half of an airplane's engines are required to function in order for it to operate. If each engine functions independently with probability p, for what value of $0 < p < 1$ is a 4-engine plane as likely to fail as a 2-engine plane?

Exercise 5.1.7 Let A and B be any events, not necessarily exclusive. Show

$$Prob(A \text{ or } B) = Prob(B) + Prob(A - B).$$

Exercise 5.1.8 Let A and B be any events, not necessarily exclusive. Extend (5.1.1) to show

$$Prob(A \text{ or } B) = Prob(A) + Prob(B) - Prob(A \text{ and } B). \tag{5.1.12}$$

(Break $A \cup B$ into three exclusive events $A - B$, $A \cap B$, and $B - A$.)

Exercise 5.1.9 [30] There is a 60% chance an event A will occur. If A does not occur, there is a 10% chance B occurs. What is the chance A or B occurs?

Exercise 5.1.10 Let A, B, C be any events, not necessarily exclusive. Use Exercise **5.1.8** to show

$$Prob(A \text{ or } B \text{ or } C) \leq Prob(A) + Prob(B) + Prob(C).$$

(Start with two events, then go from two to three events.) With $a = Prob(A^c)$, $b = Prob(B^c)$, $c = Prob(C^c)$, this exercise is the same as Exercise **A.3.4**.

Exercise 5.1.11 Toss a coin infinitely many times, and let A_1 be the outcomes $x = (x_1, x_2, \ldots)$ where the limit of the sample means

$$\lim_{n\to\infty} \frac{x_1 + x_2 + \cdots + x_n}{n}$$

equals 1. Show the event A_1 is not certain nor impossible. Here each x_k is 1 or 0. More generally, let $t = a/b$ be any fraction, and let A_t be the event of outcomes where the limit of the sample means equals t. Show A_t is not certain nor impossible.

Exercise 5.1.12 If A is any event in S_∞, let $\bar{A}$ be the event A with heads and tails interchanged. This means x_k is replaced by $1 - x_k$ in the definition of A. Then $Prob(A) = Prob(\bar{A})$. We call A *symmetric* if $\bar{A} = A$. Show for A symmetric,

$$Prob(A \text{ and } x_1 = 1) = \frac{1}{2}\, Prob(A) = Prob(A \text{ and } x_1 = 0).$$

Conclude a symmetric event A is independent of the event $x_1 = 1$. (Use additivity.)

Exercise 5.1.13 Let A_t be the event in Exercise 5.1.11. Show $\overline{A_t} = A_{1-t}$. Conclude $A_{0.5}$ is symmetric.

5.2 Binomial Probability

Suppose a coin is tossed repeatedly, landing heads or tails each time. After tossing the coin 100 times, we obtain 53 heads. What can we say about this coin? Can we claim the coin is fair? Can we claim the probability of obtaining heads is .53?

Whatever claims we make about the coin, they should be *reliable*, in that they should more or less hold up to *repeated* verification.

To obtain reliable claims, we therefore repeat the above experiment 20 times, obtaining for example the following count of heads

$$[57, 49, 55, 44, 55, 50, 49, 50, 53, 49, 53, 50, 51, 53, 53, 54, 48, 51, 50, 53].$$

On the other hand, suppose someone else repeats the same experiment 20 times with a different coin, and obtains

$$[69, 70, 79, 74, 63, 70, 68, 71, 71, 73, 65, 63, 68, 71, 71, 64, 73, 70, 78, 67].$$

In this case, one suspects the two coins are statistically distinct, and have different probabilities of obtaining heads.

In this section, we study how the probabilities of coin-tossing behave, with the goal of answering the question: *Is a given coin fair?*



Assume we are tossing a coin. If we let $p = Prob(H)$ and $q = Prob(T)$ be the probabilities of obtaining heads and tails in a single toss, then

$$p + q = 1.$$

The proportion p is the coin's *bias*. In particular, we see $q = 1 - p$, and the bias p may be any proportion between 0 and 1, depending on the particular

coin being tossed. When $p = 1/2$, $Prob(H) = Prob(T)$, we say the coin is *fair*.

If we toss the coin twice, we obtain one of four possibilities, HH, HT, TH, or TT. If we make the natural assumption that the coin has no memory, that the result of the first toss has no bearing on the result of the second toss, then the probabilities are

$$Prob(HH) = p^2, Prob(HT) = pq, Prob(TH) = qp, Prob(TT) = q^2. \quad (5.2.1)$$

These are valid probabilities since their sum equals 1,

$$p^2 + pq + qp + q^2 = (p + q)^2 = 1^2 = 1.$$

We use (5.1.7) to compute the probability that we obtain heads on the second toss given that we obtain tails on the first toss. Introduce the convenient notation

$$X_n = \begin{cases} 1, & \text{if the } n\text{-th toss is heads,} \\ 0, & \text{if the } n\text{-th toss is tails.} \end{cases}$$

Then X_n is a random variable (§5.3) and represents a numerical reward function of the outcome (heads or tails) at the n-th toss.

With this notation, (5.2.1) may be rewritten

$$\begin{aligned} Prob(X_1 = 1 \text{ and } X_2 = 1) &= p^2, \\ Prob(X_1 = 1 \text{ and } X_2 = 0) &= pq, \\ Prob(X_1 = 0 \text{ and } X_2 = 1) &= qp, \\ Prob(X_1 = 0 \text{ and } X_2 = 0) &= q^2. \end{aligned}$$

In particular, by (5.1.5), this implies (remember $q = 1 - p$)

$$\begin{aligned} Prob(X_1 = 1) &= Prob(X_1 = 1 \text{ and } X_2 = 0) + Prob(X_1 = 1 \text{ and } X_2 = 1) \\ &= pq + p^2 = p(p + q) = p. \end{aligned}$$

Similarly, $Prob(X_2 = 1) = p$. Computing,

$$Prob(X_2 = 1 \mid X_1 = 0) = \frac{Prob(X_1 = 0 \text{ and } X_2 = 1)}{Prob(X_1 = 0)} = \frac{qp}{q} = Prob(X_2 = 1),$$

so

$$Prob(X_2 = 1 \mid X_1 = 0) = Prob(X_2 = 1).$$

Thus $X_1 = 0$ has no effect on the probability that $X_2 = 1$, and similarly for the other possibilities. This is often referred to as the *independence* of the coin tosses. We conclude

Multiplication of Probabilities: Independent Coin-Tossing

With the conditional probability definition (5.1.7), a coin has no memory between successive tosses iff the probabilities at distinct tosses multiply,

$$Prob(X_1 = a_1, X_2 = a_2, \dots) = Prob(X_1 = a_1)\, Prob(X_2 = a_2) \dots \tag{5.2.2}$$

Here a_1, a_2, ... are 0 or 1.

Since we are tossing the same coin repeatedly, we can set

$$Prob(X_n = 1) = p, \qquad Prob(X_n = 0) = q = 1 - p, \qquad n \geq 1.$$

Thus all probabilities in (5.2.2) are determined by the parameter p, which may be any number between 0 and 1.



It is natural to ask for the probability of obtaining k heads in n tosses, $Prob(S_n = k)$. Here k varies between 0 and n, corresponding to all tails or all heads respectively.

There are $n + 1$ possibilities $S_n = 0$, $S_n = 1$, $S_n = 2$, ..., $S_n = n$ for the number of heads in n tosses. If we have no data to think otherwise, then all possibilities are equally likely, so one expects

$$Prob(S_n = k) = \frac{1}{n+1}, \qquad 0 \leq k \leq n.$$

Notice the total probability is 1,

$$\sum_{k=0}^{n} Prob(S_n = k) = \sum_{k=0}^{n} \frac{1}{n+1} = 1,$$

as it should be.

Assume we know the coin's bias p. Since the number of ways of choosing k heads from n tosses is the binomial coefficient $\binom{n}{k}$ (see §A.2), and the probabilities of distinct tosses multiply, the probability of k heads in n tosses is as follows.

Coin-Tossing With Known Bias

If a coin has bias p, the probability of obtaining k heads in n tosses is the binomial distribution

$$Prob(S_n = k) = \binom{n}{k} p^k (1-p)^{n-k}. \tag{5.2.3}$$

Why is this? Because the probabilities multiply, so the probability of a *specific pattern of k heads in n tosses* is $p^k(1-p)^{n-k}$. By (5.1.4), probabilities of exclusive events add, and there are $\binom{n}{k}$ exclusive events here, because $\binom{n}{k}$ is the number of ways of choosing k heads from n tosses.

By the binomial theorem,

$$\sum_{k=0}^{n} Prob(S_n = k) = \sum_{k=0}^{n} \binom{n}{k} p^k (1-p)^{n-k} = (p + 1 - p)^n = 1,$$

again as it should be.

The binomial distribution with $n = 1$ corresponds to a single coin toss, and is called the *Bernoulli distribution*. The corresponding random variable X,

$$Prob(X = 1) = p, \qquad Prob(X = 0) = 1 - p,$$

is a *Bernoulli* random variable.

The code for N samples of head-counts in n tosses is

```
from numpy import *
set_printoptions(legacy = "1.25")

from numpy.random import default_rng
samples = default_rng().binomial

n, p, N = 5, .5, 10

# counting heads from n tosses sampled N times
samples(n,p,N)
```

This returns `array([2, 2, 2, 0, 4, 3, 4, 2, 4, 2])`.

The code for the *probability* of k heads in n tosses of a coin with bias p is

```
from scipy.stats import binom

k,n,p = 5, 10, .5

B = binom(n,p)

# probability of k heads
B.pmf(k)
```

This returns `0.24609375000000003`.

More generally,

```
from scipy.stats import binom
from scipy.special import comb

# code to verify binomial pmf

def f(n,k,p): return binom(n,p).pmf(k)
def g(n,k,p): return comb(n,k,exact = True) * p**k * (1-p)**(n-k)

k,n,p = 5, 10, .5

pmf1 = array([ f(n,k,p) for k in range(n+1) ])
pmf2 = array([ g(n,k,p) for k in range(n+1) ])

allclose(pmf1,pmf2)
```

returns `True`.

Be careful to distinguish between

`numpy.random.default_rng.binomial` and `scipy.stats.binom`.

The former returns *samples* from a binomial distribution, while the latter returns a binomial *random variable*. Samples are just numbers; random variables have `cdf`'s, `pmf`'s or `pdf`'s, etc.



We explain the connection between entropy (§4.2) and coin-tossing. Recall the binomial coefficient $\binom{n}{k}$ is the number of ways of selecting k objects from n objects (A.1.1).

Toss the coin n times, and let $\#_n = \#_n(p)$ be the number of outcomes where the proportion k/n of heads is p. Here p may or may not equal the coin's bias, which we assume not known. Then the number of heads is $k = np$, so,

$$\#_n(p) = \binom{n}{np}.$$

When p is an irrational, np is replaced by the floor $\lfloor np \rfloor$, but we ignore this point. Using (A.1.2), a straightforward calculation yields the following result.[1]

Entropy and Coin-Tossing

Let $0 \le p \le 1$ be a proportion. Toss a coin n times, and let $\#_n(p)$ be the number of outcomes where the heads-proportion is p. Then

[1] This result is a precise interpretation of the original definition of entropy, due to the physicist Boltzmann (1875), as the log of the number of outcomes, or configurations, or possibilities.

$\#_n(p)$ is approximately equal to $e^{nH(p)}$ for n large.

In more detail, using Stirling's approximation (A.1.7), one can derive the asymptotic equality

$$\#_n(p) \approx \frac{1}{\sqrt{2\pi n}} \cdot \frac{1}{\sqrt{p(1-p)}} \cdot e^{nH(p)}, \qquad \text{for } n \text{ large.} \tag{5.2.4}$$

Asymptotic equality means the ratio of the two sides approaches 1 as $n \to \infty$ (see §A.7).

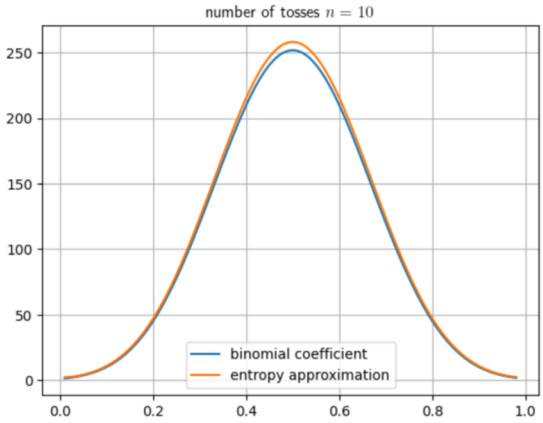


Fig. 5.6 Asymptotics of binomial coefficients.

Figure 5.6 is returned by the code below, which compares both sides of the asymptotic equality (5.2.4) for $n = 10$ and $0 \le p \le 1$.

```
from numpy import *
from scipy.special import comb
from scipy.stats import entropy
from matplotlib.pyplot import *

n = 10
p = arange(.01,1,.01)

def H(p): return entropy([p,1-p])

def stirling(n,p):
   return exp(n*H(p))/sqrt(2*n*pi*p*(1-p))


plot(p, comb(n,n*p), label = "binomial coefficient")
```

```
plot(p, stirling(n,p), label = "entropy approximation")
title("number of tosses " + "$n=" + str(n) +"$", usetex = True)
legend()

grid()
show()
```



Suppose a coin has bias q and toss the coin n times. Then we expect the long-term *proportion* of heads in n tosses to approximately equal q. Now let p be another probability, $0 \le p \le 1$.

Toss a coin n times, and let $P_n(p,q)$ be the probability of obtaining outcomes with heads-proportion p, given that the coin's bias is q.

If $p = q$, one's first guess is $P_n(p,p) \approx 1$ for n large. However, this is not correct, because $P_n(p,p)$ is specifying a specific proportion p, predicting specific behavior from the coin tosses. Because this is too specific, it turns out $P_n(p,p) \approx 0$, see Exercise **5.2.7**.

On the other hand, if $p \neq q$, we definitely expect the proportion of heads to not equal p. In other words, we expect $P_n(p,q)$ to be small for large n. In fact, when $p \neq q$, it turns out $P_n(p,q) \to 0$ exponentially, as $n \to \infty$.

We derive a formula for the speed of this decay. With $k = np$ in the binomial distribution (5.2.3),

$$P_n(p,q) = \binom{n}{np} q^{np}(1-q)^{n-np}.$$

Let $H(p,q)$ be the relative entropy (§4.2). Using (A.1.2), a straightforward calculation results in

Relative Entropy and Coin-Tossing

Assume a coin's bias is q. Toss the coin n times, and let $P_n(p,q)$ be the probability of obtaining outcomes where the heads-proportion is p. Then

$$P_n(p,q) \quad \text{is approximately equal to} \quad e^{nH(p,q)} \quad \text{for } n \text{ large.} \tag{5.2.5}$$

In more detail, using Stirling's approximation (A.1.7), one can derive the asymptotic equality

$$P_n(p,q) \approx \frac{1}{\sqrt{2\pi n}} \cdot \frac{1}{\sqrt{p(1-p)}} \cdot e^{nH(p,q)}, \qquad \text{for } n \text{ large.} \tag{5.2.6}$$

The *law of large numbers* (§5.1)) states that the heads-proportion in n tosses equals approximately q for large n. Therefore, when $p \neq q$, we expect the probabilities that the heads-proportions equal p become successively smaller as n get larger, and in fact vanish when $n = \infty$. Since $H(p, q) < 0$ when $p \neq q$, (5.2.6) implies this is so. Thus (5.2.6) may be viewed as a quantitative strengthening of the law of large numbers, in the setting of coin-tossing.



Now we assume the coin parameter p is unknown, and we interpret (5.2.3) as the conditional probability that $S_n = k$ given knowledge of p, which we rewrite as

$$Prob(S_n = k \mid p) = \binom{n}{k} p^k (1-p)^{n-k}, \qquad 0 \leq k \leq n. \tag{5.2.7}$$

By additivity of probabilities, $Prob(S_n = k)$ is the sum of the probabilities $Prob(S_n = k \text{ and } p)$ over $0 \leq p \leq 1$.

By the conditional probability chain rule (5.1.8),

$$Prob(S_n = k \text{ and } p) = Prob(S_n = k \mid p)\, Prob(p).$$

Thus $Prob(S_n = k)$ is the sum of $Prob(S_n = k \mid p)\, Prob(p)$ over $0 \leq p \leq 1$. Since p varies continuously over $0 \leq p \leq 1$, the sum is replaced by the integral, and

$$Prob(S_n = k) = \int_0^1 Prob(S_n = k \mid p)\ Prob(p)\, dp.$$

Integrals are reviewed in §A.6.

Since we don't know anything about p, it's simplest to assume a *uniform* prior probability $Prob(p) = 1$. Based on this, we obtain

$$Prob(S_n = k) = \int_0^1 \binom{n}{k} p^k (1-p)^{n-k}\, dp. \tag{5.2.8}$$

Usually, this integral is evaluated using integration by parts. However, it is easier to evaluate this for all $0 \leq k \leq n$ at once, by writing

$$I(c) = \sum_{k=0}^{n} c^k\, Prob(S_n = k).$$

Using (5.2.8) and the binomial theorem (A.2.7), $I(c)$ equals

$$\int_0^1 \left(\sum_{k=0}^{n} \binom{n}{k} c^k p^k (1-p)^{n-k} \right) dp = \int_0^1 (1 - p + cp)^n\, dp.$$

If we set

$$f(p) = (1 - p + cp)^n, \qquad F(p) = \frac{(1 - p + cp)^{n+1}}{(c-1)(n+1)},$$

then $F'(p) = f(p)$ (see (4.1.5)). By the fundamental theorem of calculus (A.6.2),

$$I(c) = F(1) - F(0) = \frac{1}{n+1} \cdot \frac{c^{n+1} - 1}{c - 1}. \tag{5.2.9}$$

But by (A.3.5) with n replaced by $n + 1$,

$$\frac{1}{n+1} \cdot \frac{c^{n+1} - 1}{c - 1} = \sum_{k=0}^{n} c^k \cdot \frac{1}{n+1}.$$

Matching coefficients of powers of c here and in $I(c)$, we conclude

Coin-Tossing With Unknown Bias

If a coin has unknown bias p, distributed uniformly on $0 \le p \le 1$, then the probability of obtaining k heads in n tosses is

$$Prob(S_n = k) = \frac{1}{n+1}, \qquad k = 0, 1, 2, \ldots, n. \tag{5.2.10}$$

Notice the difference: In (5.2.3), we know the coin's bias p, and obtain the binomial distribution, while in (5.2.10), since we don't know p, and there are $n + 1$ possibilities $0 \le k \le n$, we obtain the uniform distribution $1/(n + 1)$.



We now turn things around: Suppose we toss the coin n times, and obtain k heads. How can we use this data to estimate the coin's bias p?

To this end, we introduce the fundamental

Bayes Theorem I

$$Prob(A \mid B) = \frac{Prob(B \mid A) \cdot Prob(A)}{Prob(B)}. \tag{5.2.11}$$

The proof of Bayes Theorem is straightforward:

$$\begin{aligned} Prob(A \mid B) &= \frac{Prob(A \text{ and } B)}{Prob(B)} \\ &= \frac{Prob(A \text{ and } B)}{Prob(A)} \cdot \frac{Prob(A)}{Prob(B)} \\ &= Prob(B \mid A) \cdot \frac{Prob(A)}{Prob(B)}. \end{aligned}$$

The depth of the result lies in its widespread usefulness.

We now write Bayes Theorem to compute

$$Prob(p \mid S_n = k) = Prob(S_n = k \mid p) \cdot \frac{Prob(p)}{Prob(S_n = k)}. \tag{5.2.12}$$

But $Prob(S_n = k \mid p)$ is as in (5.2.7), $Prob(S_n = k)$ is as in (5.2.10). Since p is uniformly distributed, $Prob(p) = 1$. Inserting these quantities into (5.2.12) leads to

$$(n+1) \cdot \binom{n}{k} \cdot p^k (1-p)^{n-k} = \frac{(n+1)!}{k!(n-k)!} \cdot p^k (1-p)^{n-k}. \tag{5.2.13}$$

Summarizing,

Posterior Probability Given k Heads in n Tosses

Assume the unknown bias p of a coin is uniformly distributed on $0 \le p \le 1$. Then the conditional probability $Prob(p \mid S_n = k)$ that the bias is p given k heads in n tosses equals (5.2.13).

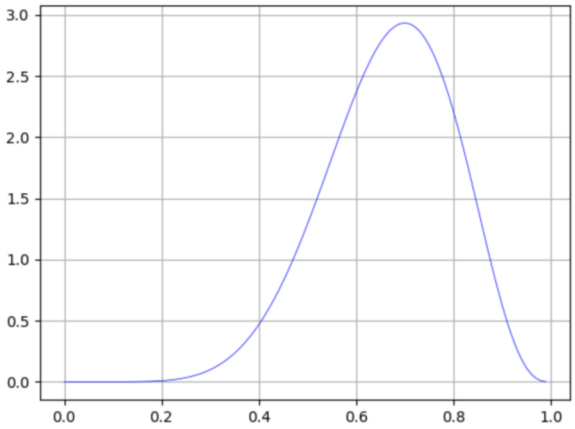


Fig. 5.7 The posterior density of p given 7 heads in 10 tosses.

In (5.2.7), p is fixed, and k is the variable. In (5.2.13), k is fixed, and p is the variable. Nevertheless, we call (5.2.13) the *binomial density*. This posterior density for $(n, k) = (10, 7)$ is plotted in Figure 5.7, and it peaks at $k/n = 7/10$ (Exercise **4.1.5**). The code generating Figure 5.7 is

```
from matplotlib.pyplot import *
from numpy import arange
from scipy.stats import binom

n = 10
k = 7

def f(p): return (n+1) * binom(n,p).pmf(k)

p = arange(0,1,.01)
plot(p,f(p),color = "blue",linewidth = .5)

grid()
show()
```



Because Bayes Theorem is so useful, here are two alternate forms. Suppose A_1, A_2, ..., A_d are several exclusive and exhaustive events, so

$$Prob(A_1) + Prob(A_2) + \cdots + Prob(A_d) = 1.$$

Then by the law of total probability (5.1.10) and the first version (5.2.11), we have the second version

Bayes Theorem II

If A_1, A_2, ... are several exclusive and exhaustive events, then for $i = 1, 2, \ldots$, $Prob(A_i \mid B)$ equals

$$\frac{Prob(B \mid A_i)\, Prob(A_i)}{Prob(B \mid A_1)\, Prob(A_1) + Prob(B \mid A_2)\, Prob(A_2) + \ldots}. \tag{5.2.14}$$

In particular, $Prob(A \mid B)$ equals

$$\frac{Prob(B \mid A)\, Prob(A)}{Prob(B \mid A)\, Prob(A) + Prob(B \mid A^c)\, Prob(A^c)}. \tag{5.2.15}$$

As an example, suppose 20% of the population are smokers, and the prevalence of lung cancer among smokers is 90%. Suppose also 80% of non-smokers are cancer-free. Then what is the probability that someone who has cancer is actually a smoker?

To use the second version, set $A = smoker$ and $B = cancer$. This means A is the event that a randomly sampled person is a smoker, and B is the event that a randomly sampled person has cancer. Then

$$Prob(A) = .2, \qquad Prob(B \mid A) = .9, \qquad Prob(B^c \mid A^c) = .8.$$

From this, we have

$$Prob(B \mid A^c) = 1 - Prob(B^c \mid A^c) = 1 - .8 = .2,$$

and

$$\begin{aligned} Prob(A \mid B) &= \frac{Prob(B \mid A)\, Prob(A)}{Prob(B \mid A)\, Prob(A) + Prob(B \mid A^c)\, Prob(A^c)} \\ &= \frac{.9 \times .2}{.9 \times .2 + .2 \times .8} = .52941. \end{aligned}$$

Thus the probability that a person with lung cancer is indeed a smoker is 53%.



To describe the third version of Bayes theorem, bring in the logistic function. Let

$$p = \sigma(y) = \frac{1}{1 + e^{-y}}. \tag{5.2.16}$$

This is the *logistic function* or *sigmoid function*. The logistic function takes as inputs real numbers y, and returns as outputs probabilities p (Figure 5.8), and is plotted in Figure 5.9.

We think of the input y as an activation energy, the output p as the probability of activation, and $y = 0$ as the activation threshold.

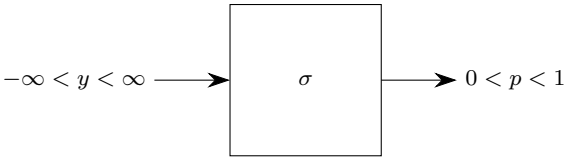


Fig. 5.8 The logistic function takes real numbers to probabilities.

In Python, σ is the `expit` function.

```
from scipy.special import expit

y = 1.5
p = expit(y)
p
```

The multinomial or vector-valued version of $\sigma(y)$ is the softmax function §5.6.

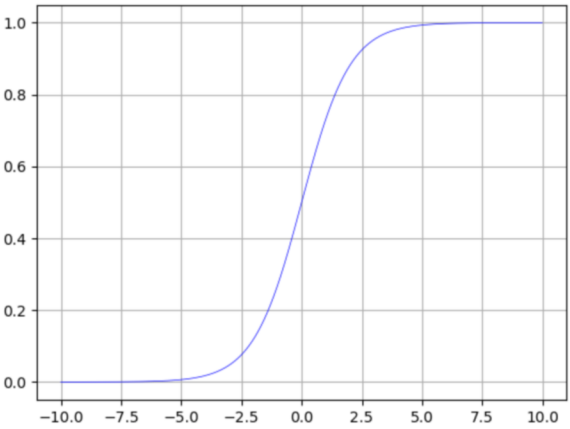


Fig. 5.9 The logistic function.

Dividing the numerator and denominator of (5.2.15) by the numerator, we obtain Bayes Theorem in terms of log-probability,

Bayes Theorem III

$$Prob(A \mid B) = \sigma\left(\log\left(\frac{Prob(B \mid A)\, Prob(A)}{Prob(B \mid A^c)\, Prob(A^c)}\right)\right). \tag{5.2.17}$$

When there are several mutually exclusive events $A_1, A_2, \ldots, A_d$, the same result holds with σ the softmax function (§5.6).

Here is an application of the third version. Suppose we have two groups of scalars, selected as follows. A fair coin is tossed. Depending on the result, select a scalar x distributed randomly according to (§5.4)

$$\begin{aligned} Prob(x \mid H) &= e^{-(x-\mu_H)^2/2}, \\ Prob(x \mid T) &= e^{-(x-\mu_T)^2/2}. \end{aligned} \tag{5.2.18}$$

Here the two choices are centered at μ_H and μ_T, according to whether the coin toss results in heads or tails.

With this setup, the question is: Given a scalar x, what is the probability x is in the heads group? In other words, what is

$$Prob(H \mid x)?$$

This question is begging for Bayes theorem.

Assume the two groups are distinct, by assuming $\mu_H \neq \mu_T$, and let

$$m = \mu_H - \mu_T, \qquad b = \frac{1}{2}\mu_T^2 - \frac{1}{2}\mu_H^2.$$

Then $m \neq 0$. Since $Prob(H) = Prob(T)$, here we have $Prob(A) = Prob(A^c)$. Inserting the formulas for $Prob(x \mid H)$ and $Prob(x \mid T)$ leads to the log-probability

$$\log\left(\frac{Prob(x \mid H)\, Prob(H)}{Prob(x \mid T)\, Prob(T)}\right) = mx + b. \tag{5.2.19}$$

By (5.2.17),

$$Prob(H \mid x) = \sigma(mx + b).$$

This shows the group membership of x is determined by the activation threshold $mx + b = 0$, or cut-off $x^* = -b/m$. Simplifying, the cut-off is

$$x^* = -\frac{b}{m} = -\frac{1}{2}\,\frac{\mu_T^2 - \mu_H^2}{\mu_H - \mu_T} = \frac{1}{2}\,(\mu_H + \mu_T)\,,$$

which is the midpoint of the line segment joining μ_H and μ_T.

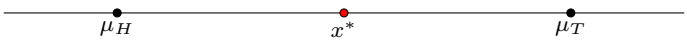


Fig. 5.10 Decision boundary in $\mathbf{R}$.

More generally, if the points x are in $\mathbf{R}^d$, then the same question may be asked, with

$$\begin{aligned} Prob(x \mid H) &= e^{-|x-\mu_H|^2/2}, \\ Prob(x \mid T) &= e^{-|x-\mu_T|^2/2}. \end{aligned} \tag{5.2.20}$$

In this case, if $\mu_H \neq \mu_T$, m is a nonzero vector, and b is still a scalar,

$$m = \mu_H - \mu_T, \qquad b = \frac{1}{2}|\mu_T|^2 - \frac{1}{2}|\mu_H|^2.$$

Then the cut-off or *decision boundary* between the two groups is the hyperplane

$$m \cdot x + b = 0$$

passing through the midpoint $x^* = (\mu_H + \mu_T)/2$, and orthogonal to the vector $m = \mu_H - \mu_T$. Written this way, the probability

$$Prob(H \mid x) = \sigma(m \cdot x + b) \tag{5.2.21}$$

is the output of a single-layer perceptron (§7.2). For hyperplanes, see §4.5.

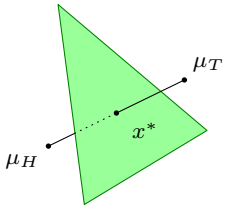


Fig. 5.11 Decision boundary in $\mathbf{R}^3$.

Exercises

Exercise 5.2.1 A fair coin is tossed. What is the probability of obtaining 5 heads in 8 tosses?

Exercise 5.2.2 A coin with bias p is tossed. What is the probability of obtaining 5 heads in 8 tosses?

Exercise 5.2.3 A coin with bias p is tossed 8 times and 5 heads are obtained. What is the most likely value for p?

Exercise 5.2.4 A coin with unknown bias p is tossed 8 times and 5 heads are obtained. Assuming a uniform prior for p, what is the probability that p lies between 0.5 and 0.7? Use `scipy.integrate.quad` (§A.6) to integrate (5.2.13) over $0.5 \leq p \leq 0.7$.)

Exercise 5.2.5 A fair coin is tossed n times. Sometimes you get more heads than tails, sometimes the reverse. If you're really lucky, the number of heads may equal exactly the number of tails. What is the least n for which the probability of this happening is less than 10%?

Exercise 5.2.6 A coin is tossed. Depending on the result, select a scalar x at random with normal probability densities as in (5.2.18). If the coin bias is p, compute the decision boundary.

Exercise 5.2.7 If a fair coin is tossed $2n$ times, show the probability of obtaining n heads and n tails is approximately $1/\sqrt{\pi n}$ for n large. (Use (5.2.6).)

Exercise 5.2.8 Show

$$\int_0^1 p^{\alpha-1}(1-p)^{\beta-1}\,dp = \frac{(\alpha-1)!(\beta-1)!}{(\alpha+\beta-1)!}.$$

(The integral of (5.2.13) over $0 < p < 1$ is 1.)

Exercise 5.2.9 The posterior is an *updating* of the prior of p: Based on our observing k heads in n tosses, our estimate of p goes from a uniform prior to a binomial posterior (5.2.13) peaking at $\hat{p} = k/n$. Now assume a binomial prior for p as in (5.2.13), with (k, n) replaced by (α, β) with $\beta > \alpha > 0$. This prior peaks at $\hat{p} = \alpha/\beta$ (Exercise **4.1.5**). Given k heads in n tosses, show the posterior $Prob(p \mid S_n = k)$ is binomial peaking at $\hat{p} = (k+\alpha)/(n+\beta)$.

5.3 Random Variables

Suppose a real number x is selected at random. Even if we don't know anything about x, we know x is a number, so our *confidence* that $-\infty < x < \infty$ equals 100%, the *chance* that x satisfies $-\infty < x < \infty$ equals 1, and the *probability* that x satisfies $-\infty < x < \infty$ equals 1.

When we say x is "selected at random", we think of a machine X that is the source of the numbers x (Figure 5.12). Such a source of numbers is called a *random number*, short for random number generator, just like a source for apples should be called a random apple, short for random apple generator. It is standard to call such a source X a random variable.

Definition of Random Variable

A *random variable* X is a function of outcomes: Each outcome results in a sample x of X.

In §1.3, this was called *vectorization*. In this section, random variables are scalar-valued. In §5.5 and §6.4, they are vector-valued.

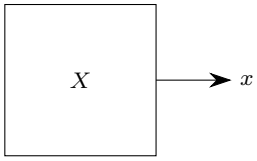


Fig. 5.12 When we sample X, we get x.

Let X and Y be random variables. Since X is a function of outcomes, the outcomes where $X = 5$ is an event A (§5.1). For simplicity, we write $X = 5$

for this event, rather than introduce the superfluous symbol A. Similarly, $(X = 5 \text{ and } Y = 7)$ is the event consisting of outcomes where $X = 5$ and $Y = 5$. With this understood, $Prob(X = 5)$ and $Prob(X = 5 \text{ and } Y = 7)$ are well-defined probabilities of events.

Let X be a random variable and let x be a sample of X. What is the chance, what is our confidence, what is the probability, of selecting x from an interval $[a, b]$? If we write

$$Prob(a < X < b)$$

for this quantity, then we are asking to compute the probability of the event that X lies in the interval $[a, b]$. If we don't know anything about X, then we can't figure out the probability, and there is nothing we can say. Knowing something about X means knowing the *distribution* of X: Where X is more likely to be and where X is less likely to be. Any quantity X where probabilities of events $Prob(a < X < b)$ can be computed is a random variable.

For example, suppose we want to estimate the proportion of American college students who have a smart phone. Instead of asking every student, we take a sample and make an estimate based on the sample.

Let p be the actual proportion of students that in fact have a smartphone. If there are N students in total, and m of them have a smartphone, then $p = m/N$. For each student, let

$$X = \begin{cases} 1, & \text{if the student has a smartphone,} \\ 0, & \text{if not.} \end{cases}$$

Then X is a random variable: X is a machine that returns 0 or 1 depending on the chosen student.

A random variable taking on only two values is a *Bernoulli* random variable. Since X takes on the two values 0 and 1, X is a Bernoulli random variable.

Throughout we adopt the convention that random variables are written in uppercase, X, while the numbers resulting when sampled are written lowercase, x. In other words, *when we sample X, we obtain x.*

We will meet many different random variables $X, Y, Z, \ldots$. The letter Z is reserved for a standard random variable, one having mean zero and variance one. Samples from Z are written as z.



Suppose X is a random variable taking on three values a, b, c with probabilities p, q, r,

$$P(X = a) = p, \qquad P(X = b) = q, \qquad P(X = c) = r.$$

Then the *mean* or *average* or *expected value* of X is

$$E(X) = ap + bq + cr.$$

Since $p + q + r = 1$, the expected value of X lies between the greatest of a, b, c, and the least,

$$\min(a, b, c) \leq E(X) \leq \max(a, b, c).$$

Let $\mu = E(X)$ be the mean. The *variance* of X is a measure of how far X deviates from its mean,

$$Var(X) = E((X - \mu)^2).$$

For the random variable X above,

$$Var(X) = (a - \mu)^2 \cdot p + (b - \mu)^2 \cdot q + (c - \mu)^2 \cdot r.$$

By expanding the squares, one has the identity

$$Var(X) = E(X^2) - \mu^2.$$

This is valid for any random variable X.



Random variables are either *discrete* or *continuous*. Even though the derivations and identities below are carried out in the context of discrete random variables, these results remain valid in the context of continuous random variables.

In the text and exercises, we consider three discrete random variables,

$$Bernoulli(p) \quad \rightarrow \quad binomial(n, p) \quad \rightarrow \quad Poisson(\lambda).$$

The Bernoulli random variable is the outcome of a single toss of a coin with bias p, and the binomial random variable is the outcome of n tosses of a coin with bias p.

As we see below (5.3.23), the mean of a binomial random variable is np. If we let the number of tosses grow without bound, $n \rightarrow \infty$, while keeping the mean fixed at $\lambda = np$, we obtain the Poisson random variable.

In the text and the exercises, we consider several continuous random variables,

$$uniform, \quad exponential, \quad logistic, \quad arcsine,$$

and

$$normal, \quad chi\text{-}squared, \quad student.$$



A random variable X is *discrete* if X takes on discrete values $x_1, x_2, \ldots$, with probabilities $p_1, p_2, \ldots$. Here the values may be scalars or vectors, and there may be finitely many or infinitely many values. If all the values are equal to the same scalar μ, then we say X is a constant.

For a discrete random variable, the *probability mass function* (`pmf` in Python) is

$$p(x) = Prob(X = x),$$

and the *cumulative distribution function* (`cdf` in Python) is

$$F(x) = Prob(X \leq x).$$

Then $p_k = p(x_k)$. By additivity of probabilities (5.1.1), $F(x)$ is the sum of $p(x_k)$ over all $x_k \leq x$.

Definition of Expectation: Discrete Case

Let X take on values $x_1, x_2, \ldots$, with probabilities $p_1, p_2, \ldots$. The *expectation* of X is

$$E(X) = x_1 p_1 + x_2 p_2 + \ldots. \tag{5.3.1}$$

$E(X)$ is also called the *mean* or *average* or *first moment* of X, and is usually denoted μ.

When there are N values, and we take $p_1 = p_2 = \cdots = 1/N$, we say the values are *equally likely* or X is *uniform*. In this case, the mean reduces to (1.3.1).

More generally, let $f(x)$ be a function. The *mean* or *expectation* of $f(X)$ is

$$E(f(X)) = f(x_1)p_1 + f(x_2)p_2 + \ldots \tag{5.3.2}$$

Since the total probability is one, when $f(x) = 1$,

$$E(1) = p_1 + p_2 + \cdots = 1.$$

If a is a constant then the values of aX are $ax_1, ax_2, \ldots$, with probabilities $p_1, p_2, \ldots$, so

$$E(aX) = ax_1p_1 + ax_2p_2 + \cdots = a(x_1p_1 + x_2p_2 + \ldots) = aE(X). \tag{5.3.3}$$

When $f(x) = x^2$, the mean of $f(X)$ is the *second moment*

$$E(X^2) = x_1^2 p_1 + x_2^2 p_2 + \ldots$$

When $f(x) = e^{tx}$, the mean of $f(X)$ is the *moment-generating function*

$$M(t) = E\left(e^{tX}\right) = e^{tx_1}p_1 + e^{tx_2}p_2 + \dots$$

The log of the moment-generating function is the *cumulant-generating function*

$$Z(t) = \log M(t) = \log E\left(e^{tX}\right).$$



A basic property of expectation is the

Linearity of the Expectation

For any random variables X and Y and constants a and b,

$$E(aX + bY) = aE(X) + bE(Y). \tag{5.3.4}$$

Linearity is used routinely whenever we compute expectations, and is deceptively simple to state. Because the derivation of linearity uses additivity of probabilities (5.1.1), it is instructive to go over this carefully.

Let X have values $x_1, x_2, \dots$, and probabilities $p_1, p_2, \dots$, and let Y have values $y_1, y_2, \dots$, and probabilities $q_1, q_2, \dots$.

If

$$r_{jk} = Prob(X = x_j \text{ and } Y = y_k), \qquad j, k = 1, 2, \dots,$$

then, by additivity of probabilities (5.1.1),

$$\begin{aligned} p_j &= Prob(X = x_j) \\ &= Prob(X = x_j \text{ and } Y = y_1) + Prob(X = x_j \text{ and } Y = y_2) + \dots \\ &= r_{j1} + r_{j2} + \dots = \sum_k r_{jk}. \end{aligned}$$

Similarly,

$$q_k = r_{1k} + r_{2k} + \dots = \sum_j r_{jk}.$$

Since the values of $X + Y$ are $x_j + y_k$, with probabilities r_{jk}, $j, k = 1, 2, \dots$, by definition of expectation,

$$E(X + Y) = \sum_j \sum_k (x_j + y_k) r_{jk}.$$

But this double sum may be written in two parts as

$$\sum_j \sum_k x_j r_{jk} + \sum_j \sum_k y_k r_{jk} = \sum_j x_j p_j + \sum_k y_k q_k = E(X) + E(Y).$$

We conclude

$$E(X+Y) = E(X) + E(Y).$$

Since we already know $E(aX) = aE(X)$ (5.3.3), this derives linearity.



Let μ be the mean of a random variable X. The *variance* of X is

$$Var(X) = E((X-\mu)^2). \tag{5.3.5}$$

The variance measures the spread of X about its mean. Since the mean of aX is $a\mu$, the variance of aX is the mean of $(aX - a\mu)^2 = a^2(X-\mu)^2$. Thus

$$Var(aX) = a^2\, Var(X).$$

However, the variance of a sum $X+Y$ is not simply the sum of the variances of X and Y: This only happens if X and Y are independent, see (5.3.21).

Using (5.3.9), we can view a dataset as the samples of a random variable X. In this case, the mean and variance of X are the same as the mean and variance of the dataset, as defined by (1.4.1) and (1.4.2).

When X is a constant, then $X = \mu$, so $Var(X) = 0$. Conversely, if $Var(X) = 0$, then by definition

$$0 = (x_1-\mu)^2 p_1 + (x_2-\mu)^2 p_2 + \dots,$$

so all values are equal to μ, hence X is a constant.

The square root of the variance is the *standard deviation*. If we write $Var(X) = \sigma^2$, then the standard deviation is σ.

Expanding the square in (5.3.5),

$$Var(X) = E(X^2) - 2\mu E(X) + \mu^2.$$

Since $\mu = E(X)$, we obtain the alternate formula for the variance

$$Var(X) = E(X^2) - E(X)^2. \tag{5.3.6}$$

This displays the variance in terms of the first moment $E(X)$ and the second moment $E(X^2)$. Equivalently,

$$E(X^2) = \mu^2 + \sigma^2 = (E(X))^2 + Var(X). \tag{5.3.7}$$



Every dataset $x_1, x_2, \dots, x_N$ may be viewed as the samples of a random variable X, as follows.

Define

$$E(X) = \frac{1}{N}\sum_{k=1}^{N} x_k. \tag{5.3.8}$$

Then $E(X)$ is the mean of the random variable X associated to the dataset. Similarly,

$$E(X^2) = \frac{1}{N}\sum_{k=1}^{N} x_k^2$$

is the second moment of the random variable X associated to the dataset.

More generally, given any function $f(x)$, we have the mean of $f(x_1)$, $f(x_2)$, $\ldots$, $f(x_N)$,

$$E(f(X)) = \frac{1}{N}\sum_{k=1}^{N} f(x_k). \tag{5.3.9}$$

Given any interval (a, b), we may set

$$f(x) = \begin{cases} 1, & a < x < b, \\ 0, & \text{otherwise}, \end{cases}$$

Then $f(x_k)$ is only counted when $a < x_k < b$, so

$$E(f(X)) = \frac{1}{N}\sum_{k=1}^{N} f(x_k) = \frac{\#\{x_k : a < x_k < b\}}{N} = Prob(a < X < b)$$

is the probability that a randomly selected sample lies in (a, b).

This shows probabilities are special cases of means. Since we can compute means by (5.3.9), we can compute probabilities for X. This is what is meant by "selecting a random sample from the dataset".



The simplest discrete random variable is the Bernoulli random variable X resulting from a coin toss, with $X = 1$ corresponding to heads, and $X = 0$ corresponding to tails,

$$Prob(X = 1) = p, \qquad Prob(X = 0) = 1 - p.$$

We say X is *Bernoulli with bias p*.

The probability mass function is

$$p(k) = \begin{cases} 1 - p, & \text{if } k = 0, \\ p, & \text{if } k = 1. \end{cases}$$

This is presented graphically in Figure 5.13.

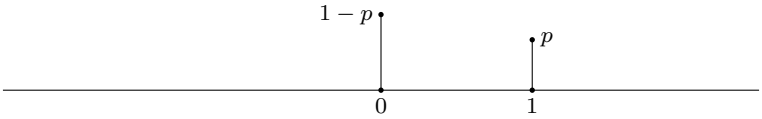


Fig. 5.13 Probability mass function $p(x)$ of a Bernoulli random variable.

The mean of the Bernoulli random variable is

$$E(X) = 1 \cdot Prob(X = 1) + 0 \cdot Prob(X = 0) = 1 \cdot p + 0 \cdot (1 - p) = p.$$

The second moment is

$$E\left(X^2\right) = 1^2 \cdot Prob(X = 1) + 0^2 \cdot Prob(X = 0) = p.$$

From this,

$$Var(X) = E(X^2) - E(X)^2 = p - p^2 = p(1 - p).$$

When $p = 0$ or $p = 1$, the variance is zero, there is no randomness. When $p = 1/2$, the randomness is maximized and the maximum variance equals $1/4$.

Mean and Variance of Bernoulli

If X is Bernoulli with bias p, then

$$E(X) = p, \qquad var(X) = p(1 - p).$$

The moment-generating function is

$$M(t) = E\left(e^{tX}\right) = e^{t1}p + e^{t0}(1 - p) = pe^t + (1 - p).$$

The cumulant-generating function is

$$Z(t) = \log M(t) = \log(pe^t + 1 - p).$$

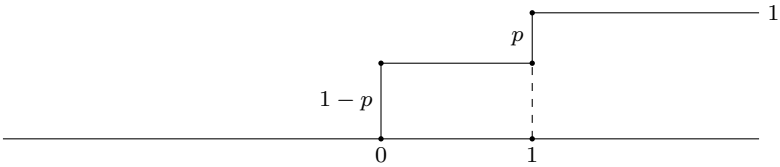


Fig. 5.14 Cumulative distribution function $F(x)$ of a Bernoulli random variable.

The cumulative distribution function $F(x)$ is in Figure 5.14. Because the Bernoulli random variable takes on only the values $x = 0, 1$, these are the values where $F(x)$ jumps.

More generally, let A be any event, and define

$$B = \begin{cases} 1, & \text{if the outcome is in } A, \\ 0, & \text{if the outcome is in } A^c. \end{cases} \tag{5.3.10}$$

Then B has values 1 and 0 with probabilities

$$p = Prob(B = 1) = Prob(A), \qquad 1 - p = Prob(B = 0) = Prob(A^c),$$

hence B is Bernoulli with bias p. We say B is *the Bernoulli random variable corresponding to event A*.

By definition of B,

$$E(B) = p, \qquad Var(B) = p(1 - p).$$

The relation between A and B is discussed further in Exercise **5.3.2**.

Bernoulli random variables are used to count sample proportions. Let X be a random variable, and fix a threshold a for X. Let $X_1, X_2, \ldots, X_n$ be a repeated sampling of X, and let $B_1, B_2, \ldots, B_n$ be the Bernoulli random variables corresponding to the events $X_1 > a$, $X_2 > a$, $\ldots$, $X_n > a$. Then

$$\hat{p} = \frac{B_1 + B_2 + \cdots + B_n}{n} \tag{5.3.11}$$

is the *proportion of samples* greater than threshold a. This is a special case of vectorization (§1.3).

Let X be any random variable. Since the total probability is one,

$$M(0) = E(e^{0X}) = E(1) = 1.$$

The derivative of the moment-generating function is

$$M'(t) = E\left(Xe^{tX}\right).$$

When $t = 0$,

$$M'(0) = E(X) = \mu.$$

Similarly, since the derivative of $\log x$ is $1/x$, for the cumulant-generating function,

$$Z'(0) = \frac{M'(0)}{M(0)} = E(X) = \mu.$$

The second derivative of $M(t)$ is

$$M''(t) = E\left(X^2 e^{tX}\right),$$

so $M''(0)$ is the second moment $E(X^2)$.

By the quotient rule, the second derivative of $Z(t)$ is

$$Z''(t) = \left(\frac{M'(t)}{M(t)}\right)' = \frac{M''(t)M(t) - M'(t)^2}{M(t)^2}.$$

Inserting $t = 0$, and recalling (5.3.6), we have

Cumulant-Generating Function and Variance

Let $Z(t)$ be the cumulant-generating function of a random variable X. Then

$$Z'(0) = E(X) \qquad \text{and} \qquad Z''(0) = Var(X). \tag{5.3.12}$$



In §5.1, we discussed independence of events. Now we do the same for random variables.

Definition of Uncorrelated

Random variables X and Y are *uncorrelated* if

$$E(XY) = E(X)\,E(Y). \tag{5.3.13}$$

Otherwise, we say X and Y are *correlated*.

By (5.3.6), a random variable X is always correlated to itself, unless it is a constant.



Suppose X and Y take on the values $X = \pm 1$ and $Y = 0, 1$ with the probabilities

$$(X,Y) = \begin{cases} (1,1) & \text{with probability } a, \\ (1,0) & \text{with probability } b, \\ (-1,1) & \text{with probability } b, \\ (-1,0) & \text{with probability } c. \end{cases} \tag{5.3.14}$$

We investigate when X and Y are uncorrelated. Here $a > 0$, $b > 0$, and $c > 0$.

First, because the total probability equals 1,

$$a + 2b + c = 1. \tag{5.3.15}$$

Also we have

$$Prob(X = 1) = a+b = Prob(Y = 1),\ Prob(X = -1) = b+c = Prob(Y = 0),$$

and

$$E(X) = a - c, \qquad E(Y) = a + b.$$

Now X and Y are uncorrelated if

$$a - b = E(XY) = E(X)E(Y) = (a - c)(a + b). \tag{5.3.16}$$

Solving (5.3.15), (5.3.16) using Python,

```
from sympy import *

a,b,c = symbols('a,b,c')
eq1 = a + 2*b + c - 1
eq2 = a - b - (a-c)*(a+b)
solutions = solve([eq1,eq2],a,b)

solutions
```

we see X and Y are uncorrelated if

$$b = \sqrt{c} - c, \qquad a = c - 2\sqrt{c} + 1. \tag{5.3.17}$$

For example, X and Y are uncorrelated when $c = 1/4$, which leads to $a = b = 1/4$. Also, X and Y are uncorrelated if $c = .01$, which leads to $a = .81$ and $b = .09$.

Let X and Y be random variables. We say X and Y are *independent* if all powers of X are uncorrelated with all powers of Y.

Definition of Independence

Random variables X and Y are independent if

$$E(X^n Y^m) = E(X^n)\, E(Y^m) \tag{5.3.18}$$

for all positive powers n and m. When X and Y are discrete, this is equivalent to the events $X = x$ and $Y = y$ being independent, for every value x of X and every value y of Y.

Clearly, if X and Y are independent, then, by taking $n = 1$ and $m = 1$, X and Y are uncorrelated.

Suppose X and Y satisfy (5.3.14) and (5.3.17). Since $X = \pm 1$, $X^n = 1$ for n even and $X^n = X$ for X odd. Since $Y = 0, 1$, $Y^n = Y$ for all n. This is enough to show that, in this case, X and Y uncorrelated is equivalent to X and Y independent. However, this is certainly not true in general.

Here is an example of uncorrelated random variables that are not independent. Let X, Y be as above and set $U = XY$. We check when U and Y are uncorrelated versus when they are independent. As before, check that $E(UY) = E(U)E(Y)$ is equivalent to

$$a - b = (a - b)(a + b).$$

This happens in one of two cases. Either $a - b \neq 0$, or $a - b = 0$. If $a - b \neq 0$, then canceling $a - b$ leads to $a + b = 1$. By (5.3.15), this leads to $b + c = 0$, which can't happen, since both b and c are positive or zero. Hence we must have the other case, $a - b = 0$. By (5.3.15), this leads to

$$a = \frac{1}{3} - \frac{c}{3}, \qquad b = \frac{1}{3} - \frac{c}{3}. \tag{5.3.19}$$

Thus U and Y are uncorrelated when (5.3.19) holds, for any choice of c.

However, since $X^2 = 1$ and $Y^2 = Y$, $U^2 = Y$, so U^2 and Y are always correlated, unless Y is constant. Hence U and Y are never independent, unless Y is constant. Note Y is a constant when $a = 1$ or $c = 1$.



Let X and Y be random variables. The *joint* moment-generating function of the pair (X, Y) is

$$M_{X,Y}(s, t) = E\left(e^{sX+tY}\right).$$

Expanding the exponentials into their series, and using (5.3.18), one can show

Independence and Moment-Generating Functions

Let X and Y be random variables. Then X and Y are independent if their moment-generating functions multiply,

$$M_{X,Y}(s, t) = M_X(s)\, M_Y(t).$$

As a special case, choosing $s = t$, we see

Independent Sums and Moment-Generating Functions

Let X and Y be independent random variables. Then the moment-generating function of $X + Y$ is

$$M_{X+Y}(t) = M_X(t)M_Y(t). \tag{5.3.20}$$



As an illustration, consider an ordinary dice with $X = 1$, $X = 2$, ..., $X = 6$ equally probable. Then $Prob(X = k) = 1/6$, $k = 1, 2, \ldots, 6$. Now suppose we have a random variable Y with values $Y = 0$, $Y = 1$, ...,$Y = 6$, and assume X and Y are independent.

If we are told the sum $X + Y$ is uniform over $1 \leq X + Y \leq 12$, how should we choose the probabilities for $Y = 0$, $Y = 1$, ...,$Y = 6$?

To answer this, we use (5.3.20). By Exercise **5.3.1**,

$$M_X(t) = \frac{1}{6}\frac{e^{7t} - e^t}{e^t - 1}.$$

By Exercise **5.3.1** again,

$$M_{X+Y}(t) = \frac{1}{12}\frac{e^{13t} - e^t}{e^t - 1},$$

It follows, by (5.3.20),

$$\frac{1}{12}\frac{e^{13t} - e^t}{e^t - 1} = \frac{1}{6}\frac{e^{7t} - e^t}{e^t - 1} \cdot M_Y(t).$$

Factoring

$$e^{13t} - e^t = e^t(e^{6t} - 1)(e^{6t} + 1), \qquad e^{7t} - e^t = e^t(e^{6t} - 1),$$

we obtain

$$M_Y(t) = \frac{1}{2}(e^{6t} + 1).$$

This says

$$Prob(Y = 0) = \frac{1}{2}, \qquad Prob(Y = 6) = \frac{1}{2},$$

and all other probabilities are zero.



Taking the log in (5.3.20), independence is related to cumulant-generating functions as follows.

Independent Sums and Cumulant-Generating Functions

Let X and Y be independent random variables. Then the cumulant-generating function of $X + Y$ is

$$Z_{X+Y}(t) = Z_X(t) + Z_Y(t).$$

Taking the second derivative, plugging in $t = 0$, and using (5.3.12), we obtain

$$Var(X + Y) = Var(X) + Var(Y).$$

This holds when X and Y are independent. In general, the result is

Independent Sums and Variances

Let $X_1, X_2, \ldots, X_n$ be independent random variables, and let

$$S = X_1 + X_2 + \cdots + X_n.$$

Then

$$Var(S) = Var(X_1) + Var(X_2) + \cdots + Var(X_n). \tag{5.3.21}$$

While this result depends strongly on independence, the corresponding result for means

$$E(S) = E(X_1) + E(X_2) + \cdots + E(X_n)$$

is valid for any sum, by linearity of the expectation (5.3.4).



The next simplest discrete random variable is the binomial random variable,

$$S = X_1 + X_2 + \cdots + X_n$$

obtained from n independent Bernoulli random variables.

Then S has values 0, 1, 2, ..., n, and the probability mass function

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}, \qquad \text{if } k = 0, 1, 2, \ldots, n. \tag{5.3.22}$$

Since the `cdf` $F(x)$ is the sum of the `pmf` $p(k)$ for $k \leq x$, the code

```
from scipy.stats import binom

n, p = 8, .5
```

```
B = binom(n,p)

for k in range(n+1): print(k, B.pmf(k), B.cdf(k))
```

returns

0	0.003906250000000007	0.00390625
1	0.031249999999999983	0.03515625
2	0.10937500000000004	0.14453125
3	0.21874999999999992	0.36328125
4	0.27343749999999994	0.63671875
5	0.2187499999999999	0.85546875
6	0.10937500000000004	0.96484375
7	0.031249999999999983	0.99609375
8	0.00390625	1.0

Table 5.2 Binomial `pmf` and `cdf` with $n = 8$ tosses.

Since

$$E(S) = E(X_1) + E(X_2) + \cdots + E(X_n) = p + p + \cdots + p = np,$$

the mean of S is np.

Since X_1, X_2 ..., X_n are independent, by (5.3.21), $Var(S) = np(1-p)$. Summarizing,

Mean and Variance of Binomial

If S is binomial with bias p and tosses n, then

$$E(S) = np, \qquad Var(S) = np(1-p). \tag{5.3.23}$$

If $\hat{p}_n$ is the *proportion* of heads, then $\hat{p}_n = S/n$, so

$$E(\hat{p}_n) = p, \qquad Var(\hat{p}_n) = \frac{p(1-p)}{n}. \tag{5.3.24}$$

By the binomial theorem, the moment-generating function is

$$E\left(e^{tS}\right) = \sum_{k=0}^{n} e^{tk} \binom{n}{k} p^k (1-p)^{n-k} = \left(pe^t + 1 - p\right)^n.$$

Then the cumulant-generating function is

$$Z(t) = n \log\left(pe^t + 1 - p\right).$$



A random variable X is *Poisson with parameter* λ if X is discrete and takes on the nonnegative integer values $k = 0, 1, 2, \ldots$ with probability mass function

$$Prob(X = k) = e^{-\lambda} \cdot \frac{\lambda^k}{k!}, \qquad k = 0, 1, 2, \ldots. \tag{5.3.25}$$

Here $\lambda > 0$. From the exponential series (A.3.13),

$$\sum_{k=0}^{\infty} Prob(X = k) = e^{-\lambda} \sum_{k=0}^{\infty} \frac{\lambda^k}{k!} = 1,$$

so the total probability is one. The Python code for a Poisson random variable is

```
from scipy.stats import poisson

lamda = 1
P = poisson(lamda)

for k in range(10): print(k, P.pmf(k), P.cdf(k))
```

Mean and Variance of Poisson

If X is Poisson with parameter λ, then

$$E(X) = \lambda, \qquad Var(X) = \lambda. \tag{5.3.26}$$

This is derived in Exercise **5.3.13**. The Poisson random variable with parameter λ is the limit of the binomial random variable as $n \to \infty$ while keeping the mean $\lambda = np$ fixed (Exercise **5.3.16**).



A continuous random variable X takes on continuous values x with *probability density function* $p(x)$ (`pdf` in Python). Here means are computed by integrals using the fundamental theorem of calculus (A.6.2).

Definition of Expectation: Continuous Case

Let X have probability density function $p(x)$. The *expectation* of X is

$$E(X) = \int x\, p(x)\, dx. \tag{5.3.27}$$

$E(X)$ is also called the *mean* or *average* or *first moment* of X, and is usually denoted μ.

Here the integration is over the entire range of the random variable: If X takes values in the interval $[a, b]$, the integral is from a to b. For a normal random variable, the range is $(-\infty, \infty)$. For a chi-squared random variable, the range is $(0, \infty)$. Below, when we do not specify the limits of integration, the integral is taken over the whole range of X.

More generally, let $f(x)$ be a function. The *mean* of $f(X)$ or *expectation* of $f(X)$ is

$$E(f(X)) = \int f(x)p(x)\,dx. \tag{5.3.28}$$

Since the total probability is one,

$$E(1) = \int p(x)\,dx = 1.$$

This only holds when the integral is over the complete range of X. When this is not so,

$$Prob(a < X < b) = \int_a^b p(x)\,dx$$

is the green area in Figure 5.15. Thus

$$\text{chance} = \text{confidence} = \text{probability} = \text{area}.$$

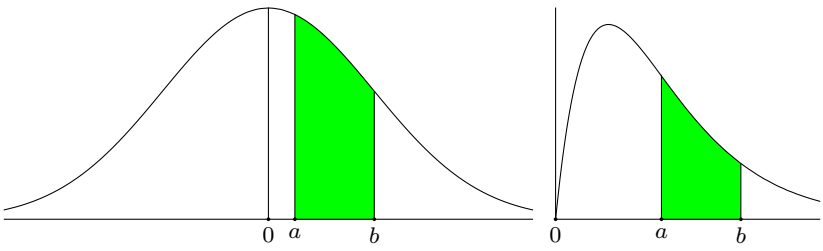


Fig. 5.15 Confidence that X lies in interval $[a, b]$.

When $f(x) = x^2$, the mean of $f(X)$ is the *second moment*

$$E(X^2) = \int x^2 p(x)\,dx.$$

When $f(x) = e^{tx}$, the mean of $f(X)$ is the *moment-generating function*

$$M(t) = E\left(e^{tX}\right) = \int e^{tx} p(x)\, dx.$$

As before, the log of the moment-generating function is the *cumulant-generating function*

$$Z(t) = \log M(t) = \log E\left(e^{tX}\right).$$



The simplest continuous distribution is the *uniform* distribution. A random variable X is distributed uniformly over the interval $[0, 1]$ if (Figure 5.1)

$$Prob(a < X < b) = b - a = \int_a^b 1\, dx, \qquad 0 \le a < b \le 1.$$

Here the probability density function is

$$p(x) = \begin{cases} 1, & a < x < b, \\ 0, & \text{otherwise}, \end{cases}$$

for any interval (a, b) inside $(0, 1)$.

The mean of a uniform random variable is

$$E(X) = \int_0^1 x\, dx.$$

Since

$$F(x) = \frac{1}{2}x^2 \qquad \Longrightarrow \qquad F'(x) = x,$$

by the fundamental theorem of calculus (A.6.2),

$$E(X) = \int_0^1 x\, dx = F(1) - F(0) = \frac{1}{2}.$$

Since $F(x) = x^3/3$ implies $F'(x) = x^2$, the second moment is

$$E(X^2) = \int_0^1 x^2\, dx = F(1) - F(0) = \frac{1}{3}.$$

Hence the variance is

$$Var(X) = E(X^2) - E(X)^2 = \frac{1}{3} - \frac{1}{4} = \frac{1}{12}.$$

The moment-generating function is

$$M(t) = E(e^{tX}) = \int_0^1 e^{tx}\,dx = \frac{e^t - 1}{t}.$$

The cumulative distribution function is

$$F(x) = \begin{cases} 0, & \text{if } x < 0, \\ x, & \text{if } 0 \le x \le 1, \\ 1, & \text{if } x > 1. \end{cases} \tag{5.3.29}$$



More generally, fix an interval $[a, b]$. A random variable X is *uniform on* $[a, b]$ if the probability density function of X is

$$p(x) = \begin{cases} \dfrac{1}{b-a}, & a < x < b, \\ 0, & \text{otherwise}, \end{cases}.$$

For such an X, the mean is

$$\mu = E(X) = \frac{1}{b-a}\int_a^b x\,dx = \frac{1}{b-a}\cdot\frac{b^2-a^2}{2} = \frac{1}{2}(a+b), \tag{5.3.30}$$

and the variance is

$$Var(X) = \frac{1}{b-a}\int_a^b (x-\mu)^2\,dx = \frac{1}{12}(b-a)^2. \tag{5.3.31}$$

In particular, if $[a, b] = [-1, 1]$, then the mean is zero, the variance is $1/3$, and

$$E(f(X)) = \frac{1}{2}\int_{-1}^1 f(x)\,dx.$$



We summarize the differences between discrete and continuous random variables. In both cases, the *cumulative distribution function* is

$$F(x) = Prob(X \le x).$$

When X is discrete,

$$F(x) = \sum_{x_k \le x} p_k.$$

When X is continuous,

$$F(x) = \int_{-\infty}^{x} p(z)\,dz.$$

Then each green area in Figure 5.15 is the difference between two areas,

$$F(b) - F(a).$$

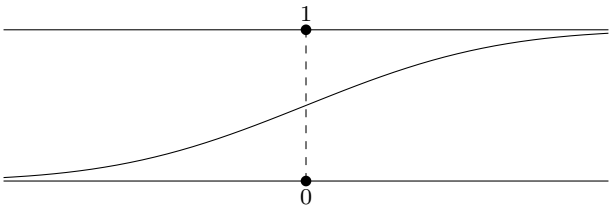


Fig. 5.16 Continuous cumulative distribution function.

When X is discrete, the probability mass function is

$$p(x) = Prob(X = x).$$

When X is continuous, the probability density function $p(x)$ satisfies

$$Prob(a < X < b) = \int_{a}^{b} p(x)\,dx.$$

	discrete	continuous
density	`pmf`	`pdf`
distribution	`cdf`	`cdf`
sum	`cdf(x) = sum([ pmf(k)for k in range(x+1)])`	`cdf(x) = integrate(pdf,x)`
difference	`pmf(k) = cdf(k)-cdf(k-1)`	`pdf(x) = derivative(cdf,x)`

Table 5.3 Densities versus distributions.

For a continuous random variable the probability density function is the derivative of the cumulative distribution function,

$$p(x) = F'(x). \tag{5.3.32}$$

Table 5.3 summarizes the situation. For the distribution on the left in Figure 5.15, the cumulative distribution function is in Figure 5.16.



A *logistic random variable* is a random variable X with cumulative distribution function $\sigma(x)$ (5.2.16). For a logistic random variable, the probability density function is

$$p(x) = \sigma'(x) = \sigma(x)(1 - \sigma(x)), \tag{5.3.33}$$

the mean is zero, and the variance is $\pi^2/3$ (see the exercises).



Let X and Y be independent uniform random variables on $[0, 1]$, and let $Z = \max(X, Y)$. We compute the `pdf` $p(x)$, the `cdf` $F(x)$, and the mean of Z. By definition of $\max(X, Y)$,

$$F(x) = Prob(Z \le x) = Prob(\max(X, Y) \le x)) = Prob(X \le x \text{ and } Y \le x).$$

By independence, for $0 \le x \le 1$, this equals

$$Prob(X \le x)\, Prob(Y \le x) = x^2.$$

Hence

$$F(x) = Prob(\max(X, Y) \le x) = \begin{cases} 0, & \text{if } x < 0, \\ x^2, & \text{if } 0 \le x \le 1, \\ 1, & \text{if } x > 1. \end{cases}$$

From this,

$$p(x) = F'(x) = \begin{cases} 0, & \text{if } x < 0, \\ 2x, & \text{if } 0 \le x \le 1, \\ 0, & \text{if } x > 1. \end{cases}$$

From this, by the FTC (§A.6),

$$E(\max(X, Y)) = \int xp(x)\, dx = \int_0^1 x(2x)\, dx = \frac{2}{3}x^3 \Big|_{x=0}^{x=1} = \frac{2}{3}.$$



Let X have mean μ and variance σ^2, and write

$$Z = \frac{X - \mu}{\sigma}.$$

Then

$$E(Z) = \frac{1}{\sigma}\, E(X - \mu) = \frac{E(X) - \mu}{\sigma} = \frac{\mu - \mu}{\sigma} = 0,$$

and

$$E(Z^2) = \frac{1}{\sigma^2} E((X-\mu)^2) = \frac{\sigma^2}{\sigma^2} = 1.$$

We conclude Z has mean zero and variance one.

A random variable is *standard* if its mean is zero and its variance is one. The variable Z is the *standardization* of X. For example, the standardization of a Bernoulli random variable is

$$Z = \frac{X-p}{\sqrt{p(1-p)}},$$

and the standardization of a uniform random variable on $[0,1]$ is

$$Z = \sqrt{12}(X - 1/2).$$



Definition of Identically Distributed

Random variables X and Y are *identically distributed* if

$$E(X^n) = E(Y^n), \qquad n \geq 1.$$

This is equivalent to X and Y having equal probabilities,

$$Prob(a < X < b) = Prob(a < Y < b),$$

for every interval $[a,b]$, and equivalent to having the same moment-generating functions,

$$M_X(t) = M_Y(t)$$

for every t.

For example, if X and Y satisfy (5.3.14), then X and $2Y-1$ are identically distributed. However, X and $2Y-1$ are independent iff X and Y are independent, which, as we saw above, happens only when (5.3.17) holds.

On the other hand, Let X be any random variable, and let $Y = X$. Then X and Y are identically distributed, but are certainly correlated. So identical distributions does not imply independence, nor vice-versa.

Let X be a random variable. A *simple random sample* of size n is a sequence of random variables $X_1, X_2, \ldots, X_n$ that are independent and identically distributed. We also say the sequence $X_1, X_2, \ldots, X_n$ is an *i.i.d. sequence* (independent identically distributed).

For example, going back to the smartphone example, suppose we select n students at random, where we are allowed to select the same student twice.

We obtain numbers $x_1, x_2, \ldots, x_n$. So the result of a single selection experiment is a sequence of numbers $x_1, x_2, \ldots, x_n$. To make statistical statements about the results, we repeat this experiment many times, and we obtain a sequence of numbers $x_1, x_2, \ldots, x_n$ each time.

This process can be thought of n machines producing $x_1, x_2, \ldots, x_n$ each time, or n random variables X_1, X_2, ..., X_n (Figure 5.17). By making each of the n selections independently, we end up with an i.i.d. sequence, or a simple random sample.

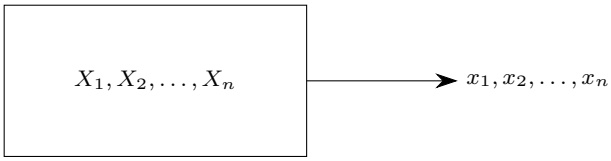


Fig. 5.17 When we sample X_1, X_2, ..., X_n, we get x_1, x_2, ..., x_n.

Let X_1, X_2, ..., X_n be independent and identically distributed, and let μ be their common mean $E(X)$. The *sample mean* is

$$\bar{X}_n = \frac{S_n}{n} = \frac{X_1 + X_2 + \cdots + X_n}{n} = \frac{1}{n}\sum_{k=1}^{n} X_k.$$

Then

$$E(\bar{X}_n) = \frac{1}{n}(E(X_1) + E(X_2) + \cdots + E(X_n)) = \frac{1}{n} \cdot n\mu = \mu.$$

We conclude *the mean of the sample mean equals the population mean.*

Now let σ^2 be the common variance of X_1, X_2, ..., X_n. By (5.3.21), the variance of S_n is $n\sigma^2$, hence the variance of $\bar{X}_n$ is σ^2/n. Summarizing,

Mean and Variance of Sample Mean

If X_1, X_2, ..., X_n are independent and identically distributed, each with mean μ and variance σ^2, then

$$E(\bar{X}_n) = \mu, \qquad Var(\bar{X}_n) = \frac{\sigma^2}{n}, \tag{5.3.34}$$

and

$$\sqrt{n}\left(\frac{\bar{X}_n - \mu}{\sigma}\right) \tag{5.3.35}$$

is standard.

For example, when $X_1, X_2, \ldots, X_n$ are independent and identically distributed according to a random variable X, the proportion $\hat{p}$ of samples (5.3.11) greater than a threshold a has mean $p = Prob(X > a)$, and variance $p(1-p)/n$. It follows that

$$Z = \sqrt{n} \cdot \frac{\hat{p} - p}{\sqrt{p(1-p)}}$$

is standard.

Exercises

Exercise 5.3.1 Let a and b be integers and let X have values a, $a+1$, $a+2$, $\ldots$, $b-1$. Assume the values are equally likely. Use (A.3.5) to show

$$M_X(t) = \frac{1}{b-a} \cdot \frac{e^{tb} - e^{ta}}{e^t - 1}.$$

Exercise 5.3.2 Let A and B be events and let X and Y be the Bernoulli random variables corresponding to A and B (5.3.10). Show that A and B are independent (5.1.9) if and only if X and Y are independent (5.3.18).

Exercise 5.3.3 [30] Let X be a binomial random variable with mean 7 and variance 3.5. What are $Prob(X = 4)$ and $Prob(X > 14)$?

Exercise 5.3.4 The proportion of adults who own a mobile phone in a certain Canadian city is believed to be 90%. Thirty adults are selected at random from the city. Let X be the number of people in the sample who own a mobile phone. What is the distribution of the random variable X?

Exercise 5.3.5 If two random samples of sizes n_1 and n_2 are selected independently from two populations with means μ_1 and μ_2, show the mean of the sample mean difference $\bar{X}_1 - \bar{X}_2$ equals $\mu_1 - \mu_2$. If σ_1 and σ_2 are standard deviations of the two populations, then the standard deviation of $\bar{X}_1 - \bar{X}_2$ equals

$$\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}.$$

Exercise 5.3.6 Check (5.3.30) and (5.3.31).

Exercise 5.3.7 [30] You arrive at the bus stop at 10:00am, knowing the bus will arrive at some time uniformly distributed during the next 30 minutes. What is the probability you have to wait longer than 10 minutes? Given that

the bus hasn't arrived by 10:15am, what is the probability that you'll have to wait at least an additional 10 minutes?

Exercise 5.3.8 If X and Y satisfy (5.3.14), show X and $2Y-1$ are identically distributed for any a, b, c.

Exercise 5.3.9 Let B and G be the number of boys and the number of girls in a randomly selected family with probabilities as in Table 5.1. Are B and G independent? Are they identically distributed?

Exercise 5.3.10 If X and Y satisfy (5.3.14), use Python to verify (5.3.17) and (5.3.19).

Exercise 5.3.11 If X and Y satisfy (5.3.14), compute $Var(X)$ and $Var(Y)$ in terms of a, b, c. What condition on a, b, c maximizes $Var(X)$? What condition on a, b, c maximizes $Var(Y)$?

Exercise 5.3.12 Let X be Poisson with parameter λ. Show the cumulant-generating function is

$$Z(t) = \lambda(e^t - 1).$$

(Use the exponential series (A.3.13).)

Exercise 5.3.13 Let X be Poisson with parameter λ. Show both $E(X)$ and $Var(X)$ equal λ (Use (5.3.12).)

Exercise 5.3.14 Let X and Y be independent Poisson with parameter λ and μ respectively. Show $X+Y$ is Poisson with parameter $\lambda+\mu$.

Exercise 5.3.15 If $X_1, X_2, \ldots, X_n$ are i.i.d. Poisson with parameter λ, show

$$S = X_1 + X_2 + \cdots + X_n$$

is Poisson with parameter $n\lambda$.

Exercise 5.3.16 With $p = \lambda/n$, use the compound-interest formula (A.3.9) to show the binomial `pmf` (5.3.22) converges to the Poisson `pmf` (5.3.25) as $n \to \infty$.

Exercise 5.3.17 The relu(x) function is a common activation function in neural networks (§7.2),

$$\text{relu}(x) = \begin{cases} x & \text{if } x \geq 0, \\ 0 & \text{if } x < 0. \end{cases}$$

If S is Poisson with parameter n, then

$$E\left(\text{relu}(S-n)\right) = e^{-n} \cdot \frac{n^{n+1}}{n!}.$$

(Use Exercise **A.1.2**.)

Exercise 5.3.18 Suppose X is a logistic random variable (5.3.33). Show the probability density function of X is $\sigma(x)(1-\sigma(x))$.

Exercise 5.3.19 Suppose X is a logistic random variable (5.3.33). Show the mean of X is zero.

Exercise 5.3.20 Suppose X is a logistic random variable (5.3.33). Use (A.3.18) with $a = -e^{-x}$ to show the variance of X is

$$4\sum_{n=1}^{\infty}\frac{(-1)^{n-1}}{n^2} = 4\left(1-\frac{1}{4}+\frac{1}{9}-\frac{1}{16}+\dots\right).$$

(This requires knowledge of integration substitution.) Using other tools, it can be shown separately this sum equals $\pi^2/3$ [15].

Exercise 5.3.21 Let $X_1, X_2, \dots, X_n$ be i.i.d. each uniformly distributed on $[0,1]$. Let

$$X_{\max} = \max(X_1, X_2, \dots, X_n).$$

Compute $F(x) = Prob(X_{\max} \le x)$. From that, compute the pdf $p(x)$ of $X_{\max}$, then the mean $E(X_{\max})$. (To evaluate the integral in $E(X_{\max})$, use the FTC.)

Exercise 5.3.22 Let $X_1, X_2, \dots, X_n$ be i.i.d. each uniformly distributed on $[0,1]$. Let

$$X_{\min} = \min(X_1, X_2, \dots, X_n).$$

Compute $1 - F(x) = Prob(X_{\min} > x)$. From that, compute the pdf $p(x)$ of $X_{\min}$, then the mean $E(X_{\min})$. (To evaluate the integral in $E(X_{\min})$, use (5.2.13) with $k = 1$.)

Exercise 5.3.23 A random variable X is *exponential* with parameter $a > 0$ if $Prob(X > x) = e^{-x/a}$. Then $Prob(X \le 0) = 0$, so the values of X are positive. Show that the mean and standard deviation of X are both a.

Exercise 5.3.24 A random variable is *arcsine* if its `pdf` is given by Figure 3.11. Compute the mean and variance of an arcsine random variable. (Substitute $x = (2/\pi)\arcsin(\sqrt{\lambda}/2)$ in the integrals and use the double-angle formula.)

Exercise 5.3.25 For k and n fixed, compute the mean of the conditional probability of a coin's bias p given k heads in n tosses. The answer is not k/n. (Use (5.2.13) with n, k replaced by $n+1$, $k+1$.)

Exercise 5.3.26 Let X and Y be discrete random variables, with values x_1, x_2, ... for X. The *conditional expectation* of X given $Y = y$ is

$$E(X \mid Y = y) = \sum_k x_k \, Prob(X = x_k \mid Y = y).$$

Then the following holds for every value y of Y.

- $E(aX + bZ \mid Y = y) = aE(X \mid Y = y) + bE(Z \mid Y = y)$.
- If X and Y are independent, then $E(X \mid Y = y) = E(X)$.
- $E(f(X,Y) \mid Y = y) = E(f(X,y) \mid Y = y)$.

These properties, also valid for continuous random variables, are used in §7.8.

Exercise 5.3.27 If $f(y) = E(X \mid Y = y)$, we define $\hat{X}$ to be $f(Y)$, and we also write $\hat{X} = E(X \mid Y)$. Continuing the previous exercise, show $E(\hat{X}f(Y)) = E(Xf(Y))$ for every $f(y)$. In particular, $E(\hat{X}) = E(X)$.

Exercise 5.3.28 By definition, $\hat{X}$ is a function of Y. Continuing the previous exercises, show $\hat{X}$ minimizes $E\left((X - f(Y))^2\right)$ over all functions $f(Y)$. Use the same argument that the mean minimizes the MSD (§1.4).

5.4 Normal Distribution

A random variable Z has a *standard normal distribution* or Z *distribution* or *gaussian distribution* if its probability density function is given by the famous formula

$$p(z) = \frac{1}{\sqrt{2\pi}} \cdot e^{-z^2/2}. \tag{5.4.1}$$

This means the normal distribution is continuous and the probability that Z lies in a *small* interval $[a,b]$ is

$$\frac{Prob(a < Z < b)}{b-a} \approx p(z), \qquad a < z < b,$$

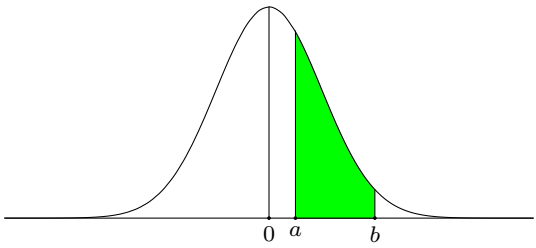


Fig. 5.18 The `pdf` of the standard normal distribution.

When the interval $[a,b]$ is not small, this is not correct. The exact formula for $Prob(a < Z < b)$ is the area under the graph (Figure 5.18). This is obtained by integration (§A.6),

$$Prob(a < Z < b) = \int_a^b p(x)\,dx. \tag{5.4.2}$$

Under this interpretation, this probability corresponds to the area under the graph (Figure 5.18) between the vertical lines at a and at b, and the total area under the graph corresponds to $a = -\infty$ and $b = \infty$.

The normal probability density function is plotted by

```
from scipy.stats import norm as Z
from numpy import *
set_printoptions(legacy = "1.25")

from matplotlib.pyplot import *

# Z defaults to standard normal
# for non-standard, use Z(mu,sdev)

mu, sdev = 0, 1
z = arange(mu-3*sdev,mu+3*sdev,.01)
p = Z.pdf(z)
plot(z,p)

grid()
show()
```



The curious constant $\sqrt{2\pi}$ in (5.4.1) is inserted to make the total area under the graph equal to one. That this is so arises from the fact that 2π is the circumference of the unit circle. Using Python, we see $\sqrt{2\pi}$ is the correct constant, since the code

```
from scipy.integrate import quad

def p(z): return exp(-z**2/2)

a,b = -inf, inf
I = quad(p,a,b)[0] # integral from a to b

allclose(I, sqrt(2*pi))
```

returns `True`.



The mean of Z is

$$E(Z) = \int zp(z)\,dz.$$

More generally, means of $f(Z)$ are

$$E(f(Z)) = \int f(z)p(z)\,dz,$$

with the integral computed using the fundamental theorem of calculus (A.6.2) or Python.

Let $p(z)$ be the probability density function of Z. If we shift the graph of $p(z)$ by horizontally by t, we obtain $p(z-t)$. Since shifting a graph doesn't change the total area under the graph,

$$\int p(z-t)\,dz = 1. \tag{5.4.3}$$

By definition, the moment-generating function of Z is

$$M(t) = E\left(e^{tZ}\right) = \int e^{tz}p(z)\,dz.$$

Using (5.4.3), one can show (Exercise **5.4.11**)

$$M(t) = e^{t^2/2} = \exp(t^2/2). \tag{5.4.4}$$

From this, the cumulant-generating function is $t^2/2$. Using (5.3.12), it follows Z is indeed a standard random variable,

$$E(Z) = 0, \qquad Var(Z) = 1$$

Expand both sides of the definition of $M_Z(t)$ in exponential series. This results in

$$\begin{aligned} 1 + tE(Z) + \frac{t^2}{2!}E(Z^2) + \frac{t^3}{3!}E(Z^3) + \frac{t^4}{4!}E(Z^4) + \dots \\ = 1 + \frac{t^2}{2} + \frac{1}{2!}\left(\frac{t^2}{2}\right)^2 + \frac{1}{3!}\left(\frac{t^2}{2}\right)^3 + \dots. \end{aligned}$$

From this, the odd moments of Z are zero, and the even moments are

$$E(Z^{2n}) = \frac{(2n)!}{2^n n!}, \qquad n = 0, 1, 2, \dots$$

By separating the even and the odd factors, this simplifies to

$$\begin{aligned} E(Z^{2n}) &= \frac{(1 \cdot 3 \cdot 5 \cdot \cdots \cdot (2n-1))(2 \cdot 4 \cdot \cdots \cdot 2n)}{2^n n!} \\ &= \frac{(1 \cdot 3 \cdot 5 \cdot \cdots \cdot (2n-1))2^n n!}{2^n n!} \\ &= 1 \cdot 3 \cdot 5 \cdot \cdots \cdot (2n-1), \qquad n \geq 1. \end{aligned} \tag{5.4.5}$$

For example,

$$E(Z) = 0, E(Z^2) = 1, E(Z^3) = 0, E(Z^4) = 3, E(Z^5) = 0, E(Z^6) = 15.$$

More generally, we say X has a *normal distribution* with parameters μ and σ^2, if its moment-generating function is

$$M_X(t) = E\left(e^{tX}\right) = \exp(\mu t + \sigma^2 t^2/2). \tag{5.4.6}$$

Then its cumulant-generating function is

$$Z_X(t) = \mu t + \frac{1}{2}\sigma^2 t^2,$$

hence its mean and variance are

$$Z'_X(0) = \mu, \qquad Z''_X(0) = \sigma^2.$$

From this, if X is normal with parameters μ and σ^2, then its standardization $Z = (X - \mu)/\sigma$ is standard normal.



We restate the two fundamental results of probability in the language of this section and in terms of limits. We usually deal with limits in an intuitive manner. For additional information on limits, see §A.7.

Given a sample from a random variable X, the population mean is $\mu = E(X)$, and the population variance is $\sigma^2 = Var(X)$. The LLN says for large sample size, the sample mean $\bar{X}$ approximately equals μ.

Law of Large Numbers (LLN)

Let $X_1, X_2, \ldots, X_n$ be independent identically distributed random variables, each with mean μ and variance σ^2, and let

$$\bar{X}_n = \frac{X_1 + X_2 + \cdots + X_n}{n}$$

be the sample mean. Then the event of outcomes where

$$\lim_{n \to \infty} \bar{X}_n = \mu$$

is sure (sure events are defined in §5.1).

In other words, the LLN says the outcomes where the limiting sample mean is not equal to μ form a null event. The event specified in the LLN is *sure*, but not *certain* (see §5.1 and Exercise **5.1.11** for the distinction).

The LLN is qualitative: There is no measure of closeness in the LLN statement. On the other hand, the CLT is more quantitative. The CLT says for large sample size, the sample mean is approximately normal with mean μ and variance σ^2/n. More exactly,

Central Limit Theorem (CLT)

Let

$$\bar{Z}_n = \sqrt{n}\left(\frac{\bar{X}_n - \mu}{\sigma}\right)$$

be the standardized sample mean, and let Z be a standard normal random variable. Then

$$\lim_{n\to\infty} Prob\left(a < \bar{Z}_n < b\right) = Prob(a < Z < b)$$

for every interval $[a, b]$.

An equivalent form of the CLT is

$$\lim_{n\to\infty} E\left(f\left(\bar{Z}_n\right)\right) = E(f(Z)) \tag{5.4.7}$$

for every function $f(x)$.

Let $M_n(t)$ be the moment-generating function of $\bar{Z}_n$. Another equivalent form of the CLT is convergence of the moment-generating functions,

$$\lim_{n\to\infty} M_n(t) = e^{t^2/2}, \tag{5.4.8}$$

for every t.



Toss a coin n times, assume the coin's bias is p, and let S_n be the number of heads. Then, by (5.3.23), S_n is binomial with mean $\mu = np$ and standard deviation $\sigma = \sqrt{np(1-p)}$. By the CLT, S_n is approximately normal with the same mean and variance, so the cumulative distribution function of S_n approximately equals the cumulative distribution function of a normal random variable with the same mean and variance.

The code

```
from scipy.stats import binom, norm

n, p = 100, pi/4
mu = n*p
sigma = sqrt(n*p*(1-p))

B = binom(n,p)
Z = norm(mu,sigma)

x = arange(mu - 2*sigma, mu + 2*sigma, .01)
plot(x, Z.cdf(x), label = "normal approx")
plot(x, B.cdf(x), label = "binomial")

grid()
legend()
show()
```

returns Figure 5.19.

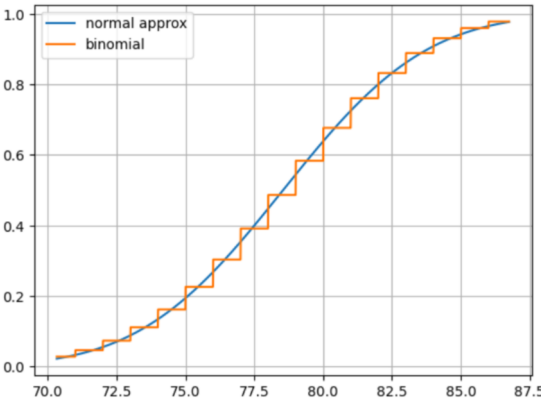


Fig. 5.19 The binomial `cdf` and its CLT normal approximation.

Using the compound-interest formula (A.3.9), it is simple to derive the CLT. We derive the third version (5.4.8) of the CLT. Let $x_1, x_2, \ldots, x_N$ be a scalar dataset, and assume the dataset is standardized. Then its mean and variance are zero and one,

$$\sum_{k=1}^{N} x_k = 0, \qquad \frac{1}{N}\sum_{k=1}^{N} x_k^2 = 1.$$

If the samples of the dataset are equally likely, then sampling the dataset results in a random variable X, with expectations given by (5.3.9). It follows that X is standard, and the moment-generating function of X is

$$E(e^{tX}) = \frac{1}{N}\sum_{k=1}^{N} e^{tx_k}.$$

If $X_1, X_2, \ldots, X_n$ are obtained by repeated sampling of the dataset, then they are i.i.d. following X.

If $\bar{X}_n$ is the sample mean

$$\bar{X}_n = \frac{X_1 + X_2 + \cdots + X_n}{n},$$

then, by (5.3.35), $\bar{Z}_n = \sqrt{n}\bar{X}_n$ is standard.

By independence, the moment-generating function $M_n(t)$ of $\bar{Z}_n$ is the product

$$M_n(t) = E\left(e^{t\sqrt{n}\bar{X}_n}\right) = E\left(e^{t(X_1+X_2+\cdots+X_n)/\sqrt{n}}\right) = \left(E\left(e^{tX/\sqrt{n}}\right)\right)^n.$$

By the exponential series,

$$e^{tX/\sqrt{n}} = 1 + \frac{t}{\sqrt{n}}X + \frac{t^2}{2n}X^2 + \ldots$$

Since the mean and variance of X are zero and 1, taking expectations of both sides,

$$E\left(e^{tX/\sqrt{n}}\right) = 1 + \frac{t^2}{2n} + \ldots.$$

From this,

$$M_n(t) = \left(1 + \frac{t^2}{2n} + \ldots\right)^n.$$

By the compound-interest formula (A.3.9) (the missing terms ... don't affect the result)

$$\lim_{n\to\infty} M_n(t) = e^{t^2/2},$$

which is the moment-generating function of the standard normal distribution.

Even though we couched this derivation in terms of a standardized dataset, it is valid in general. This completes the derivation of the CLT.



The standard normal distribution is symmetric about zero, and has a specific width. Because of the symmetry, a random number Z following this distribution is equally likely to satisfy $Z < 0$ and $Z > 0$, so $Prob(Z < 0) =$

$Prob(Z > 0)$. Since the total area equals 1,

$$Prob(Z < 0) + Prob(Z > 0) = 1,$$

we expect the chance that $Z < 0$ should equal $1/2$. In other words, because of the symmetry of the curve, we expect to be 50% confident that $Z < 0$, or 0 is at the 50-th percentile level. So

$$\text{chance} = \text{confidence} = \text{percentile} = \text{area}$$

To summarize, we expect $Prob(Z < 0) = 1/2$.

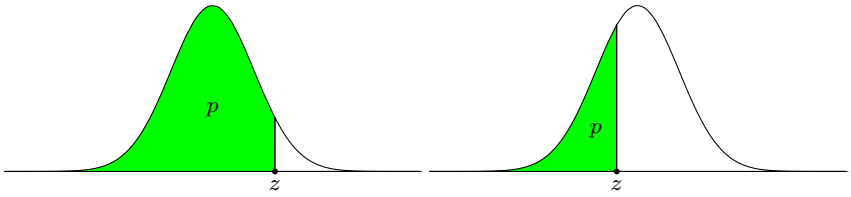


Fig. 5.20 `z = Z.ppf(p)` and `p = Z.cdf(z)`.

When

$$Prob(Z < z) = p,$$

we say z is the *z-score* z corresponding to the *p-value* p. Equivalently, we say our confidence that $Z < z$ is p, or the percentile of z equals $100p$. In Python, the relation between z and p (Figure 5.20) is specified by

```
from scipy.stats import norm as Z

# p = P(Z < z)

z_in = 1.0
p = Z.cdf(z_in)
z_out = Z.ppf(p)

z_in, p, z_out
```

`ppf` is the *percentile point function*, and `cdf` is the *cumulative distribution function*.

In Figure 5.21, the red areas are the *lower tail p-value* $Prob(Z < z)$, the *two-tail p-value* $Prob(|Z| > z)$, and the *upper tail p-value* $Prob(Z > z)$.

To go backward, suppose we are given $Prob(|Z| < z) = p$ and we want to compute the cutoff z. Then $Prob(|Z| > z) = 1 - p$, so $Prob(Z > z) = (1 - p)/2$. This implies

$$Prob(Z < z) = 1 - (1 - p)/2 = (1 + p)/2.$$

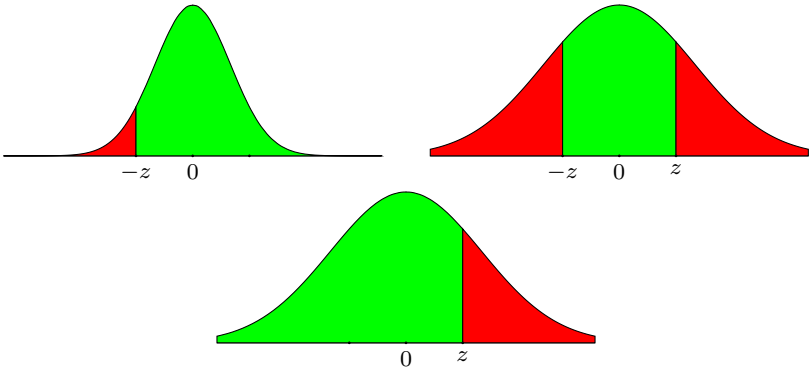


Fig. 5.21 Confidence (green) or significance (red) (lower-tail, two-tail, upper-tail).

By symmetry of the graph, upper-tail and two-tail p-values can be computed from lower tail p-values.

$$Prob(a < Z < b) = Prob(Z < b) - Prob(Z < a),$$

and

$$Prob(|Z| < z) = Prob(-z < Z < z) = Prob(Z < z) - Prob(Z < -z),$$

and

$$Prob(Z > z) = 1 - Prob(Z < z).$$

In Python,

```
# p = P(|Z| < z)

p_in = 0.6826894921370859
z = Z.ppf((1+p_in)/2)
p_out  = Z.cdf(z) - Z.cdf(-z)

p_in, z, p_out
```

Now let's zoom in closer to the graph and mark off z-scores 1, 2, 3 on the horizontal axis to obtain specific colored areas as in Figure 5.22. These areas are governed by the 68-95-99 rule (Table 5.4). Our confidence that $|Z| < 1$ equals the blue area 0.685, our confidence that $|Z| < 2$ equals the sum of the blue plus green areas 0.955, and our confidence that $|Z| < 3$ equals the sum of the blue plus green plus red areas 0.997. This is summarized in Table 5.4.

cutoff	abs confidence	two-tail p-value
z	$1-p$	p
1	.685	.315
2	.955	.045
3	.997	.003

Table 5.4 Cutoffs, confidence levels, p-values.

The possibility $|Z| > 1$ is called a 1-sigma event, $|Z| > 2$ a 2-sigma event, and so on. So a 2-sigma event is 95.5% unlikely, or 4.5% likely. An event is considered *statistically significant* if it's a 2-sigma event or more. In other words, *something is significant if it's unlikely.* A six-sigma event $|Z| > 6$ is two in a billion. You want a plane crash to be six-sigma.

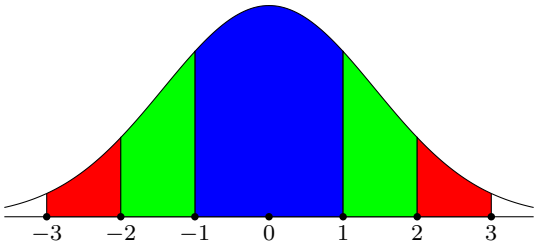


Fig. 5.22 68%, 95%, 99% confidence cutoffs for standard normal.

These terms are defined for two-tail p-values. The same terms may be used for upper-tail or lower tail p-values.

Figure 5.22 is not to scale, because a 1-sigma event should be where the curve inflects from convex to concave (in the figure this happens closer to 2.7). Moreover, according to Table 5.4, the left-over white area should be .03% (3 parts in 10,000), which is not what the figure suggests.



An event is *statistically significant* if its p-value is 5% or less (Table 5.5). For example, $Z > z$ is statistically significant if $Prob(Z > z)$ is .05 or less, which means z is greater than 1.64, $Z < z$ is statistically significant if $Prob(Z < z)$ is .05 or less, which means z is less than -1.64, and $|Z| > z$ is statistically significant if $Prob(|Z| > z)$ is .05 or less, which means $|z|$ is greater than 1.96.

event	type	p-value	z-score
$Z > z$	upper tail	.05	1.64
$Z < z$	lower tail	.05	-1.64
$\|Z\| > z$	two-tail	.05	1.96
$Z > z$	upper tail	.01	2.33
$Z < z$	lower tail	.01	-2.33
$\|Z\| > z$	two-tail	.01	2.56

Table 5.5 p-values at 5% and at 1%.

An event is *highly significant* if its p-value is 1% or less (Table 5.5). For example, $Z > z$ is highly significant if $Prob(Z > z)$ is .01 or less, which means z is greater than 2.33, $Z < z$ is highly significant if $Prob(Z < z)$ is .01 or less, which means z is less than -2.33, and $|Z| > z$ is highly significant if $Prob(|Z| > z)$ is .01 or less, which means $|z|$ is greater than 2.56.

Significant and Highly Significant

An event A is *significant* if $Prob(A) \leq 0.05$. An event A is *highly significant* if $Prob(A) \leq 0.01$.



In general, the normal distribution is not centered at the origin, but elsewhere. We say X is normal with mean μ and standard deviation σ if

$$Z = \frac{X - \mu}{\sigma}$$

is distributed according to a standard normal. We write $N(\mu, \sigma)$ for the normal with mean μ and standard deviation σ. As its name suggests, it is easily checked that such a random variable X has mean μ and standard deviation σ. For the normal distribution with mean μ and standard deviation σ, the cutoffs are as in Figure 5.23. In Python, `norm(mu,sigma)` returns the normal with mean m and standard deviation s.

Here is a sample computation. Let X be a normal random variable with mean μ and standard deviation σ, and suppose $Prob(X < 7) = .15$, and $Prob(X < 19) = .9$. Given this data, we find μ and σ as follows.

With Z as above, we have

$$Prob(Z < (7 - \mu)/\sigma) = .15, \qquad \text{and} \qquad Prob(Z < (19 - \mu)/\sigma) = .9.$$

Also, since Z is standard, we compute

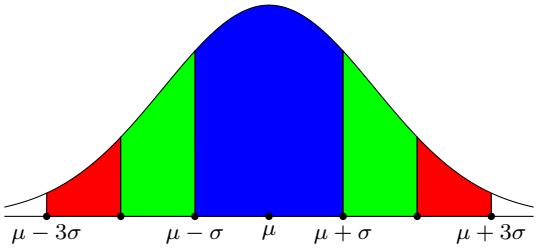


Fig. 5.23 68%, 95%, 99% cutoffs for non-standard normal.

```
a = Z.ppf(.15)
b = Z.ppf(.9)

a, b
```

By definition of `ppf` (see above), we then have

$$a = \frac{7-\mu}{\sigma}, \qquad b = \frac{19-\mu}{\sigma}.$$

These are two equations in two unknowns. Multiplying both equations by σ then subtracting, we obtain μ and σ,

$$\sigma = \frac{19-7}{b-a}, \qquad \mu = 7 - a\sigma.$$

Let $\bar{X}$ be the sample mean

$$\bar{X} = \frac{X_1 + X_2 + \cdots + X_n}{n},$$

drawn from a normally distributed population with mean μ and standard deviation σ. By (5.3.34), the standard deviation of $\bar{X}$ is $\sigma/\sqrt{n}$.

Standard Deviation of Sample Mean is Standard Error

The standard deviation of the sample mean is called the *standard error*. If the samples have standard deviation σ, the standard error is $\sigma/\sqrt{n}$.

To compute probabilities for $\bar{X}$ when X has mean μ and standard deviation σ, standardize $\bar{X}$ by writing

$$Z = \sqrt{n} \cdot \frac{\bar{X} - \mu}{\sigma},$$

then compute standard normal probabilities.



Here are two examples. In the first example, suppose student grades are normally distributed with mean $\mu = 80$ and variance $\sigma^2 = 16$. This says the average of all grades is 80, and the standard deviation is $\sigma = 4$. If a grade is g, the standardized grade is

$$z = \frac{g - \mu}{\sigma} = \frac{g - 80}{4}.$$

A student is picked and their grade was $g = 84$. Is this significant? Is it highly significant? In effect, we are asking, how unlikely is it to obtain such a grade? Remember,

$$\text{significant} = \text{unlikely}$$

Since the standard deviation is 4, the student's z-score is

$$z = \frac{g - 80}{4} = \frac{84 - 80}{4} = 1.$$

What's the upper-tail p-value corresponding to this z? It's

$$Prob(Z > z) = Prob(Z > 1) = \frac{1}{2}\, Prob(|Z| > 1) = .16,$$

or 16%. Since the upper-tail p-value is more than 5%, this student's grade is not significant.

For the second example, suppose a sample of $n = 9$ students are selected and their sample average grade is $\bar{g} = 84$. Is this significant? Is it highly significant? This time we take

$$z = \sqrt{n} \cdot \frac{\bar{g} - 80}{4} = 3\frac{84 - 80}{4} = 3.$$

What's the upper-tail p-value corresponding to this z? It's

$$Prob(Z > z) = Prob(Z > 2.5) = 0.0013,$$

or .13%. Since the upper-tail p-value is less than 1%, yes, this sample average grade is both significant and highly significant.

The same grade, $g = 84$, is not significant for a single student, but is significant for nine students. This is a reflection of the law of large numbers, which says the sample mean approaches the population mean as the sample size grows.



To extract samples from a normal distribution, use a random generator. The code below uses the default random number generator,

```
from numpy.random import default_rng
samples = default_rng().normal

mean, sdev, n = 80, 4, 20
samples(mean,sdev,n)
```

This returns 20 normally distributed numbers, with specified mean and standard deviation.

Be careful to distinguish between

`numpy.random.default_rng.normal` and `scipy.stats.norm`.

The former returns *samples* from a normal distribution, while the latter returns a normal *random variable*. Samples are just numbers; random variables have `cdf`'s, `pmf`'s or `pdf`'s, etc.



Suppose student grades are normally distributed with mean 80 and variance 16. How many students should be sampled so that the chance that at least one student's grade lies below 70 is at least 50%?

To solve this, if p is the chance that a single student has a grade below 70, then $1-p$ is the chance that the student has a grade above 70. If n is the sample size, $(1-p)^n$ is the chance that all sample students have grades above 70. Thus the requested chance is $1-(1-p)^n$. The following code shows the answer is $n = 112$.

```
from scipy.stats import norm as Z

z = 70
mean, sdev = 80, 4
p = Z(mean,sdev).cdf(z)

for n in range(2,200):
   q = 1 - (1-p)**n
   print(n, q)
```



Here is the code for computing tail probabilities for the sample mean $\bar{X}$ drawn from a normally distributed population with mean μ and standard deviation σ. When $n = 1$, this applies to a single normal random variable.

```
########################
# P-values
########################

from numpy import *
from scipy.stats import norm as Z

def pvalue(mean, sdev, n, xbar, type):
   Xbar = Z(mean,sdev/sqrt(n))
   if type == "lower-tail": p = Xbar.cdf(xbar)
   elif type == "upper-tail": p = 1 - Xbar.cdf(xbar)
   elif type == "two-tail": p = 2 *(1 - Xbar.cdf(abs(xbar)))
   else:
      print("What's the tail type (lower-tail, upper-tail,
    ↪ two-tail)?")
      return
   print("sample size: ",n)
   print("mean, sdev, xbar: ",mean, sdev, xbar)
   print("p-value: ",p)
   z = sqrt(n) * (xbar - mean) / sdev
   print("z-score: ",z)

type = "upper-tail"
mean = 80
sdev = 4
n = 1
xbar = 90

pvalue(mean, sdev, n, xbar, type)
```

Exercises

Exercise 5.4.1 Let X be a normal random variable and suppose $Prob(X < 1) = 0.3$, and $Prob(X < 2) = 0.4$ What are the mean and variance of X?

Exercise 5.4.2 [36] Consider a normal distribution curve where the middle 90% of the area under the curve lies above the interval $(4, 18)$. Use this information to find the mean and the standard deviation of the distribution.

Exercise 5.4.3 Let Z be a normal random variable with mean 30.4 and standard deviation of 0.7. What is $Prob(29 < Z < 31.1)$?

Exercise 5.4.4 [36] Consider a normal distribution where the $70th$ percentile is at 11 and the $25th$ percentile is at 2. Find the mean and the standard deviation of the distribution.

Exercise 5.4.5 [36] Let $X_1, X_2, \ldots, X_n$ be an i.i.d. sample each with mean 300 and standard deviation of 21. What is the mean and standard deviation of the sample mean $\bar{X}$?

Exercise 5.4.6 Suppose the scores of students are normally distributed with a mean of 80 and a standard deviation of 4. A sample of size n is selected, and the sample mean is 84. What is the least n for which this is significant? What is the least n for which this is highly significant?

Exercise 5.4.7 [36] A manufacturer says their laser printers' printing speeds are normally distributed with mean 17.63 ppm and standard deviation 4.75 ppm. An i.i.d. sample of $n = 11$ printers is selected, with speeds $X_1, X_2, \ldots, X_n$. What is the probability the sample mean speed $\bar{X}$ is greater than 18.53 ppm?

Exercise 5.4.8 [36] Continuing Exercise **5.4.7**, let Y_k be the Bernoulli random variable corresponding to the event $X_k > 18$ (5.3.10),

$$Y_k = \begin{cases} 1, & \text{if } X_k > 18, \\ 0, & \text{otherwise.} \end{cases}.$$

We count the proportion of printers in the sample having speeds greater than 18 by setting

$$\hat{p} = \frac{Y_1 + Y_2 + \cdots + Y_n}{n}.$$

Compute $E(\hat{p})$ and $Var(\hat{p})$. Use the CLT to compute the probability that more than 50.9% of the printers have speeds greater than 18.

Exercise 5.4.9 [36] The level of nitrogen oxides in the exhaust of a particular car model varies with mean 0.9 grams per mile and standard deviation 0.19 grams per mile . What sample size is needed so that the standard deviation of the sampling distribution is 0.01 grams per mile?

Exercise 5.4.10 [36] The scores of students had a normal distribution with mean $\mu = 559.7$ and standard deviation $\sigma = 28.2$. What is the probability that a single randomly chosen student scores 565 or higher? Now suppose $n = 30$ students are sampled, assume i.i.d. What are the mean and standard deviation of the sample mean score? What z-score corresponds to the mean score of 565? What is the probability that the mean score is 565 or higher?

Exercise 5.4.11 Complete the square in the moment-generating function of the standard normal `pdf` and use (5.4.3) to derive (5.4.4).

Exercise 5.4.12 Let Z be a standard normal random variable, and let $\text{relu}(x)$ be as in Exercise **5.3.17**. Show

$$E(\text{relu}(Z)) = \frac{1}{\sqrt{2\pi}}.$$

(Use the fundamental theorem of calculus (A.6.2).)

Exercise 5.4.13 [31] Let $X_1, X_2, \ldots, X_n$ be i.i.d. Poisson random variables (5.3.25) with parameter 1, let $S = X_1 + X_2 + \cdots + X_n$, and let $\bar{X}_n = S/n$ be the sample mean. Then the mean of $\bar{X}_n$ is 1, and the variance of $\bar{X}_n$ is $1/n$, so

$$\bar{Z}_n = \sqrt{n}(\bar{X}_n - 1) = \frac{S - n}{\sqrt{n}}$$

is standard (5.3.35). By the CLT, with Z standard normal,

$$E(\text{relu}(\bar{Z}_n)) \to E(\text{relu}(Z)), \qquad n \to \infty.$$

Use this to derive Stirling's approximation (A.1.7). (Exercises **5.3.17** and **5.4.12**.)

5.5 Chi-squared Distribution

Let X and Y be independent standard normal random variables. Then (X, Y) is a random point in the plane. What is the probability that the point (X, Y) lies inside a square (Figure 5.24)?

Specifically, assume the square is $|X| \le 1$ and $|Y| \le 1$. Since X and Y independent, the probability (X, Y) lies in the square is

$$\begin{aligned} Prob(|X| \le 1 \text{ and } |Y| \le 1) &= Prob(|X| \le 1)\, Prob(|Y| \le 1) \\ &= Prob(|X| \le 1)^2 = .685^2 = .469. \end{aligned}$$

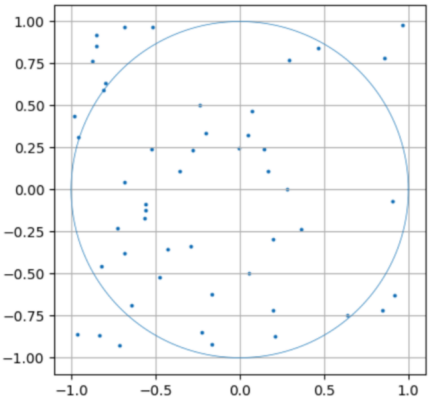


Fig. 5.24 (X, Y) inside the square and inside the disk.

What is the probability (X, Y) lies inside the unit disk,

$$Prob(X^2 + Y^2 \leq 1)?$$

Here the answer is not as straightforward, and leads us to introduce the chi-squared distribution.



A random variable U has a *chi-squared distribution with degree* 1 if

$$M_U(u) = E(e^{uU}) = \frac{1}{\sqrt{1-2u}}.$$

To compute the moments of U, we use the binomial theorem (4.1.25)

$$(1+x)^p = \sum_{n=0}^{\infty} \binom{p}{n} x^n = 1 + px + \binom{p}{2}x^2 + \binom{p}{3}x^3 + \dots$$

to write out $M_U(u)$. Taking $p = -1/2$ and $x = -2u$,

$$\frac{1}{\sqrt{1-2u}} = (1-2u)^{-1/2} = \sum_{n=0}^{\infty} \binom{-1/2}{n} (-2u)^n.$$

Since

$$\frac{1}{\sqrt{1-2u}} = E\left(e^{uU}\right) = \sum_{n=0}^{\infty} \frac{u^n}{n!} E(U^n),$$

comparing coefficients of $u^n/n!$ shows

$$E(U^n) = (-2)^n n! \binom{-1/2}{n}, \qquad n = 0, 1, 2, \dots \tag{5.5.1}$$

Using the definition

$$\binom{p}{n} = \frac{p \cdot (p-1) \cdot \dots \cdot (p-n+1)}{n!},$$

$\binom{p}{n}$ makes sense for fractional p (see (A.2.11)). With this, we have

$$\begin{aligned} E(U^n) &= (-2)^n n! \frac{(-1/2) \cdot (-1/2-1) \cdot \dots \cdot (-1/2-n+1)}{n!} \\ &= 1 \cdot 3 \cdot 5 \cdot 7 \cdot \dots \cdot (2n-1). \end{aligned}$$

But this equals the right side of (5.4.5). Thus the left sides of (5.4.5) and (5.5.1) are equal. This shows

Chi-squared is the Square of Normal

If Z is standard normal, then $U = Z^2$ is chi-squared with degree 1, and $E(U) = 1$, $Var(U) = 2$.



More generally, we say U is *chi-squared with degree d* if

$$U = U_1 + U_2 + \cdots + U_d = Z_1^2 + Z_2^2 + \cdots + Z_d^2, \tag{5.5.2}$$

with independent standard normal $Z_1, Z_2, \ldots, Z_d$.

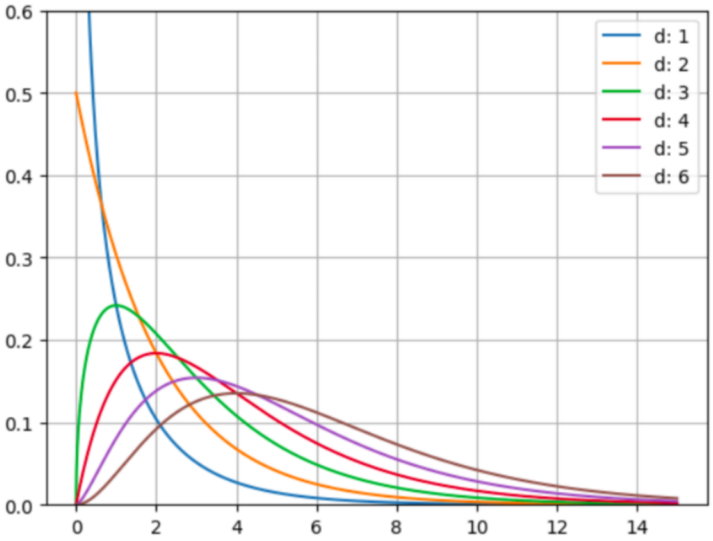


Fig. 5.25 Chi-squared distribution with different degrees.

By independence, the moment-generating functions multiply (§5.3), so the moment-generating function for chi-squared with degree d is

$$M_U(t) = E(e^{tU}) = \frac{1}{(1-2t)^{d/2}}.$$

Going back to the question posed at the beginning of the section, we have X and Y independent standard normal and we want

$$Prob(X^2 + Y^2 \leq 1).$$

If we set $U = X^2 + Y^2$, we want[2] $Prob(U \leq 1)$. Since U is chi-squared with degree $d = 2$, we use `chi2.cdf(u,d)`. Then the code

```
from scipy.stats import chi2 as U

from numpy import *
set_printoptions(legacy = "1.25")

d = 2
u = 1

U(d).cdf(u)
```

returns 0.39.

Figure 5.25 is returned by the code

```
from matplotlib.pyplot import *

u = arange(0,15,.01)

for d in range(1,7):
   p = U(d).pdf(u)
   plot(u,p,label="d: " + str(d))

ylim(ymin=0,ymax=.6)
grid()
legend()
show()
```



Let us compute the mean and variance of a chi-squared U with degree d. When $d = 1$, we already know $E(U) = 1$ and $Var(U) = 2$. In general, by (5.5.2) and (5.3.21),

$$E(U) = \sum_{k=1}^{d} E(Z_k^2) = \sum_{k=1}^{d} 1 = d,$$

and

$$Var(U) = \sum_{k=1}^{d} Var(Z_k^2) = \sum_{k=1}^{d} 2 = 2d.$$

We conclude

[2] Geometrically, the p-value $Prob(U > 1)$ is the probability that a normally distributed point in d-dimensional space is outside the unit sphere.

Mean and Variance of Chi-squared

If U is chi-squared with degree d, the mean and variance of U are

$$E(U) = d, \qquad \text{and} \qquad Var(U) = 2d.$$

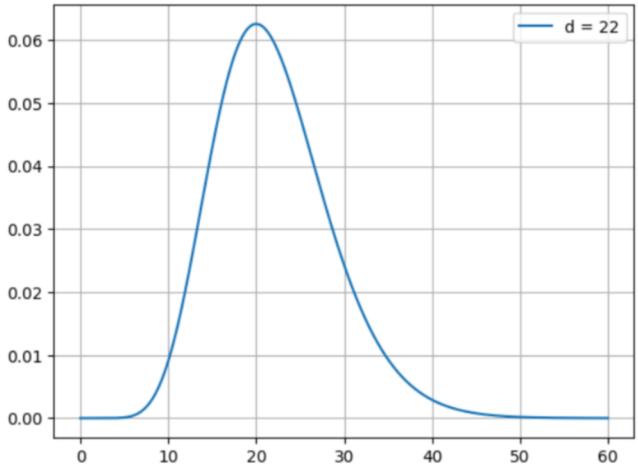


Fig. 5.26 With degree $d \geq 2$, the chi-squared density peaks at $d - 2$.

The peak (maximum likelihood point) in the chi-squared density of degree d is not at the mean d. Using polar coordinates, one can show the peak is at $d - 2$ (Figure 5.26).

Because

$$\frac{1}{(1-2t)^{d/2}} \frac{1}{(1-2t)^{d'/2}} = \frac{1}{(1-2t)^{(d+d')/2}},$$

we obtain

Independent Chi-squared Variables

If U and U' are independent chi-squared with degrees d and d', then $U + U'$ is chi-squared with degree $d + d'$.

To compute distributions for sample variances (below) and chi-squared tests (§6.4), we need to derive chi-squared for correlated normal samples. This is best approached using vector-valued random variables.

A *vector-valued random variable* is a vector $X = (X_1, X_2, \ldots, X_d)$ in $\mathbf{R}^d$ whose components are random variables. A vector-valued random variable is also called a *random vector*. For example, a simple random sample X_1, X_2, $\ldots$, X_n may be collected into a single random vector

$$X = (X_1, X_2, \ldots, X_n)$$

in $\mathbf{R}^n$.

Random vectors have means, variances, moment-generating functions, and cumulant-generating functions, just like scalar-valued random variables. Moreover we can have simple random samples of random vectors X_1, X_2, $\ldots$, X_n.

If X is a random vector in $\mathbf{R}^d$, its *mean* is the vector

$$\mu = E(X) = (E(X_1), E(X_2), \ldots, E(X_d)) = (\mu_1, \mu_2, \ldots, \mu_d).$$

The *variance* of X is the $d \times d$ matrix Q whose (i, j)-th entry is

$$Q_{ij} = E((X_i - \mu_i)(X_j - \mu_j)), \qquad 1 \le i, j \le d.$$

In the notation of §2.2,

$$Q = E((X - \mu) \otimes (X - \mu)).$$

By (2.2.13),

$$w \cdot ((X - \mu) \otimes (X - \mu))w = ((X - \mu) \cdot w)^2,$$

hence

$$w \cdot Qw = E\left(((X - \mu) \cdot w)^2\right). \tag{5.5.3}$$

Thus the variance of a random vector is a nonnegative matrix

A random vector is *standard* if $\mu = 0$ and $Q = I$. If X is standard, then

$$E(X \cdot w) = 0, \qquad Var(X \cdot w) = |w|^2. \tag{5.5.4}$$



In §2.2, we defined the mean and variance of a dataset (2.2.18). Then the mean and variance there is the same as the mean and variance defined here, that of a random variable.

To see this, we must build a random variable X corresponding to a dataset x_1, x_2, $\ldots$, x_N. But this was done in (5.3.9). Thus *every dataset may be interpreted as a random variable.*



In §5.3, we considered i.i.d. sequences of scalar random variables. We can also do the same with random vectors. If X_1, X_2, ..., X_n is an i.i.d. sequence of random vectors, each with mean μ and variance Q, then the same calculation as in §5.3 shows

$$\sqrt{n}\left(\frac{1}{n}\sum_{k=1}^{n} X_k - \mu\right) \tag{5.5.5}$$

has mean zero and variance Q.



A random vector X is *normal with mean μ and variance Q* if for every vector w, the scalar random variable $X \cdot w$ is normal with mean $\mu \cdot w$ and variance $w \cdot Qw$. When $\mu = 0$ and $Q = I$, then X is *standard normal*.

From §5.3, we see

Standard Normal Random Vectors

$Z_1, Z_2, \ldots, Z_d$ is a simple random sample of standard normal random variables iff

$$Z = (Z_1, Z_2, \ldots, Z_d)$$

is a standard normal random vector in $\mathbf{R}^d$.

The central limit theorem remains valid for random vectors: If X_1, X_2, ..., X_n is an i.i.d. sequence of random vectors with mean μ and variance Q, then (5.5.5) is approximately normal, with mean zero and variance Q, for large n.

From (5.5.2),

Uncorrelated Chi-squared

If $Z =$ is a standard normal random vector in $\mathbf{R}^d$, then

$$|Z|^2 \tag{5.5.6}$$

is chi-squared with degree d.

If X is a normal random vector with mean zero and variance Q, then, by definition, $X \cdot w$ is normal with mean zero and $Var(X \cdot w) = w \cdot Qw$. Using (5.4.6) with $t = 1$, $\mu = 0$, and $\sigma^2 = w \cdot Qw$, the moment-generating function of the random vector X is

$$M_X(w) = E\left(e^{w\cdot X}\right) = e^{w\cdot Qw/2}. \tag{5.5.7}$$

In Python, the probability density function of a normal random variable with mean μ and variance Q is

```
from scipy.stats import multivariate_normal as Z

# mu is mean vector array
# Q is variance matrix array

# standard normal
mu = array([0,0])
Q = array([[1,0],[0,1]])

x = array([1.3,-2.1])

# here x.shape == mu.shape
Z.pdf(x, mean = mu, cov = Q)
```

If `x` and `y` are arrays, then `cartesian_product(x,y)` is defined by

```
def cartesian_product(x,y): return dstack(meshgrid(x,y))
```

If `x` and `y` have shapes `(m,)` and `(n,)` then `xy = cartesian_product(x,y)` has shape `(m,n,2)`, with `xy[i,j,:] = array([x[i],y[j]])`.

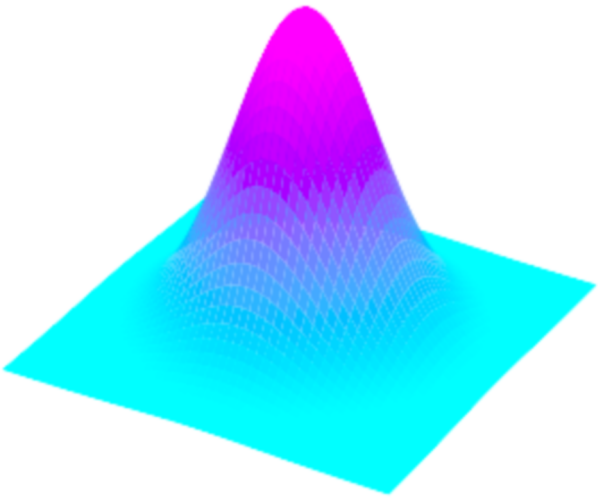


Fig. 5.27 Normal probability density on $\mathbf{R}^2$.

Using this, we can plot the probability density function of a normal random vector in $\mathbf{R}^2$,

```
%matplotlib ipympl

# standard normal
mu = array([0,0])
Q = array([[1,0],[0,1]])

x = arange(-3,3,.01)
y = arange(-3,3,.01)

xy = cartesian_product(x,y)
# last axis of xy is fed into pdf
z = Z(mu,Q).pdf(xy)

ax = axes(projection = '3d')
ax.set_axis_off()
x,y = meshgrid(x,y)
ax.plot_surface(x,y,z, cmap = 'cool')
show()
```

resulting in Figure 5.27.



In §5.3 we studied correlation and independence. We saw how independence implies uncorrelatedness, but not conversely. Now we show that, for normal random vectors, they are in fact the same.

Independence and Correlation

If (X, Y) is a normal random vector, then X and Y are uncorrelated iff X and Y are independent.

Saying (X, Y) is normal is more than just saying X is normal and Y is normal, This is *joint* normality of X and Y. By subtracting their means, we may assume the means of X and Y are zero.

To derive the result, we write down

$$E(X \otimes X) = A, \qquad E(X \otimes Y) = B, \qquad E(Y \otimes Y) = C.$$

Then the variance of (X, Y) is

$$Q = \begin{pmatrix} E(X \otimes X) & E(X \otimes Y) \\ E(Y \otimes X) & E(Y \otimes Y) \end{pmatrix} = \begin{pmatrix} A & B \\ B^t & C \end{pmatrix}$$

. From this, we see X and Y are uncorrelated when $B = 0$.

With $w = (u, v)$, we write

$$w \cdot Qw = \begin{pmatrix} u \\ v \end{pmatrix} \cdot \begin{pmatrix} A & B \\ B^t & C \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = u \cdot Au + u \cdot Bv + v \cdot B^t u + v \cdot Cv.$$

Then

$$M_{X,Y}(w) = E\left(e^{w\cdot(X,Y)}\right) = e^{w\cdot Qw/2} = M_X(u)\, M_Y(v)\, e^{(u\cdot Bv + v\cdot B^t u)/2}.$$

From this, X and Y are independent when $B = 0$. Thus, for normal random vectors, independence and uncorrelatedness are the same.



If Z is a standard normal random vector in $\mathbf{R}^d$, then (5.5.6) we saw $|Z|^2$ is chi-squared with degree d. Now we generalize this result to correlated normal random vectors.

Correlated Chi-squared

Let X be a normal random vector with mean zero and variance Q. Let r be the rank of Q, and let Q^+ be the pseudo-inverse (§2.3) of Q. Then

$$X \cdot Q^+ X \tag{5.5.8}$$

is chi-squared with degree r.

To derive this, we use the eigenvalue decomposition (3.2.6) of Q: There is a square diagonal matrix E and a matrix U satisfying

$$E = U^t Q U, \qquad Q^+ = U E^+ U^t,$$

and

$$E = \begin{pmatrix} \lambda_1 & 0 & 0 & \dots & 0 \\ 0 & \lambda_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & \lambda_r & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \qquad E^+ = \begin{pmatrix} 1/\lambda_1 & 0 & 0 & \dots & 0 \\ 0 & 1/\lambda_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & 1/\lambda_r & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Here r, the number of nonzero eigenvalues of Q, is the rank of Q.

Then, with $Y = U^t X$,

$$X \cdot Q^+ X = X \cdot (U E^+ U^t) X = (U^t X) \cdot E^+ (U^t X) = Y \cdot E^+ Y = \sum_{i=1}^{r} \frac{Y_i^2}{\lambda_i}.$$

Since Y has variance U^tQU (Exercise **5.5.5**), and $U^tQU = E$, $X \cdot Q^+X$ is chi-squared with degree r (Exercise **5.5.6**).

Let μ be a unit vector in $\mathbf{R}^d$, and let $Q = I - \mu \otimes \mu$. Then Q has rank $d-1$ (Exercise **2.9.2**). Suppose X is a normal random vector with mean zero and variance Q. Then

$$E((X \cdot \mu)^2) = \mu \cdot Q\mu = \mu \cdot (I - \mu \otimes \mu)\mu = \mu - (\mu \cdot \mu)\mu = 0,$$

so $X \cdot \mu = 0$.

By Exercise **2.6.7**, $Q^+ = Q$. Since $X \cdot \mu = 0$,

$$X \cdot Q^+X = X \cdot QX = X \cdot (X - (X \cdot \mu)\mu) = |X|^2.$$

We conclude

Singular Chi-squared

Let μ be a unit vector, and let X be a normal random vector with mean zero and variance $I - \mu \otimes \mu$. Then $|X|^2$ is chi-squared with degree $d-1$.

We use the above to derive the distribution of the sample variance. Let $X_1, X_2, \ldots, X_n$ be a random sample, and let $\bar{X}$ be the sample mean,

$$\bar{X} = \frac{X_1 + X_2 + \cdots + X_n}{n}.$$

Let S^2 be the *sample variance*,

$$S^2 = \frac{(X_1 - \bar{X})^2 + (X_2 - \bar{X})^2 + \cdots + (X_n - \bar{X})^2}{n-1}. \tag{5.5.9}$$

Since $(n-1)S^2$ is a sum-of-squares similar to (5.5.2), we expect $(n-1)S^2$ to be chi-squared. In fact this is so, but the degree is $n-1$, not n. We will show

Independence of Sample Mean and Sample Variance

Let $Z_1, Z_2, \ldots, Z_n$ be independent standard normal random variables, let $\bar{Z}$ be the sample mean, and let S^2 be the sample variance.

Then $(n-1)S^2$ is chi-squared with degree $n-1$, and $\bar{Z}$ and S^2 are independent.

To see this, we work with the random vector $Z = (Z_1, Z_2, \ldots, Z_n)$ with mean zero and variance I. Let u and v be vectors in $\mathbf{R}^n$, let

$$\mathbf{1} = (1, 1, \ldots, 1)$$

be in $\mathbf{R}^n$, and let $\mu = \mathbf{1}/\sqrt{n}$. Then μ is a unit vector and

$$Z \cdot \mu = \frac{1}{\sqrt{n}} \sum_{k=1}^{n} Z_k = \sqrt{n}\, \bar{Z}.$$

Since Z_1, Z_2, …, Z_n are i.i.d standard, $Z \cdot \mu = \sqrt{n}\bar{Z}$ is standard.

Now let $U = I - \mu \otimes \mu$ and

$$X = UZ = Z - (Z \cdot \mu)\mu = (Z_1 - \bar{Z}, Z_2 - \bar{Z}, \ldots . Z_n - \bar{Z}).$$

Then the mean of X is zero. Since Z has variance I, by Exercises **2.2.3** and **5.5.5**,

$$Var(X) = U^t I U = U^2 = U = I - \mu \otimes \mu.$$

By singular chi-squared above,

$$(n-1)S^2 = |X|^2$$

is chi-squared with degree $n-1$. Since $Z \cdot \mu$ is standard,

$$E(X(Z \cdot \mu)) = E(Z(Z \cdot \mu)) - E((Z \cdot \mu)^2)\mu = \mu - \mu = 0,$$

so X and $Z \cdot \mu$ are uncorrelated. Since X and $Z \cdot \mu$ are normal, X and $Z \cdot \mu$ are independent. Since $(n-1)S^2 = |X|^2$ and $\sqrt{n}\bar{Z} = Z \cdot \mu$, S^2 and $\bar{Z}$ are independent.

Exercises

Exercise 5.5.1 Let X and Y be independent *uniform* random variables with values in the interval $[-1, 1]$. Then (X, Y) is a point in the square $\{|x| \leq 1, |y| \leq 1\}$. Let

$$B = \begin{cases} 1 & \text{if } X^2 + Y^2 \leq 1, \\ 0 & \text{otherwise} \end{cases}$$

be the Bernoulli variable corresponding to (X, Y) being in the unit disk. Then (5.3.10) the mean $p = E(B)$ equals $Prob\left(X^2 + Y^2 \leq 1\right)$. Show that $p = \pi/4$. (This uses integration with polar coordinates $rdrd\theta$ replacing $dxdy$.)

Exercise 5.5.2 Let $X_1, X_2, \ldots, X_n$, and $Y_1, Y_2, \ldots, Y_n$ be independent i.i.d. samples of *uniform* random variables with values in the interval $[-1, 1]$. Then (X_1, Y_1), (X_2, Y_2), $\ldots$, (X_n, Y_n) are points in the square $\{|x| \leq 1, |y| \leq 1\}$ (Figure 5.24). Let $\hat{p}_n$ be the *proportion* of points (5.3.11) lying in the unit disk $\{x^2 + y^2 \leq 1\}$. Use the LLN to estimate $\hat{p}_n$ for large n.

Exercise 5.5.3 Continuing the previous problem with $n = 20$, use the CLT to estimate the probability that fewer than 50% of the points lie in the unit disk. Is this a 1-sigma event, a 2-sigma event, or a 3-sigma event?

Exercise 5.5.4 Let X be a random vector with mean zero and variance Q Show v is a zero variance direction (§2.5) for Q iff $X \cdot v = 0$.

Exercise 5.5.5 Let μ and Q be the mean and variance of a random d-vector X, and let A be any $N \times d$ matrix. Then AX is a random vector with mean $A\mu$ and variance AQA^t.

Exercise 5.5.6 Let $Y_1, Y_2, \ldots, Y_r$ be independent normal random variables with mean zero and variances $\lambda_1, \lambda_2, \ldots, \lambda_r$. Then

$$\frac{Y_1^2}{\lambda_1} + \frac{Y_2^2}{\lambda_2} + \cdots + \frac{Y_r^2}{\lambda_r}$$

is chi-squared with degree r.

Exercise 5.5.7 If X is a random vector with mean zero and variance Q, then

$$E((X \cdot u)(X \cdot v)) = u \cdot Qv.$$

(Insert $w = u + v$ in (5.5.3).)

Exercise 5.5.8 Assume the classes of the Iris dataset are normally distributed with their means and variances (Exercise **2.2.9**), and assume the classes are equally likely. Using Bayes theorem (5.2.14), write a Python function that returns the probabilities (p_1, p_2, p_3) that a given iris $x = (t_1, t_2, t_3, t_4)$ lies in each of the three classes. Feed your function the 150 samples of the Iris dataset. How many samples are correctly classified?

5.6 Multinomial Probability

Let X be a discrete random variable, with values $i = 1, 2, \ldots, d$, and probabilities $p = (p_1, p_2, \ldots, p_d)$. Then each $p_i \geq 0$ and

$$p_1 + p_2 + \cdots + p_d = 1.$$

Such a vector p is a *probability vector*.

Since the values of X are 1, 2, ..., d, the moment-generating function of X is

$$M(t) = E(e^{tX}) = e^t p_1 + e^{2t} p_2 + \cdots + e^{dt} p_d.$$

A vector $v = (v_1, v_2, \ldots, v_d)$ is *one-hot encoded at slot j* if all components of v are zero except the j-th component. For example, when $d = 3$, the vectors

$$(a, 0, 0), \qquad (0, a, 0), \qquad (0, 0, a)$$

are one-hot encoded.

At the other extreme, a vector $v = (v_1, v_2, \ldots, v_d)$ is *strict* if all components are positive, $v_i > 0$, $i = 1, 2, \ldots, d$.

A useful alternative to $M(t)$ above is to use one-hot encoding and to define a vector-valued random variable $Y = (Y_1, Y_2, \ldots, Y_d)$ by

$$Y_i = \begin{cases} 1, & \text{if } X = i, \\ 0, & \text{otherwise} \end{cases} \qquad i = 1, 2, \ldots, d.$$

This is called *one-hot encoding* since all slots in Y are zero except for one "hot" slot.

For example, suppose X has three values 1, 2, 3, say X is the class of a random sample from the Iris dataset. Then Y is $\mathbf{R}^3$-valued, and we have

$$Y = \begin{cases} (1, 0, 0), & \text{if } X = 1, \\ (0, 1, 0), & \text{if } X = 2, \\ (0, 0, 1), & \text{if } X = 3. \end{cases}$$

With this understood, set $t = (t_1, t_2, t_3)$. Then the moment-generating function of Y is

$$M(t) = E\left(e^{t \cdot Y}\right) = e^{t_1} p_1 + e^{t_2} p_2 + e^{t_3} p_3.$$

More generally, let X have d values. Then with one-hot encoding, the moment-generating function is

$$M(t) = e^{t_1} p_1 + e^{t_2} p_2 + \cdots + e^{t_d} p_d,$$

and the cumulant-generating function is

$$Z(t) = \log\left(e^{t_1} p_1 + e^{t_2} p_2 + \cdots + e^{t_d} p_d\right).$$

In particular, for a fair dice with d sides, the values are equally likely, so the one-hot encoded cumulant-generating function is

$$Z(t) = \log\left(e^{t_1} + e^{t_2} + \cdots + e^{t_d}\right) - \log d. \tag{5.6.1}$$

In this section, we define

$$Z(y) = \log\left(e^{y_1} + e^{y_2} + \cdots + e^{y_d}\right), \tag{5.6.2}$$

so we ignore the constant $\log d$ in (5.6.1). Then Z is a function of d variables $y = (y_1, y_2, \ldots, y_d)$. If we insert $y = 0$, we obtain $Z(0) = \log d$.

Let

$$\mathbf{1} = (1, 1, \ldots, 1).$$

Then

$$p \cdot \mathbf{1} = \sum_{k=1}^{d} p_k = 1.$$

Because

$$Z(y + a\mathbf{1}) = Z(y_1 + a, y_2 + a, \ldots, y_d + a) = Z(y) + a,$$

Z is not bounded below and does not have a minimum.

The *softmax* function is the vector-valued function $q = \sigma(y)$ with components

$$q_k = \sigma_k(y) = \frac{e^{y_k}}{e^{y_1} + e^{y_2} + \cdots + e^{y_d}} = \frac{e^{y_k}}{e^{Z(y)}}, \qquad k = 1, 2, \ldots, d.$$

Thus

$$q = \sigma(y) = e^{-Z(y)}\left(e^{y_1}, e^{y_2}, \ldots, e^{y_d}\right).$$

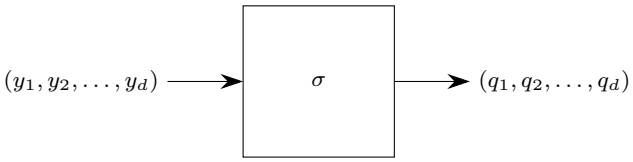


Fig. 5.28 The softmax function takes vectors to probability vectors.

By the chain rule, the gradient of the cumulant-generating function is the softmax function,

$$\nabla Z(y) = \sigma(y). \tag{5.6.3}$$

When $d = 2$, the vector softmax function reduces to the scalar logistic function (5.2.16), since

$$q_1 = \frac{e^{y_1}}{e^{y_1} + e^{y_2}} = \frac{1}{1 + e^{-(y_1 - y_2)}} = \sigma(y_1 - y_2),$$
$$q_2 = \frac{e^{y_2}}{e^{y_1} + e^{y_2}} = \frac{1}{1 + e^{-(y_2 - y_1)}} = \sigma(y_2 - y_1).$$

Because of this, the softmax function is the multinomial analog of the logistic function, and we use the same symbol σ to denote both functions.

```
from numpy import *
set_printoptions(legacy = "1.25")

from scipy.special import softmax

y1 = array([1,2,3])
y2 = array([2,4,6])

p = softmax(y1)
q = softmax(y2)

p, q
```



In §4.5, we studied global minimizers of convex functions. As we saw above, Z does not have a global minimizer over unrestricted z.

Since $\sigma(y) = \nabla Z(y)$, a critical point y^* of Z must satisfy $\sigma(y^*) = 0$. For Z, a critical point cannot be unique, because

$$\sigma(y_1, y_2, \ldots, y_d) = \sigma(y_1 + a, y_2 + a, \ldots, y_d + a),$$

or

$$\sigma(y) = \sigma(y + a\mathbf{1}). \tag{5.6.4}$$

We say a vector y is *centered* if y is orthogonal to $\mathbf{1}$,

$$y \cdot \mathbf{1} = y_1 + y_2 + \cdots + y_d = 0.$$

To guarantee uniqueness of a global minimum of Z, we have to restrict attention to centered vectors y.

Suppose y is centered. Since the exponential function is convex,

$$\frac{e^Z}{d} = \frac{1}{d}\sum_{k=1}^{d} e^{y_k} \geq \exp\left(\frac{1}{d}\sum_{k=1}^{d} y_k\right) = e^0 = 1.$$

This establishes

Restricted Global Minimum of the Cumulant-generating Function

If y is centered, then $Z(y) \geq Z(0) = \log d$.



The inverse of the softmax function is obtained by solving $p = \sigma(y)$ for y, obtaining

$$y_k = Z + \log p_k, \qquad k = 1, 2, \ldots, d. \tag{5.6.5}$$

Define

$$\log p = (\log p_1, \log p_2, \ldots, \log p_d).$$

Then the inverse of $p = \sigma(z)$ is

$$y = Z\mathbf{1} + \log p. \tag{5.6.6}$$



The function

$$I(p) = p \cdot \log p = \sum_{k=1}^{d} p_k \log p_k \tag{5.6.7}$$

is the *absolute information*. Since $0 \leq p_k \leq 1$, $\log p_k \leq 0$, hence $I(p) \leq 0$.

Since log is concave,

$$\sum_{k=1}^{d} p_k \log(e^{y_k}) \leq \log\left(\sum_{k=1}^{d} p_k e^{y_k}\right).$$

This implies

$$\begin{aligned} p \cdot y &= \sum_{k=1}^{d} p_k y_k = \sum_{k=1}^{d} p_k \log(e^{y_k}) \\ &\leq \log\left(\sum_{k=1}^{d} p_k e^{y_k}\right) = \log\left(\sum_{k=1}^{d} e^{y_k + \log p_k}\right) = Z(y + \log p). \end{aligned}$$

Replacing y by $y - \log p$, this establishes

$$I(p) \geq p \cdot y - Z(y). \tag{5.6.8}$$

By (5.6.6), (5.6.8) is an equality when $p = \sigma(y)$. We conclude

Information and Cumulant-Generating Function are Convex Duals

For all probability vectors p,

$$I(p) = \max_y \left(p \cdot y - Z(y)\right).$$

For all vectors y,

$$Z(y) = \max_p \left(p \cdot y - I(p)\right).$$

The second equality follows by switching Z and I in (5.6.8), and repeating the same logic used to derive the first equality.

Inserting $y = 0$ in (5.6.8), we have

Absolute Information is Bounded

For all $p = (p_1, p_2, \ldots, p_d)$,

$$0 \geq I(p) \geq -\log(d). \tag{5.6.9}$$

The *absolute entropy*, the analog of (4.2.1), is

$$H(p) = -I(p) = -\sum_{k=1}^{d} p_k \log(p_k). \tag{5.6.10}$$

Since

$$D^2 I(p) = \operatorname{diag}\left(\frac{1}{p_1}, \frac{1}{p_2}, \ldots, \frac{1}{p_d}\right),$$

we see $I(p)$ is strictly convex, and $H(p)$ is strictly concave.

In Python, the entropy is

```
from scipy.stats import entropy

entropy(p),entropy(q)
```

Here is the multinomial analog of the relation between entropy and coin-tossing (5.2.4). Suppose a dice has d faces.

Entropy and Dice-Rolling

Let $p = (p_1, p_2, \ldots, p_d)$ be a probability vector. Roll a d-faced dice n times, and let $\#_n(p)$ be the number of outcomes where the face-proportions are p. Then

$$\#_n(p) \quad \text{is approximately equal to} \quad e^{nH(p)} \quad \text{for } n \text{ large.}$$

In more detail, using Stirling's approximation (A.1.7), here is the asymptotic equality,

$$\#_n(p) \approx \frac{1}{(2\pi n)^{(d-1)/2}} \cdot \frac{1}{\sqrt{p_1 p_2 \ldots p_d}} \cdot e^{nH(p)}, \qquad \text{for } n \text{ large.}$$

Asymptotic equality means the ratio of the two sides approaches 1 as $n \to \infty$ (A.7).

Now

$$\frac{\partial^2 Z}{\partial y_j \partial y_k} = \frac{\partial \sigma_j}{\partial y_k} = \begin{cases} \sigma_j - \sigma_j \sigma_k, & \text{if } j = k, \\ -\sigma_j \sigma_k, & \text{if } j \neq k. \end{cases}$$

Hence we have

$$D^2 Z(y) = \nabla \sigma(y) = \operatorname{diag}(q) - q \otimes q, \qquad q = \sigma(z). \tag{5.6.11}$$

Let $\bar{v} = v \cdot q = \sum q_k v_k$. Since $Q = D^2 Z(z)$ satisfies

$$v \cdot Qv = \sum_{k=1}^{d} q_k v_k^2 - (v \cdot q)^2 = \sum_{k=1}^{d} q_k (v_k - \bar{v})^2,$$

which is nonnegative, Q is a variance matrix, and Z is convex.

In fact Z is strictly convex along centered directions v, the directions satisfying $v \cdot \mathbf{1} = 0$. If $v \cdot Qv = 0$, then, since $q_k > 0$ for all k, $v = \bar{v}\mathbf{1}$. By Exercise **5.6.1**, if v is centered, this forces $v = 0$. This shows Z is strictly convex along centered directions.

Moreover, Z is proper (4.3.7) on centered vectors. To see this, suppose $y \cdot \mathbf{1} = 0$ and $Z(y) \leq c$. Since $y_j \leq Z(y)$, this implies

$$y_j \leq c, \qquad j = 1, 2, \ldots, d.$$

Given $1 \leq j \leq d$, add the inequalities $y_k \leq c$ over all indices $k \neq j$. Since $y \cdot \mathbf{1} = 0$, $-y_j = \sum_{k \neq j} y_k$. Hence

$$-y_j = \sum_{k \neq j} y_k \leq (d-1)c, \qquad j = 1, 2, \ldots, d.$$

Combining the last two inequalities,

$$|y_j| = \max(y_j, -y_j) \leq (d-1)c, \qquad j = 1, 2, \ldots, d,$$

which implies

$$|y|^2 = \sum_{k=1}^{d} y_k^2 \leq d(d-1)^2 c^2.$$

Setting $C = \sqrt{d}(d-1)c$, we conclude

$$Z(y) \leq c \quad \text{and} \quad y \cdot \mathbf{1} = 0 \qquad \Longrightarrow \qquad |y| \leq C. \tag{5.6.12}$$

By (4.3.7), we have shown

The Cumulant-generating Function is Proper and Strictly Convex

On centered vectors, $Z(y)$ is proper and strictly convex.



Let $p = (p_1, p_2, \ldots, p_d)$ and $q = (q_1, q_2, \ldots, q_d)$ be probability vectors. The *relative information* is

$$I(p, q) = \sum_{k=1}^{d} p_k \log(p_k/q_k). \tag{5.6.13}$$

Let

$$\log q = (\log q_1, \log q_2, \ldots, \log q_d).$$

Then

$$p \cdot \log q = \sum_{k=1}^{d} p_k \log q_k,$$

and

$$I(p, q) = I(p) - p \cdot \log q. \tag{5.6.14}$$

Similarly, the *relative entropy* is

$$H(p, q) = -I(p, q). \tag{5.6.15}$$

In Python, as of this writing, the code

```
from scipy.stats import entropy

entropy(p,q)
```

returns the relative information, not the relative entropy. Always check your Python code's conventions and assumptions. See below for more on this terminology confusion.



Here is the multinomial analog of the relation between relative entropy and coin-tossing (5.2.5). Suppose a dice has d faces, and suppose the probability of rolling the i-th face in a single roll is q_i, $i = 1, 2, \ldots, d$. Then $q = (q_1, q_2, \ldots, q_d)$, the dice's *bias*, is a probability vector, and we expect the long-term *proportions* of faces in n rolls to approximately equal q.

Let $p = (p_1, p_2, \ldots, p_d)$ be another probability vector. Roll a d-faced dice n times, and let $P_n(p, q)$ be the probability that the face-proportions are $p = (p_1, p_2, \ldots, p_d)$, given that the dice's bias is q.

If $p = q$, one's first guess is $P_n(p, p) \approx 1$ for n large. However, this is not correct, because $P_n(p, p)$ is specifying a specific proportion p, predicting specific behavior from the coin tosses. Because this is too specific, it turns out $P_n(p, p) \approx 0$, see Exercise **5.2.7**.

On the other hand, if $p \neq q$, we expect the proportion of faces to equal p to be unlikely. In other words, we expect $P_n(p, q)$ to be small for large n. In fact, when $p \neq q$, it turns out $P_n(p, q) \to 0$ exponentially, as $n \to \infty$. Using (A.1.2), a straightforward calculation results in

Relative Entropy and Dice-Rolling

Assume a d-faced dice's bias is $q = (q_1, q_2, \ldots, q_d)$. Roll the dice n times, and let $P_n(p, q)$ be the probability of obtaining outcomes where the face-proportions are $p = (p_1, p_2, \ldots, p_d)$. Then

$P_n(p, q)$ is approximately equal to $e^{nH(p,q)}$ for n large.

More exactly, using Stirling's approximation (A.1.7), here is the asymptotic equality,

$$P_n(p, q) \approx \frac{1}{(2\pi n)^{(d-1)/2}} \cdot \frac{1}{\sqrt{p_1 p_2 \cdots p_d}} \cdot e^{nH(p,q)}, \qquad \text{for } n \text{ large.}$$



The *relative cumulant-generating function* is

$$Z(y, q) = \log\left(\sum_{k=1}^{d} e^{y_k} q_k\right),$$

As we saw above, this is the one-hot encoded cumulant-generating function of a d-sided dice with side-probabilities $q = (q_1, q_2, \ldots, q_d)$.

If we insert $q_k = \exp(\log(q_k))$ in the definition of $Z(y, q)$, one obtains

$$Z(y, q) = Z(y + \log q).$$

From this, using the change of variable $y' = y + \log q$,

$$\begin{aligned}
\max_y (p \cdot y - Z(y, q)) &= \max_y (p \cdot y - Z(y + \log q)) \\
&= \max_{y'} (p \cdot (y' - \log q) - Z(y')) \\
&= \max_y (p \cdot y - Z(y)) - p \cdot \log q \\
&= I(p) - p \cdot \log q \\
&= I(p, q).
\end{aligned}$$

As before, this shows

Relative Information and Relative Cumulant-generating Function are Convex Duals

For all p and q,

$$I(p, q) = \max_y (p \cdot y - Z(y, q)).$$

For all y and q,

$$Z(y, q) = \max_p (p \cdot y - I(p, q)).$$



In logistic regression (§7.6), the *output* is y, the *computed target* is $q = \sigma(y)$, the *desired target* is p, and the *information error function* is $I(p, \sigma(y))$. To compute the information error, by (5.6.6),

$$q = \sigma(y) \qquad \Longrightarrow \qquad \log q = y - Z(y)\mathbf{1}.$$

By (5.6.14), this yields

Information Error Identity

For all p and all y, if $q = \sigma(y)$, then

$$I(p, q) = I(p) - p \cdot y + Z(y). \tag{5.6.16}$$

This identity is the direct analog of (4.5.23). The identity (4.5.23) is used in linear regression. Similarly, (5.6.16) is used in logistic regression.

From (5.6.11) and (5.6.16),

$$D_y^2 I(p, \sigma(y)) = \operatorname{diag}(q) - q \otimes q, \qquad q = \sigma(y). \tag{5.6.17}$$



The *cross-information* is

$$I_{\text{cross}}(p, q) = -\sum_{k=1}^{d} p_k \log q_k,$$

and the *cross-entropy* is

$$H_{\text{cross}}(p, q) = -I_{\text{cross}}(p, q) = \sum_{k=1}^{d} p_k \log q_k.$$

In the literature, the terminology is backward: the cross-information is usually erroneously called "cross-entropy," see the discussion at the end of the section.

Cross-information and relative information are related by

$$I(p, q) = I(p) + I_{\text{cross}}(p, q).$$

A probability vector $p = (p_1, p_2, \ldots, p_d)$ is *one-hot encoded at slot* j if $p_j = 1$. When p is one-hot encoded at slot j, then $p_k = 0$ for $k \neq j$.

When p is one-hot encoded, then $I(p) = 0$, so

$$I(p, q) = I_{\text{cross}}(p, q), \tag{5.6.18}$$

and, from (5.6.16),

$$I_{\text{cross}}(p, \sigma(y)) = -p \cdot y + Z(y).$$



From (5.6.3) and (5.6.16),

$$\nabla_y I(p, \sigma(y)) = q - p, \qquad q = \sigma(y). \tag{5.6.19}$$

Since $I(p, \sigma(y))$ and $I_{\text{cross}}(p, \sigma(y))$ differ by the constant $I(p)$, we also have

$$\nabla_y I_{\text{cross}}(p, \sigma(y)) = q - p, \qquad q = \sigma(y),$$

so it doesn't matter whether $I(p, q)$ or $I_{\text{cross}}(p, q)$ is used in gradient descent (§7.3). Nevertheless, we stick with $I(p, q)$, because $I(p, q)$ arises naturally as the convex dual of $Z(y, q)$.



Let $q = (q_1, q_2, \ldots, q_d)$ be a probability vector. The *relative softmax function* is

$$\sigma(y, q) = e^{-Z(y,q)} \left(e^{y_1} q_1, e^{y_2} q_2, \ldots, e^{y_d} q_d\right).$$

Then the relative version of (5.6.16) is

$$I(p, \sigma(y, q)) = I(p, q) - p \cdot y + Z(y, q).$$

This is easily checked using the definitions of $I(p, q)$ and $\sigma(y, q)$.



In the literature, in the industry, in Wikipedia, and in Python, the terminology[3] is confused: The relative information $I(p, q)$ is almost always called "relative entropy."

Since the entropy H is the negative of the information I, this is looking at things upside-down. In other settings, $I(p, q)$ is called the "Kullback–Leibler divergence," which is not exactly intuitive terminology.

Also, in machine learning, $I_{\text{cross}}(p, q)$ is called the "cross-entropy," not cross-information, continuing the confusion.

Rubbing salt into the wound, in Python, `scipy.stats.entropy(p)` returns $H(p)$, which is correct, but `scipy.stats.entropy(p,q)` returns $I(p, q)$, which is incorrect, or, at the very least, inconsistent, even within Python.

How does one keep things straight? By remembering that it's convex functions that we like to minimize, not concave functions. In more vivid terms, would you rather ski down a convex slope, or a concave slope?

In machine learning, loss functions are built to be minimized, and information, in any form, is convex, while entropy, in any form, is concave. Table 5.6 summarizes the situation.

[3] The quantities used here are identical to those in the literature, it's only the naming that is confused.

$H = -I$	Information	Entropy
Absolute	$I(p)$	$H(p)$
Cross	$I_{\text{cross}}(p, q)$	$H_{\text{cross}}(p, q)$
Relative	$I(p, q)$	$H(p, q)$
Curvature	Convex	Concave
Error	$I(p, q)$ with $q = \sigma(z)$	

Table 5.6 The third row is the sum of the first and second rows.

Exercises

Exercise 5.6.1 Let v be a centered vector, and suppose v is a multiple of $\mathbf{1}$. Show $v = 0$.

Exercise 5.6.2 Let p be a probability vector, and v be a vector. Then $p+tv$ is a probability vector for all scalar t iff v is centered.

Exercise 5.6.3 Use strict convexity of $-\log x$ to show $I(p, q)$ is positive if $p \neq q$, and $I(p, p) = 0$.

Exercise 5.6.4 Let p be a probability vector. Use (5.6.16) to show the information error $I(p, \sigma(y))$ is strictly convex on centered vectors y.

Exercise 5.6.5 Let p be a strict probability vector with $\min p_i = \epsilon > 0$. Using $p \cdot \mathbf{1} = 1$, show

$$\sum_{j=1}^{d} p_j(Z(y) - y_j) = Z(y) - p \cdot y.$$

Use this and (5.6.16) to show

$$I(p, \sigma(y)) \geq I(p) + \epsilon dZ(y)$$

for y centered. Conclude the information error $I(p, \sigma(y))$ is proper on centered vectors y.

Exercise 5.6.6 Let A be a matrix satisfying $A^t\mathbf{1} = 0$, let b be a strict probability vector with $A^t b$ defined, and let $f(x) = I(b, \sigma(Ax))$. Use Exercise **4.5.21** and the two previous exercises to conclude $f(x)$ is proper and strictly convex on the row space of A. Compute $\nabla f(x)$ to show the minimizer x^* in the row space of A is the unique solution of

$$A^t\sigma(Ax) = A^t b.$$

This is the logistic analog of the regression equation.

Exercise 5.6.7 Let v be a vector, q a probability vector, and $\bar{v} = q \cdot v$. Show

$$\sum_{j=1}^{d} q_j (v_j - \bar{v})^2 \leq \sum_{j=1}^{d} q_j v_j^2 \leq |v|^2.$$

Exercise 5.6.8 Let $q = \sigma(y)$ and $Q = D_y^2 I(p, q)$. Use the previous exercise and (5.6.17) to show

$$v \cdot Qv \leq |v|^2.$$

Exercise 5.6.9 Show

$$\lim_{t \to \infty} \frac{1}{t} Z(ty) = \max(y_1, y_2, \ldots, y_d) = \max y$$

for y centered.

Exercise 5.6.10 Use (5.6.16) and the previous exercise to show

$$\lim_{t \to \infty} \frac{1}{t} I(p, q) = -p \cdot y + \max y, \qquad q = \sigma(ty),$$

for y centered.

Chapter 6
Statistics

6.1 Estimation

In statistics, like any science, we start with a guess or an assumption or hypothesis, then we take a measurement, then we accept or modify our guess/assumption based on the result of the measurement. This is common sense, and applies to everything in life, not just statistics.

For example, suppose you see a sign on campus saying

There is a lecture in room B120.

How can you tell if this is true/correct or not? One approach is to go to room B120 and look. Either there is a lecture or there isn't. Problem solved.

But then someone might object, saying, wait, what if there is a lecture in room B120 tomorrow? To address this, you go every day to room B120 and check, for 100 days. You find out that in 85 of the 100 days, there is a lecture, and in 15 days, there is none. Based on this, you can say you are 85% confident there is a lecture there. Of course, you can never be sure, it depends on which day you checked, you can only provide a confidence level. Nevertheless, this kind of thinking allows us to quantify the probability that our hypothesis is correct.

In general, the measurement is significant if it is *unlikely*. When we obtain a significant measurement, then we are likely to reject our guess/assumption. So

$$\text{significance} = 1 - \text{confidence}.$$

In practice, our guess/assumption allows us to calculate a p-value, which is the probability that the measurement is *not* consistent with our assumption. In the above scenario, the p-value is .15, determined by repeatedly sampling the room.

This is what statistics is about, summarized in Figure 6.1. The details may be more or less complicated depending on the problem situation or setup, but this is the central idea.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

O. Hijab, *Math for Data Science*, https://doi.org/10.1007/978-3-031-89707-8_6

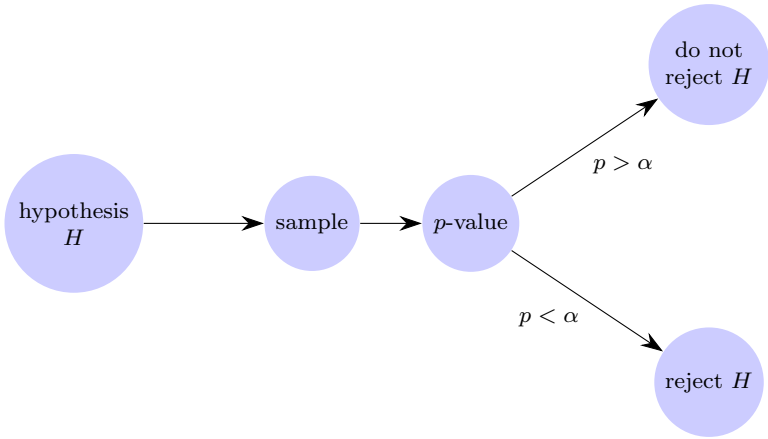


Fig. 6.1 Statistics flowchart: p-value p and significance α.



Here is a geometric example. Grab two vectors in three dimensions at random and measure the angle between them. Is there any pattern to the answer? Doing so twenty times, we see the answer is no, the resulting angle can be any angle. Now grab two vectors in 784 dimensions at random. Then, as we shall see, there is a pattern.

The *null hypothesis* and the *alternate hypothesis* are

- H_0: The angle between two randomly selected vectors in 784 dimensions is approximately 90°
- H_a: The angle between two randomly selected vectors in 784 dimensions is approximately 60°.

To test these hypotheses, we run the code

```
from numpy import *
set_printoptions(legacy = "1.25")

def Angle(u,v):
   a = dot(u,v)
   b = dot(u,u)
   c = dot(v,v)
   theta = arccos(a / sqrt(b*c))
   return degrees(theta)

from numpy.random import default_rng
samples = default_rng().normal

mu, sigma, d = 0, 1, 784
```

```
for _ in range(20):
   u = samples(mu,sigma,d)
   v = samples(mu,sigma,d)
   print(Angle(u,v))
```

to randomly select u, v twenty times. Here `samples(mu,sigma,d)` returns a vector in $\mathbf{R}^d$ whose components are selected independently and randomly according to a normal distribution with mean μ and standard deviation σ. This code returns (since the selection is random, your numbers will differ)

```
86.27806537791886
87.91436653824776
93.00098725550777
92.73766421951748
90.005139015804
87.99643434444482
89.77813370637857
96.09801014394806
90.07032573539982
89.37679070400239
91.3405728939376
86.49851399221568
87.12755619082597
88.87980905998855
89.80377324818076
91.3006921339982
91.43977096117017
88.52516224405458
86.89606919838387
90.49100744167357
```

and we see strong evidence supporting H_0.

On the other hand, run the code

```
from numpy.random import default_rng
samples = default_rng().binomial

n, p, d = 1, .5, 784 # one coin toss
# n, p, d = 3, .5, 784 # three coin tosses

for _ in range(20):
   u = samples(n,p,d)
   v = samples(n,p,d)
   print(Angle(u,v))
```

to randomly select u, v twenty times. Here `samples(n,p,d)` returns a vector in $\mathbf{R}^d$ whose components are selected independently and randomly according to the number of heads in n tosses of a coin with bias p. This code returns

```
59.43464627897324
59.14345748418916
60.31453922165891
60.38024365702492
59.24709660805488
59.27165957992343
61.21424657806321
60.55756381536082
61.59468919876665
61.33296028237481
60.03925473033243
60.25732069941224
61.77018692842784
60.672901794058326
59.628519516164666
59.41272458020638
58.43172340007064
59.863796136907744
59.45156367988921
59.95835532791699
```

and we see strong evidence supporting H_1.



The difference between the two scenarios is the distribution. In the first scenario, the components are distributed according to a standard normal. In the second scenario, the components are distributed according to one or three fair coin tosses. To see how the distribution affects things, we bring in the law of large numbers, which is discussed in §5.3.

Let $X_1, X_2, \ldots, X_d$ be a simple random sample from some population, and let μ be the population mean. Recall this means $X_1, X_2, \ldots, X_d$ are i.i.d. random variables, with $\mu = E(X)$. The sample mean is

$$\bar{X} = \frac{X_1 + X_2 + \cdots + X_d}{d}.$$

Law of Large Numbers

For large sample size d, the sample mean $\bar{X}$ approximately equals the population mean μ, $\bar{X} \approx \mu$.

We use the law of large numbers to explain the closeness of the vector angles to specific values.

Assume $u = (x_1, x_2, \ldots, x_d)$, and $v = (y_1, y_2, \ldots, y_d)$ where all components are selected independently of each other, and each is selected according to the same distribution.

Let $U = (X_1, X_2, \ldots, X_d)$, $V = (Y_1, Y_2, \ldots, Y_d)$, be the corresponding random variables. Then X_1, X_2, …, X_d and Y_1, Y_2, …, Y_d are independent and identically distributed (i.i.d.), with population mean $E(X_1) = E(Y_1)$.

From this, X_1Y_1, X_2Y_2, …, X_dY_d are i.i.d. random variables with population mean $E(X_1Y_1)$. By the law of large numbers,[1]

$$\frac{X_1Y_1 + X_2Y_2 + \cdots + X_dY_d}{d} \approx E(X_1Y_1),$$

so

$$U \cdot V = X_1Y_1 + X_2Y_2 + \cdots + X_dY_d \approx d\,E(X_1Y_1).$$

Similarly, $U \cdot U \approx d\,E(X_1^2)$ and $V \cdot V \approx d\,E(Y_1^2)$. Hence (check that the d's cancel)

$$\cos(U, V) = \frac{U \cdot V}{\sqrt{(U \cdot U)(V \cdot V)}} \approx \frac{E(X_1Y_1)}{\sqrt{E(X_1^2)E(Y_1^2)}}.$$

Since X_1 and Y_1 are independent with mean μ and variance σ^2,

$$E(X_1Y_1) = E(X_1)E(Y_1) = \mu^2, \qquad E(X_1^2) = \mu^2 + \sigma^2, \qquad E(Y_1^2) = \mu^2 + \sigma^2.$$

If θ is the angle between U and V, we conclude

$$\cos(\theta) = \frac{U \cdot V}{\sqrt{(U \cdot U)(V \cdot V)}} \approx \frac{\mu^2}{\mu^2 + \sigma^2}.$$

When the distribution is standard normal, $\mu = 0$, so the angle is approximately 90°. When the distribution is Bernoulli with parameter p,

$$\frac{\mu^2}{\mu^2 + \sigma^2} = \frac{p^2}{p^2 + p(1-p)} = p.$$

For $p = .5$, this results in an angle of 60°.

The general result is

Random Vectors in High Dimensions

Let U and V be two vectors selected randomly. Assume the components of U and V are independent and identically distributed with mean μ and variance σ^2. Let θ be the angle between them. When the vector dimension is high,

$$\cos(\theta) \qquad \text{is approximately} \qquad \frac{\mu^2}{\mu^2 + \sigma^2}.$$

[1] $\approx$ means the ratio of the two sides approaches 1 for large n, see §A.7.

6.2 Z-test

Suppose we want to estimate the proportion of American college students who have a smart phone. Instead of asking every student, we take a sample and make an estimate based on the sample.

The *population proportion* p is the actual proportion of students that in fact have a smart phone. Then $0 < p < 1$. Pick a student, and let

$$X = \begin{cases} 1, & \text{if the student has a smartphone,} \\ 0, & \text{if not.} \end{cases}$$

Then X is a Bernoulli random variable with mean p.

For example, suppose the population proportion of students that have a smartphone is $p = .7$, and we sample $n = 25$ students, obtaining a sample proportion $\bar{X}$. If we repeat the sampling $N = 1000$ times, we will obtain 1000 values for $\bar{X}$. Figure 6.2 displays the resulting histogram of $\bar{X}$ values. Here is the code

```
from numpy import *
set_printoptions(legacy = "1.25")

from matplotlib.pyplot import *

from numpy.random import default_rng
samples = default_rng().binomial

n, p, N = 25, .7, 1000
v = samples(n,p,N)/n

hist(v,edgecolor = 'Black')
show()
```

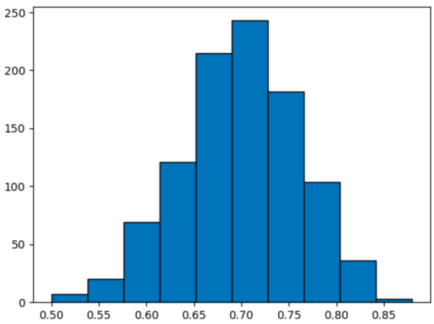


Fig. 6.2 Histogram of sampling $n = 25$ students, repeated $N = 1000$ times.

Let $X_1, X_2, \ldots, X_n$ be a simple random sample of size n. This means n students were selected randomly and independently and whether or not they had smartphones was recorded in the variables $X_1, X_2, \ldots, X_n$. Each of these variables equals one or zero with probability p or $1 - p$.

The sample mean (§5.3) is

$$\bar{X} = \frac{X_1 + X_2 + \cdots + X_n}{n} = \frac{1}{n} \sum_{k=1}^{n} X_k.$$

Because each X_k is 0 or 1, this is the *sample proportion* of the students in the sample that have smartphones. Like p, $\bar{X}$ is also between zero and one.

Because the samples vary, it is impossible to make absolute statements about the population. Instead, as we see below, the best we can do is make statements that come with a *confidence level*. Confidence levels are expressed as percentages, such as a 95% confidence level, or as a proportion, such as a .95 confidence level.

Often levels are expressed as *significance levels*. The *significance level* is the corresponding tail probability, so

$$\text{significance level} = 1 - \text{confidence level}.$$

A confidence level of zero indicates that we have *no faith at all* that selecting another sample will give similar results, while a confidence level of 1 indicates that we have *no doubt at all* that selecting another sample will give similar results.

When we say p is within $\bar{X} \pm \epsilon$, or

$$|p - \bar{X}| < \epsilon,$$

we call ϵ the *margin of error.*. The interval

$$(L, U) = (\bar{X} - \epsilon, \bar{X} + \epsilon)$$

is a *confidence interval*.

With the above setup, we have the population proportion p, and the four sample characteristics

- sample size n
- sample proportion $\bar{X}$,
- margin of error ϵ,
- confidence level α.

Suppose we do not know p, but we know n and $\bar{X}$. We say the margin of error is ϵ, at confidence level α, if

$$Prob(|p - \bar{X}| < \epsilon) = \alpha.$$

Here are some natural questions:

1. Given a sample of size $n = 20$ and sample proportion $\bar{X} = .7$, what can we say about the margin of error ϵ with confidence $\alpha = .95$?
2. Given a sample proportion $\bar{X} = .7$, what sample size n should we take to obtain a margin of error $\epsilon = .15$ with confidence $\alpha = .95$?
3. Given a sample proportion $\bar{X} = .7$, what sample size n should we take to obtain a margin of error $\epsilon = .15$ with confidence $\alpha = .99$?
4. Given a sample of size $n = 20$ and sample proportion $\bar{X} = .7$, with what confidence level α is the margin of error $\epsilon = .1$?

The answers are at the end of the section.



Suppose each X_k in the sample $X_1, X_2, \ldots, X_n$ has mean μ and standard deviation σ. From §5.3, we know the mean and standard deviation of $\bar{X}$ are μ and $\sigma/\sqrt{n}$. In particular, when $X_1, X_2, \ldots, X_n$ is a Bernoulli sample, the mean and variance of the sample proportion $\bar{X}$ are p and $p(1-p)/n$.

Therefore, the mean and variance of the standardized random variable

$$Z = \sqrt{n}\frac{\bar{X} - p}{\sqrt{p(1-p)}}$$

are zero and one.

Returning to our smartphone question, how close is the sample mean $\bar{X}$ to the population mean $E(X) = p$? Remember, both $\bar{X}$ and p are between 0 and 1. More specifically, given a *margin of error* ϵ, we want to compute the confidence level

$$Prob\left(\left|\bar{X} - p\right| < \epsilon\right).$$

This corresponds to the confidence interval

$$L, U = \bar{X} - \epsilon, \bar{X} + \epsilon.$$

The key result is the central limit theorem (§5.3): Z is approximately normal. How large should the sample size n be in order to apply the central limit theorem? When we have *success-failure condition*

$$np \geq 10, \qquad n(1-p) \geq 10.$$

For example, $p = .7$ and $n = 50$ satisfies the success-failure condition.

Let α be the two-tail significance level, say $\alpha = .05$. Assuming Z is exactly normal, let z^* be the z-score corresponding to significance α,

$$Prob(|Z| > z^*) = \alpha.$$

Let $\sigma/\sqrt{n}$ be the standard error. By the central limit theorem,

$$\alpha \approx Prob\left(\frac{|\bar{X}-p|}{\sqrt{p(1-p)}} > \frac{z^*}{\sqrt{n}}\right).$$

To compute the confidence interval (L, U), we solve

$$\frac{|\bar{X}-p|}{\sqrt{p(1-p)}} = \frac{z^*}{\sqrt{n}} \tag{6.2.1}$$

for p. But (6.2.1) may be rewritten as a quadratic equation in p, leading to the approximate solution

$$L, U = \bar{X} \pm \epsilon = \bar{X} \pm \frac{z^*}{\sqrt{n}} \cdot \sqrt{\bar{X}(1-\bar{X})}.$$

From here we obtain the margin of error

$$\epsilon = \frac{z^*}{\sqrt{n}} \cdot \sqrt{\bar{X}(1-\bar{X})}.$$



More generally, let z^* be the z-score corresponding to significance level α, so

```
from scipy.stats import norm as Z

alpha = .3
zstar = Z.ppf(alpha)     # lower-tail, zstar < 0
zstar = Z.ppf(1-alpha)   # upper-tail, zstar > 0
zstar = Z.ppf(1-alpha/2) # two-tail,   zstar > 0
```

Given a population with known standard deviation σ, sample size n, and sample mean $\bar{X}$, the *margin of error* is

$$\epsilon = z^* \cdot \frac{\sigma}{\sqrt{n}},$$

and the intervals

$$(L, U) = \begin{cases} (\bar{X}-\epsilon, \bar{X}), & \text{lower-tail,} \\ (\bar{X}, \bar{X}+\epsilon), & \text{upper-tail,} \\ (\bar{X}-\epsilon, \bar{X}+\epsilon), & \text{two-tail,} \end{cases}$$

are the *confidence intervals* at significance level α. When not specified, a confidence interval is usually taken to be two-tail.

In the Python code below, instead of working with the standardized statistic Z, we work directly with the $\bar{X}$. When σ is not known, we have to replace the normal distribution by the t distribution (§6.3).

```
##########################
# Confidence Interval - Z
##########################

# significance level alpha

def confidence_interval(xbar, sdev, n, alpha, type):
   Xbar = Z(xbar,sdev/sqrt(n))
   if type == "two-tail":
      U = Xbar.ppf(1-alpha/2)
      L = Xbar.ppf(alpha/2)
   elif type == "upper-tail":
      U = Xbar.ppf(1-alpha)
      L = xbar
   elif type == "lower-tail":
      L = Xbar.ppf(alpha)
      U = xbar
   else: print("what's the test type?"); return
   return L, U

# when X is not Bernoulli 0,1,
# Z-test assumes sdev is given
# when X is Bernoulli, sdev = sqrt(xbar*(1-xbar))

alpha = .02
sdev = 228
n = 35
xbar = 95
type = "two-tail"

L, U = confidence_interval(xbar, sdev, n, alpha, type)

print("type: ", type)
print("significance, sdev, n, xbar: ", alpha, sdev, n, xbar)
print("lower, upper: ",L, U)
```

Now we can answer the questions posed at the start of the section. Here are the answers.

1. When $n = 20$, $\alpha = .95$, and $\bar{X} = .7$, we have $[L, U] = [.5, .9]$, so $\epsilon = .2$.
2. When $\bar{X} = .7$, $\alpha = .95$, and $\epsilon = .15$, we run `confidence_interval` for $15 \leq n \leq 40$, and select the least n for which $\epsilon < .15$. We obtain $n = 36$.
3. When $\bar{X} = .7$, $\alpha = .99$, and $\epsilon = .15$, we run `confidence_interval` for $1 \leq n \leq 100$, and select the least n for which $\epsilon < .15$. We obtain $n = 62$.
4. When $\bar{X} = .7$, $n = 20$, and $\epsilon = .1$, we have

$$z^* = \frac{\epsilon\sqrt{n}}{\sigma} = .976.$$

Since $Prob(Z > z^*) = .165$, the confidence level is $1 - 2 * .165 = .68$ or 68%.



The speed limit on a highway is $\mu_0 = 120$. Ten automatic speed cameras are installed along a stretch of the highway to measure passing vehicles speeds. Because the cameras aren't perfect, the average speed $\bar{X}$ measured by the cameras may not equal a vehicle's true speed μ. As a consequence, some drivers who were driving at the speed limit may be fined. These drivers are *false positives*.

Suppose the distribution of a vehicle's measured speed is normal with standard deviation 2. What measured speed cutoff μ^* should the authorities use to keep false positives below 1%? Here we are asked for the upper-tail confidence interval $(L, U) = (\mu_0, \mu^*)$ at significance level .01. A driver will be fined if their average measured speed $\bar{X}$ is higher than μ^*.

Using the above code, the cutoff μ^* equals 121.47.



One use of confidence intervals is *hypothesis testing*. Here we have two hypotheses, a null hypothesis and an alternate hypothesis. In the above setting where we are estimating a population parameter μ, the *null hypothesis* is that μ equals a certain value μ_0, and the *alternate hypothesis* is that μ is not equal to μ_0. Hypothesis testing is of three types, depending on the alternate hypothesis: $\mu \neq \mu_0$, $\mu > \mu_)$, $\mu < \mu_0$. These are two-tail, lower-tail, and upper-tail hypotheses.

- H_0: $\mu = \mu_0$
- H_a: $\mu \neq \mu_0$ or $\mu < \mu_0$ or $\mu > \mu_0$.

For example, going back to our smartphone setup, if we sample $n = 20$ students, obtaining a mean $\bar{x} = .7$, then $\sigma = \sqrt{\bar{x}(1 - \bar{x})} = .46$, and the two-tail 5% confidence interval is then $[.5, .9]$. If μ_0 lies outside the confidence interval, we reject H_0 and accept H_a, at the 5% level. Otherwise, if μ_0 lies within the interval, we do not reject H_0.

Suppose 35 people are randomly selected and the accuracy of their wristwatches is checked, with positive errors representing watches that are ahead of the correct time and negative errors representing watches that are behind the correct time. The sample has a mean of 95 seconds and a population standard deviation of 228 seconds. At the 2% significance, can we claim the population mean is $\mu_0 = 0$?

Here

- H_0: $\mu = 0$

- H_a: $\mu \neq 0$.

Here the significance level is $\alpha = .02$ and $\mu_0 = 0$. To decide whether to reject H_0 or not, compute the *standardized test statistic*

$$z = \sqrt{n} \cdot \frac{\bar{x} - \mu_0}{\sigma} = 2.465.$$

Since z is a sample from an approximately normal distribution Z, the p-value

$$p = Prob(|Z| > z) = .0137.$$

On the other hand, the z-score corresponding to the requested significance level is $z^* = 2.326$, since

$$Prob(|Z| > 2.326) = .02.$$

Since p is less than α, or equivalently, since $|z| > z^*$, we reject H_0. In other words, when the p-value is smaller than the significance level, it is *more* significant, and we reject H_0.

Equivalently, the 98% confidence interval is

$$(\bar{x} - \epsilon, \bar{x} + \epsilon) = (5.3, 184.6)\,.$$

Since $\mu_0 = 0$ is outside this interval, we reject H_0.



Hypothesis Testing

There are three types of alternative hypotheses H_a:

$$\mu < \mu_0, \qquad \mu > \mu_0, \qquad \mu \neq \mu_0.$$

These are lower-tail, upper-tail, and two-tail tests. In every case, we have a sample of size n, a statistic $\bar{x}$, a standard deviation σ, a standardized statistic

$$z = \sqrt{n} \cdot \frac{\bar{x} - \mu_0}{\sigma},$$

a significance level α, the p-value

$$p = Prob(Z < z), \qquad p = Prob(Z > z), \qquad p = Prob(|Z| > z),$$

and the critical cutoff z^*,

$$Prob(Z < z^*) = \alpha, \qquad Prob(Z > z^*) = \alpha, \qquad Prob(|Z| > z^*) = \alpha.$$

Then we reject H_0 whenever z is *more* significant than z^*, which is the same as saying whenever the p-value p is less than the significance level α.



In the Python code below, instead of working with the standardized statistic Z, we work directly with $\bar{X}$, which is normally distributed with mean μ_0 and standard deviation $\sigma/\sqrt{n}$.

```
###################
# Hypothesis Z-test
###################

# significance level alpha

def ztest(mu0, sdev, n, xbar, type):
   Xbar = Z(mu0,sdev/sqrt(n))
   print("mu0, sdev, n, xbar: ", mu0, sdev, n, xbar)
   if type == "lower-tail": p = Xbar.cdf(xbar)
   elif type == "upper-tail": p = 1 - Xbar.cdf(xbar)
   elif type == "two-tail": p = 2 * (1 - Xbar.cdf(abs(xbar)))
   print("type: ",type)
   print("pvalue: ",p)
   if p < alpha: print("reject H0")
   else: print("do not reject H0")

xbar = 122
n = 10
type = "upper-tail"
mu0 = 120
sdev = 2
alpha = .01

ztest(mu0, sdev, n, xbar, type)
```

Going back to the driving speed example, the hypothesis test is

- H_0: $\mu = \mu_0$
- H_a: $\mu > \mu_0$

If a driver's measured average speed is $\bar{X} = 122$, the above code rejects H_0. This is consistent with the confidence interval cutoff we found above.



There are two types of possible errors we can make. a *Type I error* is when H_0 is true, but we reject it, and a *Type 2 error* is when H_0 is not true but we fail to reject it.

	H_0 is true	H_0 is false
do not reject H_0	$1-\alpha$	Type II error: β
reject H_0	Type I error: α	Power: $1-\beta$

Table 6.1 The error matrix.

We reject H_0 when the p-value of Z is less than the significance level α, which happens when $z < z^*$ or $z > z^*$ or $|z| > z^*$. In all cases, the chance of this happening is by definition α. In other words,

$$Prob(\text{Type I error}) = Prob(p\text{-value} < \alpha \mid H_0) = \alpha.$$

Thus *the probability of a type I error is the significance level* α.

We make a Type II error when we do not reject H_0, but H_0 is false. To compute the probability of a Type II error, suppose the true value of μ is μ_1. Then we do not reject H_0 if $|z| < |z^*|$, which is when μ_0 lies in the confidence interval $\bar{x} \pm z^*\sigma/\sqrt{n}$, or when $\bar{x}$ lies in the interval

$$\mu_0 - \frac{z^*\sigma}{\sqrt{n}} < \bar{x} < \mu_0 + \frac{z^*\sigma}{\sqrt{n}}.$$

But when $\mu = \mu_1$, $\bar{X}$ is $N(\mu_1, \sigma)$, so the probability of this event can be computed.

Standardize $\bar{X}$ by subtracting μ_1 and dividing by the standard error. Then we have a Type II error when

$$\sqrt{n}\frac{(\mu_0 - \mu_1)}{\sigma} - z^* < z < \sqrt{n}\frac{(\mu_0 - \mu_1)}{\sigma} + z^*.$$

If we set δ to equal the standardized difference in the means,

$$\delta = \sqrt{n}\frac{(\mu_0 - \mu_1)}{\sigma},$$

then we have a Type II error when

$$\delta - z^* < Z < \delta + z^*,$$

or when $|Z - \delta| < z^*$. Hence

$$Prob(\text{Type II error}) = Prob\left(|Z - \delta| < z^*\right).$$

This calculation was for a two-tail test. When the test is upper-tail or lower-tail, a similar calculation leads to the code

```
############################
# Type1 and Type2 errors - Z
############################

def type2_error(type, mu0, mu1, sdev, n, alpha):
   print("significance,mu0,mu1, sdev, n: ", alpha, mu0, mu1,
     ↪ sdev, n)
   print("prob of type1 error: ", alpha)
   delta = sqrt(n) * (mu0 -  mu1) / sdev
   if type == "lower-tail":
      zstar =  Z.ppf(alpha)
      type2 = 1 - Z.cdf(delta + zstar)
   elif type == "upper-tail":
      zstar = Z.ppf(1-alpha)
      type2 = Z.cdf(delta + zstar)
   elif type == "two-tail":
      zstar = Z.ppf(1 - alpha/2)
      type2 = Z.cdf(delta + zstar) - Z.cdf(delta - zstar)
   else: print("what's the test type?"); return
   print("test type: ",type)
   print("zstar: ", zstar)
   print("delta: ", delta)
   print("prob of type2 error: ", type2)
   print("power: ", 1 - type2)

mu0 = 120
mu1 = 122
sdev = 2
n = 10
alpha = .01
type = "upper-tail"

type2_error(type, mu0, mu1, sdev, n, alpha)
```



A type II error is when we do not reject the null hypothesis and yet it's false. The *power* of a test is the probability of rejecting the null hypothesis when it's false (Figure 6.1). If the probability of a type II error is β, then the power is $1 - \beta$.

Going back to the driving speed example, what is the chance that someone driving at $\mu_1 = 122$ is not caught? This is a type II error; using the above code, the probability is

$$\beta = Prob(\bar{X} = 120 \mid \mu = 122) = 20\%.$$

Therefore this test has power 80% to detect such a driver.

6.3 *T*-test

Let $X_1, X_2, \ldots, X_n$ be a simple random sample from a population. We repeat the previous section when we know neither the population mean μ, nor the population variance σ^2. We only know the sample mean

$$\bar{X} = \frac{X_1 + X_2 + \cdots + X_n}{n}$$

and the sample variance

$$S^2 = \frac{1}{n-1} \sum_{k=1}^{n} (X_k - \bar{X})^2.$$

For example, assume $X_1, X_2, \ldots, X_n$ are Bernoulli random variables with values 0, 1. Then as we've seen before,

$$(n-1)S^2 = \sum_{k=1}^{n} (X_k - \bar{X})^2 = n\bar{X}(1-\bar{X}).$$

From §5.5, *when the population is standard normal,*

- $(n-1)S^2$ is chi-squared of degree $n-1$, and
- $\bar{X}$ and S^2 are independent.



A *Student*[2] *random variable* is a continuous random variable T with probability density function

$$p(t) = C \cdot \left(1 + \frac{t^2}{d}\right)^{-(d+1)/2}, \qquad a < t < b. \tag{6.3.1}$$

Here C is a constant to make the total area under the graph equal to one (Figure 6.3).

The distribution of a Student random variable T is the *Student distribution with degree d*, also called the *t-distribution with degree d*. The Student distribution has `pdf` (6.3.1), and the probability that T lies in a *small* interval $[a, b]$ is

$$\frac{Prob(a < T < b)}{b-a} \approx p(t), \qquad a < t < b,$$

When the interval $[a, b]$ is not small, this is not correct. The exact formula for $Prob(a < T < b)$ is the area under the graph (Figure 6.3). This is obtained by integration,

[2] This terminology is due to the statistician R. A. Fisher.

$$Prob(a < T < b) = \int_a^b p(t)\,dt. \tag{6.3.2}$$

Under this interpretation, this probability corresponds to the area under the graph between the vertical lines at a and at b, and the total area under the graph corresponds to $a = -\infty$ and $b = \infty$.

More generally, means of $f(T)$ are computed by integration,

$$E(f(T)) = \int_{-\infty}^{\infty} f(t)p(t)\,dt,$$

with the integral computed via the fundamental theorem of calculus (A.6.2) or Python.

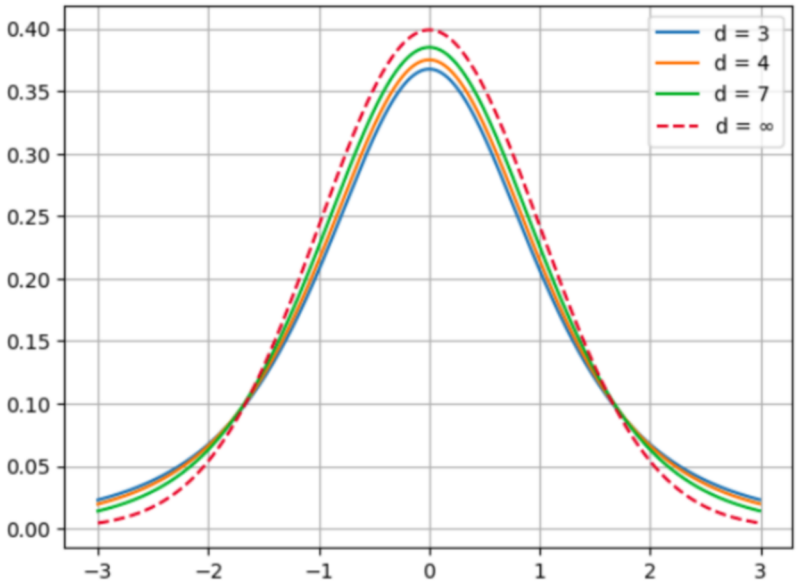


Fig. 6.3 Student distribution, against normal (dashed).

The Student `pdf` (6.3.1) approaches the standard normal `pdf` (5.4.1) as $d \to \infty$ (Exercise **6.3.1**).

```
from numpy import *
set_printoptions(legacy = "1.25")

from scipy.stats import t as T, norm as Z
from matplotlib.pyplot import *
```

```
for d in [3,4,7]:
   t = arange(-3,3,.01)
   plot(t,T(d).pdf(t),label = "d = "+str(d))

plot(t,Z.pdf(t),"--",label = r"d = $\infty$")
grid()
legend()
show()
```

The main result of this section, derived using calculus, is

Relation Between Z, U, and T

Suppose Z and U are independent, where Z is standard normal, and U is chi-squared with degree d. Then

$$T = \frac{Z}{\sqrt{U/d}}$$

is Student with degree d.



In the previous section, we normalized a sample mean by subtracting the mean μ and dividing by the standard error $\sigma/\sqrt{n}$. Since now we don't know σ, it is reasonable to divide by the sample standard error, obtaining

$$\sqrt{n} \cdot \frac{\bar{X} - \mu}{S} = \sqrt{n} \cdot \frac{\bar{X} - \mu}{\sqrt{\dfrac{1}{n-1}\displaystyle\sum_{k=1}^{n}(X_k - \bar{X})^2}}.$$

If we standardize each variable by

$$X_k = \mu + \sigma Z_k,$$

then we can verify

$$\bar{X} = \mu + \sigma\bar{Z}, \qquad \bar{Z} = \frac{Z_1 + Z_2 + \cdots + Z_n}{n},$$

and

$$S^2 = \sigma^2 \frac{1}{n-1}\sum_{k=1}^{n}(Z_k - \bar{Z})^2.$$

From this, we have

$$\sqrt{n}\cdot\frac{\bar{X}-\mu}{S}=\sqrt{n}\cdot\frac{\bar{Z}}{\sqrt{\dfrac{1}{n-1}\displaystyle\sum_{k=1}^{n}(Z_k-\bar{Z})^2}}=\sqrt{n}\cdot\frac{\bar{Z}}{\sqrt{U/(n-1)}}.$$

Using the main result with $d=n-1$, we arrive at

Samples and Student Distributions

Let X_1, X_2, ..., X_n be independent normal random variables with mean μ. Let $\bar{X}$ be the sample mean, let S^2 be the sample variance, and let

$$T=\sqrt{n}\cdot\frac{\bar{X}-\mu}{S}.$$

Then T is a Student random variable with degree $(n-1)$.

The takeaway here is we do not need to know the population standard deviations σ of X_1, X_2, ..., X_n to compute T.



The t-score t^* corresponding[3] to significance α is

```
alpha = .3
tstar = T(d).ppf(alpha)     # lower-tail, tstar < 0
tstar = T(d).ppf(1-alpha)   # upper-tail, tstar > 0
tstar = T(d).ppf(1-alpha/2) # two-tail,   tstar > 0
```

Here d is the degree of T. Then we have

```
############################
# Confidence Interval - T
############################

from numpy import *
from scipy.stats import t as T

def confidence_interval(xbar,s,n,alpha,type):
   d = n-1
   if type == "two-tail":
      tstar = T(d).ppf(1-alpha/2)
      L  = xbar - tstar * s / sqrt(n)
      U = xbar + tstar * s / sqrt(n)
   elif type == "upper-tail":
```

[3] Geometrically, the p-value $Prob(T>1)$ is the probability that a normally distributed point in $(d+1)$-dimensional spacetime is inside the light cone.

```
        tstar = T(d).ppf(1-alpha)
        L = xbar
        U = xbar + tstar* s / sqrt(n)
    elif type == "lower-tail":
        tstar = T(d).ppf(alpha)
        L = xbar + tstar* s / sqrt(n)
        U = xbar
    else: print("what's the test type?"); return
    print("type: ",type)
    return L, U

n = 10
xbar = 120
s = 2
alpha = .01
type = "upper-tail"
print("significance, s, n, xbar: ", alpha,s,n,xbar)

L,U = confidence_interval(xbar,s,n,alpha,type)
print("lower, upper: ", L,U)
```

Going back to the driving speed example from §6.2, instead of assuming the population standard deviation is $\sigma = 2$, we compute the sample standard deviation and find it's $S = 2$. Recomputing with $T(9)$, instead of Z, we see $(L, U) = (120, 121.78)$, so the cutoff now is $\mu^* = 121.78$, as opposed to $\mu^* = 121.47$ there.



We turn now to *hypothesis testing*. As before, we have two hypotheses, a null hypothesis and an alternate hypothesis. In the above setting where we are estimating a population parameter, the *null hypothesis* is that μ equals a certain value μ_0, and the *alternate hypothesis* is that μ is not equal to μ_0.

- H_0: $\mu = \mu_0$
- H_a: $\mu \neq \mu_0$.

Here is the code:

```
####################
# Hypothesis T-test
####################

from numpy import *
from scipy.stats import t as T

def ttest(mu0, s, n, xbar, type):
    d = n-1
    print("mu0, s, n, xbar: ", mu0, s, n, xbar)
```

```
    t = sqrt(n) * (xbar -  mu0) / s
    print("t: ",t)
    if type == "lower-tail": p = T(d).cdf(t)
    elif type == "upper-tail": p = 1 - T(d).cdf(t)
    elif type == "two-tail": p = 2 * (1 - T(d).cdf(abs(t)))
    print("pvalue: ",p)
    if p < alpha: print("reject H0")
    else: print("do not reject H0")

xbar = 122
n = 10
type = "upper-tail"
mu0 = 120
s = 2
alpha = .01

ttest(mu0, s, n, xbar,type)
```

Going back to the driving speed example, the hypothesis test is

- H_0: $\mu = \mu_0$
- H_a: $\mu > \mu_0$

If a driver's measured average speed is $\bar{X} = 122$, the above code rejects H_0. This is consistent with the confidence interval cutoff we found above. However, the p-value obtained here is greater than the corresponding p-value in §6.2.



For Type I and Type II errors, the code is

```
########################
# Type1 and Type2 errors
########################

from numpy import *
from scipy.stats import t as T

def type2_error(type, mu0, mu1, s, n, alpha):
   d = n-1
   print("significance, mu0, mu1, n: ", alpha, mu0, mu1, n)
   print("prob of type1 error: ", alpha)
   delta = sqrt(n) * (mu0 -  mu1) / s
   if type == "lower-tail":
      tstar =  T(d).ppf(alpha)
      type2 = 1 - T(d).cdf(delta + tstar)
   elif type == "upper-tail":
      tstar = T(d).ppf(1-alpha)
      type2 = T(d).cdf(delta + tstar)
   elif type == "two-tail":
```

```
        tstar = T(d).ppf(1 - alpha/2)
        type2 = T(d).cdf(delta + tstar) - T(d).cdf(delta - tstar)
    else: print("what's the test type?"); return
    print("test type: ",type)
    print("tstar: ", tstar)
    print("delta: ", delta)
    print("prob of type2 error: ", type2)
    print("power: ", 1 - type2)

mu0 = 120
mu1 = 122
s = 2
n = 10
alpha = .01
type = "upper-tail"

type2_error(type,mu0,mu1,s,n,alpha)
```

Going back to the driving speed example, if a driver's measured average speed is $\bar{X} = 122$, what is the chance they will not be fined? From the code, the probability of this Type II error is 37%, and the power to detect such a driver is 63%.

Exercises

Exercise 6.3.1 Use the compound-interest formula (A.3.9) to show the Student `pdf` (6.3.1) equals the standard normal `pdf` (5.4.1) when $d = \infty$. Since the formula for the constant C is not given, ignore C in your calculation.

6.4 Chi-Squared Tests

Let $X_1, X_2, \ldots, X_n$ be i.i.d. random variables, where each X_k is *categorical*. This means each X_k is a discrete random variable (§5.3), taking values in one of d categories. For simplicity, assume the categories are

$$i = 1, 2, \ldots, d.$$

When $d = 2$, this reduces to the Bernoulli case.

When $d = 2$ and $X_k = 0, 1$, the sample mean $\bar{X}$ is a proportion $\hat{p}$, the population mean is $p = Prob(X_k = 1)$, the population standard deviation is $\sqrt{p(1-p)}$, and the sample standard deviation is $\sqrt{\bar{X}(1-\bar{X})} = \sqrt{\hat{p}(1-\hat{p})}$. By the central limit theorem, the test statistic

$$Z = \sqrt{n} \cdot \frac{\hat{p} - p}{\sqrt{p(1-p)}} \tag{6.4.1}$$

is approximately standard normal for large enough sample size, and consequently $U = Z^2$ is approximately chi-squared with degree one. The chi-squared test generalizes this from $d = 2$ categories to $d > 2$ categories.

Given a category i, let $\#_i$ denote the number of times $X_k = i$, $1 \le k \le n$, in a sample of size n. Then $\#_i$ is the *count* that $X_k = i$, and $\hat{p}_i = \#_i/n$ is the *observed frequency*, in a sample of size n. Let p_i be the *expected frequency*,

$$p_i = Prob(X_k = i).$$

Then $p = (p_1, p_2, \ldots, p_d)$ is the *probability vector* associated to X. Since X_k are identically distributed, this does not depend on k.

By the central limit theorem,

$$\sqrt{n}(\hat{p}_i - p_i) = \sqrt{n}\left(\frac{\#_i}{n} - p_i\right),$$

are approximately normal for large n. Based on this, we have the

Goodness-Of-Fit Test

Let $\hat{p} = (\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_d)$ be the observed frequencies corresponding to samples X_1, X_2, $\ldots$, X_n, and let $p = (p_1, p_2, \ldots, p_d)$ be the expected frequencies. Then, for large sample size n, the statistic

$$n \sum_{i=1}^{d} \frac{(\hat{p}_i - p_i)^2}{p_i} \tag{6.4.2}$$

is approximately chi-squared with degree $d - 1$.

By clearing denominators, (6.4.2) may be rewritten in terms of counts as follows,

$$\sum_{i=1}^{d} \frac{(\#_i - np_i)^2}{np_i} = \sum_{i=1}^{d} \frac{(observed_i - expected_i)^2}{expected_i}.$$

When $d = 2$, this statistic reduces to Z^2, where Z is given by (6.4.1). Here is the code.

```
from numpy import *
set_printoptions(legacy = "1.25")

from scipy.stats import chi2 as U

def goodness_of_fit(observed,expected):
```

```
# assume len(observed) == len(expected)
d = len(observed)
u = sum([ (observed[i] - expected[i])**2/expected[i] for i in
  ↪ range(d) ])
pvalue = 1 - U(d-1).cdf(u)
print("statistic: ", u)
print("significance: ", pvalue)
```



Suppose a dice is rolled $n = 120$ times, and the observed counts are

$$O_1 = 17, O_2 = 12, O_3 = 14, O_4 = 20, O_5 = 29, O_6 = 28.$$

Notice

$$O_1 + O_2 + O_3 + O_4 + O_5 + O_6 = 120.$$

If the dice is fair, the expected counts are

$$E_1 = 20, E_2 = 20, E_3 = 20, E_4 = 20, E_5 = 20, E_6 = 20.$$

Based on the observed counts, at 5% significance, what can we conclude about the dice?

```
observed = array([27,12,14,20,29,28])
expected = array([20,20,20,20,20,20])

goodness_of_fit(observed,expected)
```

Here there are $d = 6$ categories, and the statistic (6.4.2) and corresponding significance equal

$$u = 14.7, \qquad \alpha = 1.17\%.$$

The dice is fair if u is not large and the dice is unfair if u is large. Since 1.17% is less than 5%, the dice is not fair.



To derive the goodness-of-fit test, let X be a discrete random variable, taking values in 1, 2, ..., d, with distribution $p = (p_1, p_2, \ldots, p_d)$. We vectorize (§1.3) X by defining the one-hot encoded (§2.4) vector-valued random variable $V = (V_1, V_2, \ldots, V_d)$ as follows,

$$V_i = \begin{cases} \dfrac{1}{\sqrt{p_i}} & \text{if } X = i, \\ 0 & \text{if } X \neq i. \end{cases} \tag{6.4.3}$$

For future reference, we denote $V = \text{vect}_p(X)$.

Then

$$E(V_i) = \frac{1}{\sqrt{p_i}} Prob(X = i) = \frac{p_i}{\sqrt{p_i}} = \sqrt{p_i},$$

and

$$E(V_i V_j) = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j, \end{cases}$$

for $i, j = 1, 2, \ldots, d$. If

$$\mu = \left(\sqrt{p_1}, \sqrt{p_2}, \ldots, \sqrt{p_d}\right),$$

we conclude

$$E(V) = \mu, \qquad E(V \otimes V) = I.$$

From this,

$$E(V) = \mu, \qquad Var(V) = I - \mu \otimes \mu. \tag{6.4.4}$$

Now define

$$V_k = \text{vect}_p(X_k), \qquad k = 1, 2, \ldots, n.$$

Since $X_1, X_2, \ldots, X_n$ are i.i.d, $V_1, V_2, \ldots, V_n$ are i.i.d. By (5.5.5), we conclude the random vector

$$Z = \sqrt{n}\left(\frac{1}{n}\sum_{k=1}^{n} V_k - \mu\right)$$

has mean zero and variance $I - \mu \otimes \mu$.

Since $V_1, V_2, \ldots, V_n$ are i.i.d, by the central limit theorem, we also conclude Z is approximately normal for large n.

Since

$$|\mu|^2 = (\sqrt{p_1})^2 + (\sqrt{p_2})^2 + \cdots + (\sqrt{p_d})^2 = p_1 + p_2 + \cdots + p_d = 1,$$

μ is a unit vector. By the singular chi-squared result in §5.5, $|Z|^2$ is approximately chi-squared with degree $d - 1$. Since

$$Z_i = \sqrt{n}\left(\frac{\hat{p}_i}{\sqrt{p_i}} - \sqrt{p_i}\right), \tag{6.4.5}$$

we write $|Z|^2$ out,

$$|Z|^2 = \sum_{i=1}^{d} Z_i^2 = n\sum_{i=1}^{d}\left(\frac{\hat{p}_i}{\sqrt{p_i}} - \sqrt{p_i}\right)^2 = n\sum_{i=1}^{d}\frac{(\hat{p}_i - p_i)^2}{p_i},$$

obtaining (6.4.2).



Suppose $X_1, X_2, \ldots, X_n$ and $Y_1, Y_2, \ldots, Y_n$ are samples measuring two possibly related effects. Suppose the X variables take on d categories, $i = 1, 2, \ldots, d$, and the Y variables take on N categories, $j = 1, 2, \ldots, N$. Let

$$p_i = Prob(X_k = i), \qquad q_j = Prob(Y_k = j),$$

and set $p = (p_1, p_2, \ldots, p_d)$, $q = (q_1, q_2, \ldots, q_N)$. The goal is test whether the two effects are independent or not.

Let

$$r_{ij} = Prob(X_k = i \text{ and } Y_k = j).$$

Then r is a $d \times N$ matrix. The effects are independent when

$$r_{ij} = p_i q_j,$$

or $r = p \otimes q$.

For example, suppose 300 people are polled and the results are collected in a *contingency table* (Figure 6.2).

	Democrat	Republican	Independent	Total
Women	68	56	32	156
Men	52	72	20	144
Total	120	128	52	300

Table 6.2 $2 \times 3 = d \times N$ contingency table [30].

Is a person's gender correlated with their party affiliation, or are the two variables independent? To answer this, let $\hat{p}$ and $\hat{q}$ be the observed frequencies

$$\hat{p}_i = \frac{\#\{k : X_k = i\}}{n}, \qquad \hat{q}_j = \frac{\#\{k : Y_k = j\}}{n},$$

and let $\hat{r}$ be the joint observed frequencies

$$\hat{r}_{ij} = \frac{\#\{k : X_k = i \text{ and } Y_k = j\}}{n}.$$

Then $\hat{r}$ is also a $d \times N$ matrix.

When the effects are independent, $r = p \otimes q$, so, by the law of large numbers, we should have

$$\hat{r} \approx \hat{p} \otimes \hat{q}$$

for large sample size. The chi-squared independence test quantifies the difference of the two matrices r and $\hat{r}$.

Chi-squared Independence Test

If $X_1, X_2, \ldots, X_n$ and $Y_1, Y_2, \ldots, Y_n$ are independent, then, for large sample size n, the statistic

$$n \sum_{i,j=1}^{d,N} \frac{(\hat{r}_{ij} - \hat{p}_i \hat{q}_j)^2}{\hat{p}_i \hat{q}_j} \tag{6.4.6}$$

is approximately chi-squared with degree $(d-1)(N-1)$.

Only sample data is used to compute the statistic (6.4.6), knowledge of p and q is not needed. Conversely, the test says nothing about p and q, and only queries independence.

By clearing denominators, (6.4.6) may be rewritten in terms of counts as follows,

$$\begin{aligned} \sum_{i,j=1}^{d,N} \frac{\left(n\#_{ij}^{XY} - \#_i^X \#_j^Y\right)^2}{n\#_i^X \#_j^Y} &= -n + n \sum_{i,j=1}^{d,N} \frac{\left(\#_{ij}^{XY}\right)^2}{\#_i^X \#_j^Y} \\ &= -n + n \sum_{i,j=1}^{d,N} \frac{(observed)^2}{expected}. \end{aligned}$$



The code

```
# table is dxN numpy array

def chi2_independence(table):
   n = sum(table) # total sample size
   d = len(table)
   N = len(table.T)
   rowsum = array([ sum(table[i,:]) for i in range(d) ])
   colsum = array([ sum(table[:,j]) for j in range(N) ])
   expected = outer(rowsum,colsum) # tensor product
   u = -n + n*sum([[ table[i,j]**2/expected[i,j] for j in
    ↪ range(N) ] for i in range(d) ])
   deg = (d-1)*(N-1)
   pvalue = 1 - U(deg).cdf(u)
   return pvalue

table = array([[68,56,32],[52,72,20]])
chi2_independence(table)
```

returns a p-value of 0.0401, so, at the 5% significance level, the effects are not independent.

The derivation of the independence test is similar to the goodness-of-fit test. There are two differences. First, because there are two indices $X_k = i$, $Y_k = j$, we work with matrices, not vectors. Second, we appeal to the law of large numbers to replace p_i by $\hat{p}_i$ and q_j by $\hat{q}_j$ for large n.

Let Z be the $d \times N$ matrix

$$Z_{ij} = \sqrt{n}\left(\frac{\hat{r}_{ij} - \hat{p}_i\hat{q}_j}{\sqrt{\hat{p}_i}\sqrt{\hat{q}_j}}\right). \tag{6.4.7}$$

Then (see (2.2.16))

$$\|Z\|^2 = \operatorname{trace}(Z^t Z) = \sum_{i,j=1}^{d,N} Z_{ij}^2$$

equals (6.4.6).

Let $u_1, u_2, \ldots, u_d$ and $v_1, v_2, \ldots, v_N$ be orthonormal bases for $\mathbf{R}^d$ and $\mathbf{R}^N$ respectively. By (2.9.8),

$$\|Z\|^2 = \operatorname{trace}(Z^t Z) = \sum_{i,j=1}^{d,N} (u_i \cdot Z v_j)^2. \tag{6.4.8}$$

We will show $\|Z\|^2$ is asymptotically chi-squared of degree $(d-1)(N-1)$. To achieve this, we show Z is asymptotically normal.

Let X and Y be discrete random variables with probability vectors $p = (p_1, p_2, \ldots, p_d)$ and $q = (q_1, q_2, \ldots, q_N)$, and assume X and Y are independent.

Let

$$\mu = (\sqrt{p_1}, \sqrt{p_2}, \ldots, \sqrt{p_d}), \qquad \nu = (\sqrt{q_1}, \sqrt{q_2}, \ldots, \sqrt{q_N}).$$

Then μ and ν are unit vectors. Following (6.4.3), define

$$M = (\operatorname{vect}_p(X) - \mu) \otimes (\operatorname{vect}_q(Y) - \nu). \tag{6.4.9}$$

Then M is a $d \times N$ matrix-valued random variable, and

$$u \cdot Mv = (\operatorname{vect}_p(X) \cdot u - \mu \cdot u)(\operatorname{vect}_q(Y) \cdot v - \nu \cdot v).$$

If u and v are unit vectors in $\mathbf{R}^d$ and $\mathbf{R}^N$ respectively, by (6.4.4),

$$E(\operatorname{vect}_p(X) \cdot u) = \mu \cdot u, \qquad Var(\operatorname{vect}_p(X) \cdot u) = 1 - (\mu \cdot u)^2,$$

and

$$E(\text{vect}_q(Y)\cdot v)=\nu\cdot v,\qquad Var(\text{vect}_q(Y)\cdot v)=1-(\nu\cdot v)^2.$$

By independence of X and Y, the mean of $u\cdot Mv$ is zero, and

$$Var(u\cdot Mv)=\left(1-(\mu\cdot u)^2\right)\left(1-(\nu\cdot v)^2\right).$$

In particular, when the unit vectors u or v are orthogonal to μ and ν respectively, $u\cdot Mv$ is a standard random variable, i.e. has mean zero and variance one. This also shows $u\cdot M\nu=0$, $\mu\cdot Mv=0$ for any u and v.

More generally (Exercise **6.4.3**) $u\cdot Mv$ and $u'\cdot Mv'$ are uncorrelated when $u\perp u'$ and $v\perp v'$.

Our goal is to show for large n, Z has the same mean and variance as that of M. If we also show $u\cdot Zv$ and $u'\cdot Zv'$ are independent for large n when $u\perp u'$ and $v\perp v'$, then (6.4.8) leads to the result. Now to the details.

Let $W=W_n$ be a random variable that depends on n. We write

$$W\approx 0$$

if all probabilities of W converge to zero for n large. In this case, we say W is *asymptotically zero* (see §A.7 for more information).

Let W' be another random variable depending on n. We write $W\approx W'$, and we say W is *asymptotically equal* to W', if all probabilities of W and W' agree asymptotically for n large. In particular, when $W\approx W'$,

$$E(W)\approx E(W'),\qquad Var(W)\approx Var(W').$$

If $W\approx W'$ and W' is a normal random variable not depending on n, we write $W\approx$ normal, and we say W is *asymptotically normal*. If $W\approx W'$ and $W'\approx$ normal, then $W\approx$ normal.

Let M_k correspond to X_k and Y_k, $k=1,2,\dots,n$, and let

$$Z^{CLT}=\sqrt{n}\left(\frac{1}{n}\sum_{k=1}^{n}M_k-E(M_k)\right)=\frac{1}{\sqrt{n}}\sum_{k=1}^{n}M_k.$$

Then, by independence, and the central limit theorem,

- the mean and variance of Z^{CLT} are the same as those of M,
- $u\cdot Z^{CLT}\nu=0$, $\mu\cdot Z^{CLT}v=0$ for any u and v, and,
- $Z^{CLT}\approx$ normal.

Although Z and Z^{CLT} are not equal, we will show $Z\approx Z^{CLT}$. To this end, multiplying out the expression (6.4.9) for each $M=M_k$, and summing over $k=1,2,\dots,n$, we see

$$Z_{ij}^{CLT}=\sqrt{n}\left(\frac{\hat{r}_{ij}}{\sqrt{p_i}\sqrt{q_j}}-\frac{\hat{p}_i\sqrt{q_j}}{\sqrt{p_i}}-\frac{\hat{q}_j\sqrt{p_i}}{\sqrt{q_j}}+\sqrt{p_iq_j}\right).\tag{6.4.10}$$

By the law of large numbers, $\hat{p}_i\approx p_i$ and $\hat{q}_j\approx q_j$ so

$$\frac{\hat{q}_j - q_j}{\sqrt{\hat{p}_i}\sqrt{\hat{q}_j}} \approx 0.$$

As we saw before (6.4.5), by the central limit theorem,

$$\sqrt{n}\,(\hat{p}_i - p_i) \approx \text{normal}.$$

Hence[4] the product

$$\sqrt{n} \cdot \frac{(p_i - \hat{p}_i)(\hat{q}_j - q_j)}{\sqrt{\hat{p}_i}\sqrt{\hat{q}_j}} \approx 0. \tag{6.4.11}$$

Similarly, $\hat{p}_i \approx p_i$ and $\hat{q}_j \approx q_j$, so

$$Z_{ij}^{CLT}\left(\frac{\sqrt{p_i}\sqrt{q_j}}{\sqrt{\hat{p}_i}\sqrt{\hat{q}_j}} - 1\right) \approx 0. \tag{6.4.12}$$

Adding (6.4.10), (6.4.11), and (6.4.12), we obtain (6.4.7), hence

$$Z \approx Z^{CLT}.$$

We conclude

- the mean and variance of Z are asymptotically the same as those of M,
- $u \cdot Z\nu \approx 0$, $\mu \cdot Zv \approx 0$ for any u and v, and,
- $Z \approx$ normal.

In particular, since $u \cdot Zv$ and $u' \cdot Zv'$ are asymptotically uncorrelated when $u \perp u'$ and $v \perp v'$, and Z is asymptotically normal, we conclude $u \cdot Zv$ and $u' \cdot Zv'$ are asymptotically independent when $u \perp u'$ and $v \perp v'$.

Now choose the orthonormal bases with u_1 and v_1 equal to μ and ν respectively. Then

$$u_i \cdot Zv_j, \qquad i = 1, 2, 3, \ldots, d, j = 1, 2, 3, \ldots, N$$

are independent normal random variables with mean zero, asymptotically for large n, and variances according to the listing

$\mu \cdot Z\nu$	$\mu \cdot Zv_2$	.....	$\mu \cdot Zv_{N-1}$	$\mu \cdot Zv_N$		0	0	.....	0	0
$u_2 \cdot Z\nu$	$u_2 \cdot Zv_2$	.....	$u_2 \cdot Zv_{N-1}$	$u_2 \cdot Zv_N$		0	1	.....	1	1
...	...	.....	...	...	$\approx$	...	.....			
$u_{d-1} \cdot Z\nu$	$u_{d-1} \cdot Zv_2$	.....	$u_{d-1} \cdot Zv_{N-1}$	$u_{d-1} \cdot Zv_N$		0	1	.....	1	1
$u_d \cdot Z\nu$	$u_d \cdot Zv_2$	.....	$u_d \cdot Zv_{N-1}$	$u_d \cdot Zv_N$		0	1	.....	1	1

Table 6.3 Limiting variances.

[4] The theoretical basis for this intuitively obvious result is *Slutsky's theorem* [8].

From this, only $(d-1)(N-1)$ terms are nonzero in (6.4.8), hence $\|Z\|^2$ is chi-squared with degree $(d-1)(N-1)$, completing the proof.

Exercises

Exercise 6.4.1 Let V be the vectorization (6.4.3) of the discrete random variable X, and let μ be the mean of V. Then $V \cdot \mu = 1$.

Exercise 6.4.2 Verify (6.4.10).

Exercise 6.4.3 Let M be as in (6.4.9). Then $u \cdot Mv$ and $u' \cdot Mv'$ are uncorrelated when $u \perp u'$ and $v \perp v'$.

Exercise 6.4.4 Verify the goodness-of-fit test statistic (6.4.2) is the square of (6.4.1) when $d = 2$.

Exercise 6.4.5 [30] Among 100 vacuum tubes tested, 41 had lifetimes of less than 30 hours, 31 had lifetimes between 30 and 60 hours, 13 had lifetimes between 60 and 90 hours, and 15 had lifetimes of greater than 90 hours. Are these data consistent with the hypothesis that a vacuum tube's lifetime is exponentially distributed (Exercise **5.3.23**) with a mean of 50 hours? At what significance? Here $p = (p_1, p_2, p_3, p_4)$.

Exercise 6.4.6 [30] A study was instigated to see if southern California earthquakes of at least moderate size are more likely to occur on certain days of the week than on others. The catalogs yielded the data in Figure 6.4. Test, at the 5 percent level, the hypothesis that an earthquake is equally likely to occur on any of the 7 days of the week.

Day	Sun	Mon	Tues	Wed	Thurs	Fri	Sat	Total
Number of Earthquakes	156	144	170	158	172	148	152	1100

Table 6.4 Earthquake counts.

Exercise 6.4.7 [30] In a famous article (S. Russell, "A red sky at night..." Metropolitan Magazine London, 61, p. 15, 1926) the following dataset of frequencies of sunset colors and whether each was followed by rain was presented. Test the hypothesis that whether it rains tomorrow is independent of the color of today's sunset.

Sky Color	Number of Observations	Number Followed by Rain
Red	61	26
Mainly red	194	52
Yellow	159	81
Mainly yellow	188	86
Red and yellow	194	52
Gray	302	167

Table 6.5 Sunset and rain counts.

Exercise 6.4.8 [30] A sample of 300 cars having mobile phones and one of 400 cars without phones were tracked for 1 year. The following table gives the number of these cars involved in accidents over that year. Use the above to test the hypothesis that having a mobile phone in your car and being involved in an accident are independent. Use the 5 percent level of significance.

	Accident	No Accident
Mobile phone	22	278
No phone	26	374

Table 6.6 Phone and accident counts.

Chapter 7
Machine Learning

In this chapter, we go over the structure of neural networks, in enough detail to write weight gradient code. Here the neural networks need not be layered, they are allowed any topology. Then we study gradient descent methods, and use them to train neural networks. We also analyze stochastic gradient descent, and gradient descent with momentum.

The aim here is to consider only model cases where the analysis flows smoothly. There are many variations, both theoretical and implementation issues, that improve real-life performance, see [4] and [39]. Here we avoid these variations in order to highlight the main ideas.

A key application of these ideas to binary classifiers is in §7.6. Since this application appears for the first time here and in [16], we describe the problem.

A two-class dataset is a targeted dataset $x_1, x_2, \ldots, x_N$ where the targets or labels are *bits* $p_1, p_2, \ldots, p_N$ reflecting the two classes $p = 0$ and $p = 1$ (§4.5).

Given a two-class dataset, a *binary classifier* is a procedure for classifying points in sample space into two classes, in a manner consistent with the dataset [2].

One way to obtain a binary classifier is to use a hyperplane to separate the sample space into two parts. This is a *linear* binary classifier [2]. More generally, given a hyperplane, we seek a procedure that computes a probability $q = q(x)$, depending only on the hyperplane, that a point x should be assigned to the class $p = 1$.

To be as consistent as possible with the dataset, we choose some *measure of discrepancy* $I(p, q)$ between the label p and the probability q corresponding to x, and we select the hyperplane that *minimizes* the *loss function*, by definition the average of the discrepancies over all samples. This is *logistic regression*.

Stated in this manner, logistic regression is a special case of neural network training. Even in this special case, the loss function may not have a minimizer. When the loss function does have a minimizer, we say logistic regression is *trainable*.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

O. Hijab, *Math for Data Science*, https://doi.org/10.1007/978-3-031-89707-8_7

In §7.6, it is shown logistic regression is not trainable if and only if the dataset classes are separable by a hyperplane. Equivalently, logistic regression is trainable if and only if the dataset classes are not separable by a hyperplane. This result is a *fundamental theorem of linear binary classifiers*.

As a consequence, for the logistic regression loss function, short-step gradient descent either diverges to infinity, or converges to the unique minimizer, according to whether the dataset classes are separable or not (§7.6).

This last result is a criterion for checking the hyperplane separability of a two-class dataset.

7.1 Overview

This first section is an overview of neural network training. Here is a summary of the structure of neural networks.

- A graph consists of nodes and edges (§3.3).
- If each edge has a direction, the graph is directed.
- If each edge has a weight, the graph is weighed.
- A directed graph has input nodes, output nodes, and hidden nodes.
- A node with an activation function f is a neuron (§4.4).
- Each neuron has inputs and an output. The output is the result of inserting the inputs into f.
- A network is a weighed directed graph where all hidden nodes are neurons.
- A neural network is a network where each activation function is a function of the *weighed sum* of the inputs (§7.2).

The goal is to train a neural network. To *train* a neural network means to find weights W so that the input-output behavior of the network is as close as possible to a given dataset of source-target sample pairs $(x, y) = (x_k, y_k)$, $k = 1, 2, \ldots, N$.

To do this, we must have an error measure J between outputs and targets. Since outputs depend on sources and weights, so does $J = J(x, y, W)$. Here is a summary of how neural networks are trained (§7.4).

1. Start with a source-target pair (x, y) and a weight matrix W.
2. Inject source x at the input nodes.
3. Compute the output x at all nodes (forward propagation).
4. Inject the target y and the derivative δ of J at the output nodes.
5. Compute the derivative δ of J at all nodes (back propagation).
6. Compute the error J between output and target.
7. Compute the weight gradient $\nabla_W J = \delta \otimes x$.
8. Update W using gradient descent (§7.3), $W^+ = W - t\nabla_W J$ (§7.4).
9. Repeat steps 1-8 over all sample pairs (x_k, y_k), $k = 1, 2, \ldots, N$ (§7.4).
10. Repeat step 9 until convergence.

Steps 1-8 is an *iteration*, and step 9 is an *epoch*. An iteration uses a single sample (more generally a batch of samples), and an epoch uses the entire dataset. The *mean loss* or *mean error* over the dataset is

$$J(W) = \frac{1}{N} \sum_{k=1}^{N} J(x_k, y_k, W).$$

With the dataset given, the mean loss is a function of the weights.

A weight matrix W^* is *optimal* if it is a minimizer of the mean error,

$$J(W^*) \leq J(W), \qquad \text{for all } W.$$

Convergence means W is close to W^*.

7.2 Neural Networks

In §4.4, we saw two versions of forward and back propagation. In this section we see a third version. We begin by reviewing the definition of graph and network as given in §3.3 and §4.4.

A *graph* consists of *nodes* and *edges*. Nodes are also called *vertices*, and an edge is an ordered pair (j, i) of nodes. Because the ordered pair (i, j) is not the same as the ordered pair (j, i), our graphs are *directed*.

The edge (j, i) is *incoming* to node i and *outgoing* from node j. If a node j has no outgoing edges, then j is an *output node*. If a node i has no incoming edges, then i is an *input node*. If a node is neither an input nor an output, it is a *hidden node*.

We assume our graphs have no cycles: every forward path terminates at an output node in a finite number of steps, and every backward path terminates at an input node in a finite number of steps.

A graph is *weighed* if a scalar weight w_{ij} is attached to each edge (j, i) pointing node j to node i. If (j, i) is not an edge, we set $w_{ij} = 0$. To repeat, the weight w_{ij} is on the edge (j, i), not (i, j). This convention, assumed already in §3.3, simplifies the formulas in what follows.

If a graph has d nodes, the nodes are labeled 0, 1, 2, ..., $d - 1$, and the edges are completely specified by the $d \times d$ *weight matrix* $W = (w_{ij})$.

A node with an attached *activation function* f_i is a *neuron*. A *network* is a directed weighed graph where nodes are input nodes, output nodes, or neurons. In the next paragraph, we define a special kind of network, a neural network.

We attach to each node i an *outgoing signal* recursively as follows.

- At each input node i, the outgoing signal x_i is obtained from a dataset sample.

- At each neuron i, the outgoing signal x_i is obtained by evaluating the activation function f_i on the outgoing signals of all nodes pointing to i.
- At each output node i, there is no outgoing signal, and $x_i = \texttt{None}$.

Explicitly, at each neuron i, the *incoming list* is

$$x_i^- = (w_{i0}x_0, w_{i1}x_1, \ldots, w_{i(d-1)}x_{d-1}).$$

Because $w_{ij} = 0$ if (j, i) is not an edge, the nonzero entries in the incoming list to node i correspond to the edges incoming to node i. Then

$$x_i = f_i(x_i^-).$$

This is the setup for networks generally (§4.4). Now we specialize to neural networks.



A *neural network* is a network where every activation function is the *sum* of the entries of the incoming list,

$$x_i = f_i\left(\sum_{j\to i} w_{ij}x_j\right).$$

Here the sum is over all nodes j sending signals to node i. All the networks in this section are neural networks, but the network in Figure 4.16 is not a neural network.

In a neural network, if i is not an input node, we set

$$x_i^- = \sum_{j\to i} w_{ij}x_j. \tag{7.2.1}$$

Then x_i^- is the *incoming signal* to node i. For general networks, as in the previous paragraph or as in §4.4, x_i^- was a vector. In this chapter, x_i^- is the scalar given by (7.2.1).

The outgoing signal from node i is

$$x_i = f_i(x_i^-). \tag{7.2.2}$$

When i is an input node, $x_i^- = \texttt{None}$. When i is an output node, $x_i = \texttt{None}$.

If the network has d nodes, the *outgoing vector* is

$$x = (x_0, x_1, \ldots, x_{d-1}),$$

and the *incoming vector* is

$$x^- = (x_0^-, x_1^-, \ldots, x_{d-1}^-).$$

Let W be the weight matrix. If the network has d nodes, the *activation vector* is

$$f = (f_0, f_1, \ldots, f_{d-1}).$$

where f_j is the source on input nodes, and f_j=`None` on output nodes. Then a neural network may be written in vector-matrix form

$$x = f(Wx).$$

However, this representation is more useful when the network has structure, for example in a dense shallow layer (7.2.14).

Neural networks gained wide exposure after Minsky and Papert's famous 1969 book [21], from which Figure 7.1 is taken.

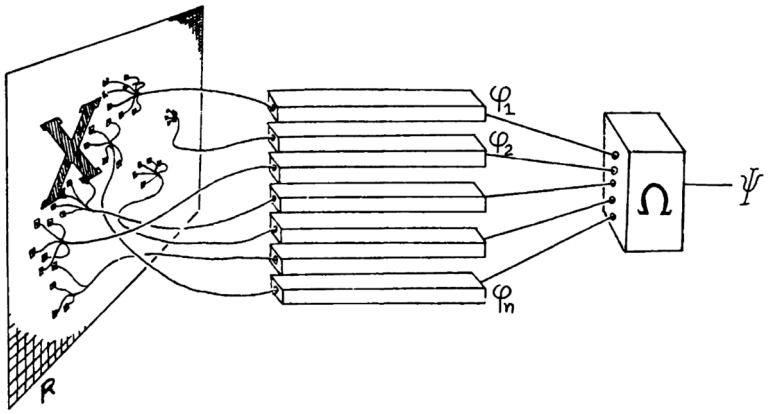


Fig. 7.1 Perceptrons in parallel (R in the figure is the retina) [21].

A *perceptron* is a network of the form

$$y = f(w_0x_0 + w_1x_1 + \cdots + w_{d-1}x_{d-1}) = f(w \cdot x)$$

(Figure 7.2). This is the simplest neural network.

If we set $y^- = w \cdot x$, then y^- is the incoming signal, and $y = f(y^-)$ is the outgoing signal.

Thus a perceptron is a linear function followed by an activation function. By our definition of neural network,

Neural Network

Every neural network is a combination of perceptrons.

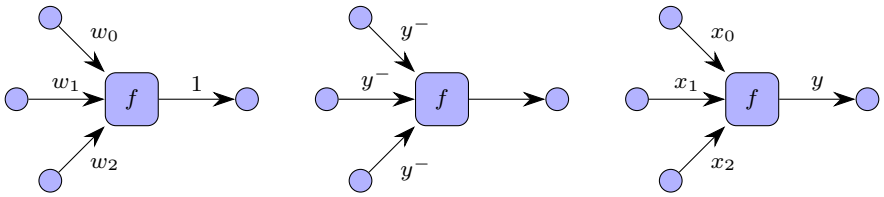


Fig. 7.2 A perceptron: weights, incoming signals, and outgoing signals.

When one of the inputs x_0 is fixed to equal 1, $x_0 = 1$, the corresponding weight w_0 is called a *bias*. The role of the bias is to shift the threshold in the activation function.

In this case, we write

$$y = f(m_1x_1 + m_2x_2 + \cdots + m_dx_d + b) = f(m \cdot x + b) = f(w \cdot \tilde{x}),$$

with

$$w = \begin{pmatrix} m \\ b \end{pmatrix}, \qquad \tilde{x} = \begin{pmatrix} x \\ 1 \end{pmatrix},$$

and we call $\tilde{x}$ the *augmented sample.*

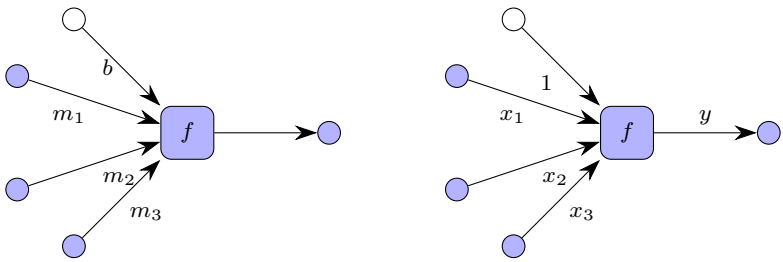


Fig. 7.3 A perceptron with bias: weights and outgoing signals.

If $x_1, x_2, \ldots, x_N$ is a dataset, then

$$\tilde{x}_1 = \begin{pmatrix} x_1 \\ 1 \end{pmatrix}, \qquad \tilde{x}_2 = \begin{pmatrix} x_2 \\ 1 \end{pmatrix}, \qquad \ldots \qquad \tilde{x}_N = \begin{pmatrix} x_N \\ 1 \end{pmatrix}, \tag{7.2.3}$$

is the *augmented* dataset. In this regard, Exercise **7.2.2** is relevant.

By passing to the augmented dataset, a neural network with bias can be thought of as a neural network without bias but with one additional feature, the bias input. Augmented datasets are studied in regression §7.7.

In (5.2.21), Bayes theorem is used to express a conditional probability in terms of a perceptron,

$$Prob(H \mid x) = \sigma(m \cdot x + b).$$

This is a basic example of how a perceptron computes probabilities.



Here is a listing of common activation functions.

- The identity function,
$$\mathrm{id}(z) = z$$
and its derivative $\mathrm{id}' = 1$.
- The binary output,
$$\mathrm{bin}(z) = \begin{cases} 1 & \text{if } z > 0, \\ 0 & \text{if } z < 0, \end{cases}$$
and its derivative $\mathrm{bin}' = 0$, $z \neq 0$, and $\mathrm{bin}'(0)$ undefined.
- The logistic or sigmoid function (Figure 5.9)
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$
and its derivative $\sigma' = \sigma(1 - \sigma)$.
- The hyperbolic tangent function
$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$
and its derivative $\tanh' = 1 - \tanh^2$.
- The rectified linear unit relu,
$$\mathrm{relu}(z) = \begin{cases} z & \text{if } z \geq 0, \\ 0 & \text{if } z < 0, \end{cases}$$
and its derivative $\mathrm{relu}' = \mathrm{bin}$.

Here is the code

```
from numpy import *
set_printoptions(legacy = "1.25")

# activation functions

def relu(z): return 0 if z < 0 else z
def bin(z): return 0 if z < 0 else 1
def sigmoid(z): return 1/(1+exp(-z))
def id(z): return z
# tanh already part of numpy
def one(z): return 1
def zero(z): return 0

# derivative of relu is bin
# derivative of bin is zero
# derivative of s=sigmoid is s*(1-s)
# derivative of id is one
# derivative of tanh is 1-tanh**2

def D_relu(z): return bin(z)
def D_bin(z): return 0
def D_sigmoid(z): return sigmoid(z)*(1-sigmoid(z))
def D_id(z): return 1
def D_relu(z): return bin(z)
def D_tanh(z): return 1 - tanh(z)**2

der_dict = { relu:D_relu, id:D_id, bin:D_bin, sigmoid:D_sigmoid,
    ↪ tanh: D_tanh}
```



The neural network in Figure 7.4 has nodes 0, 1, 2, 3, 4, 5, 6, 7 and activation functions f_2, f_3, f_4, f_5. Here 0 and 1 are input nodes, 2, 3, 4, 5 are neurons, and 6, 7 are output nodes. Figures 7.5 and 7.6 show the incoming and outgoing signals.

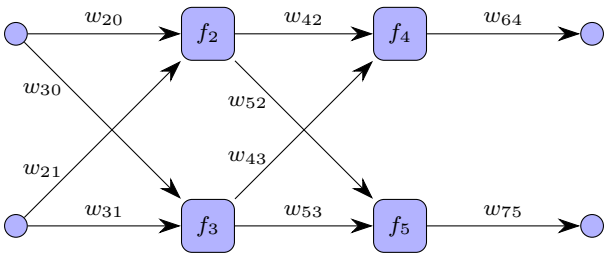


Fig. 7.4 Neural network: weights.

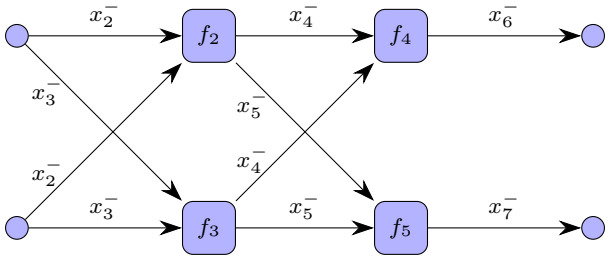


Fig. 7.5 Neural network: incoming signals.

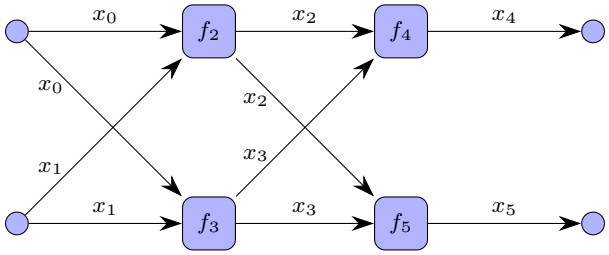


Fig. 7.6 Neural network: outgoing signals.

This network has incoming and outgoing signals

$$x^- = (\texttt{None}, \texttt{None}, x_2^-, x_3^-, x_4^-, x_5^-, x_6^-, x_7^-),$$
$$x = (x_0, x_1, x_2, x_3, x_4, x_5, \texttt{None}, \texttt{None}).$$

In Exercise **7.2.4**, this network is modified to include biases.



In a neural network, we insert the activation functions along the diagonal of the weight matrix. We also indicate the bias input node i by setting the diagonal entry $w_{ii} = \texttt{"b"}$. The resulting matrix is the *network weight matrix*.

When the nonzero weights are all equal to 1, the matrix is the *network adjacency matrix*. We have seen four types of graph matrices:

- adjacency matrix,
- weight matrix,
- network weight matrix,
- network adjacency matrix.

A neural network is completely characterized by its network adjacency matrix. Here are the network weight matrix and network adjacency matrix for the network in Figure 7.4.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0\,0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0\,0 \\ w_{20} & w_{21} & f_2 & 0 & 0 & 0 & 0\,0 \\ w_{30} & w_{31} & 0 & f_3 & 0 & 0 & 0\,0 \\ 0 & 0 & w_{42} & w_{43} & f_4 & 0 & 0\,0 \\ 0 & 0 & w_{52} & w_{53} & 0 & f_5 & 0\,0 \\ 0 & 0 & 0 & 0 & w_{64} & 0 & 0\,0 \\ 0 & 0 & 0 & 0 & 0 & w_{75} & 0\,0 \end{pmatrix}, \quad \begin{pmatrix} 0\,0 & 0 & 0 & 0 & 0 & 0 & 0\,0 \\ 0\,0 & 0 & 0 & 0 & 0 & 0 & 0\,0 \\ 1\,1 & f_2 & 0 & 0 & 0 & 0 & 0\,0 \\ 1\,1 & 0 & f_3 & 0 & 0 & 0 & 0\,0 \\ 0\,0 & 1 & 1 & f_4 & 0 & 0 & 0\,0 \\ 0\,0 & 1 & 1 & 0 & f_5 & 0 & 0\,0 \\ 0\,0 & 0 & 0 & 1 & 0 & 0 & 0\,0 \\ 0\,0 & 0 & 0 & 0 & 1 & 0 & 0\,0 \end{pmatrix}. \tag{7.2.4}$$

In the current section, we run code with the specific network weight matrix

$$W = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0\,0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0\,0 \\ .1 & .1 & \text{relu} & 0 & 0 & 0 & 0\,0 \\ -2.0 & -2.0 & 0 & \text{id} & 0 & 0 & 0\,0 \\ 0 & 0 & -.3 & .22 & \sigma & 0 & 0\,0 \\ 0 & 0 & -.3 & .22 & 0 & \tanh & 0\,0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0\,0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0\,0 \end{pmatrix}. \tag{7.2.5}$$

The code for the network adjacency matrix in Figure 7.4 is

```
# dtype = "object" because entries
# are floats, functions, and strings

# network adjacency matrix

w = zeros((8,8),dtype="object")

w[2][0] = w[2][1] = w[3][0] = w[3][1] = w[4][2] = 1
w[5][2] = w[4][3] = w[5][3] = w[6][4] = w[7][5] = 1

w[2][2] = relu
w[3][3] = id
w[4][4] = sigmoid
w[5][5] = tanh

# network weight matrix

w[2][0] = w[2][1] = 0.1
w[3][0] = w[3][1] = -2.0
w[4][2] = w[5][2] = -0.3
w[4][3] = w[5][3] = .22
w[6][4] = w[7][5] = 1
```

If input node 0 were a bias, we would indicate this by adding

```
# if bias
# w[0][0] = "b"
```



Here are the incoming and outgoing signals in Figures 7.5 and 7.4.

Node	Incoming	Outgoing
0	$x_0^- =$ `None`	x_0
1	$x_1^- =$ `None`	x_1
f_2	$x_2^- = w_{20}x_0 + w_{21}x_1$	$x_2 = f_2(x_2^-)$
f_3	$x_3^- = w_{30}x_0 + w_{31}x_1$	$x_3 = f_3(x_3^-)$
f_4	$x_4^- = w_{42}x_2 + w_{43}x_3$	$x_4 = f_4(x_4^-)$
f_5	$x_5^- = w_{52}x_2 + w_{53}x_3$	$x_5 = f_5(x_5^-)$
6	$x_6^- = w_{64}x_4 + w_{65}x_5$	$x_6 =$ `None`
7	$x_7^- = w_{74}x_4 + w_{75}x_5$	$x_7 =$ `None`

Table 7.1 Incoming and Outgoing signals.



The nodes may be labeled in any order. Network structure is recovered from the network adjacency matrix or network weight matrix using the code

```
def edges(w):
   W = full(w.shape,False)
   W[w != 0] = True
   fill_diagonal(W,False)
   return W

def outputs(w):
   return invert(bool(sum(edges(w),axis = 0)))

# these are the non-bias inputs

def inputs(w):
   vector = invert(bool(sum(edges(w),axis = 1)))
   return where(diagonal(w) == "b", False, vector)

# bias inputs are here
```

```
def biases(w):
   vector = full(len(w),False)
   return where(diagonal(w) == "b", True, vector)

def neurons(w):
   vector = full(len(w),True)
   vector = where(diagonal(w) == 0, False, vector)
   return where(diagonal(w) == "b", False, vector)
```

Then

```
print(inputs(w))
print(outputs(w))
print(biases(w))
print(neurons(w))
```

returns

```
#[True True False False False False False False]
#[False False False False False False  True  True]
#[False False False False False False False False]
#[False False True True True True False False]
```

In the code that follows we add and multiply weights and outgoing signals only over edges. The function `edges(w)` above encapsulates this condition.

Related functions `num_edges`, `num_inputs`, `num_outputs`, `num_biases`, `num_neurons` are in Exercise **7.2.1**.



In §7.4, we train a neural network using gradient descent: We repeatedly modify the network weight matrix W until we obtain an optimal network weight matrix W^*.

Since we do not know the optimal weight W^*, we may as well start network training with any W. In practice, to avoid effects resulting from specific choices, one starts with a random W, as follows

```
from numpy.random import default_rng
samples = default_rng().random

# w is a network weight matrix

def initial_weights(w,random = "no"):
   W = w.copy()
   if random == "yes":
      W[edges(w)] = samples(w.shape)[edges(w)]
```

```
    return W

initial_weights(w,random = "yes")
```

This code returns a random W. The default is `random = "no"`; this returns a copy of the input weight matrix.



The following code injects an input source at the input nodes,

```
# insert source values at input nodes
# also insert 1 at bias nodes

def inject_source(source,w):
    x = full(len(w),None)
    xminus = full(len(w),None)
    x[inputs(w)] = source
    x[biases(w)] = 1
    return xminus, x

source = array([1.5,2.5])
xminus, x = inject_source(source,w)
xminus, x
```

This returns

$$x^- = (\texttt{None}, \texttt{None}, \texttt{None}, \texttt{None}, \texttt{None}, \texttt{None}, \texttt{None}, \texttt{None}),$$
$$x = (1.5, 2.5, \texttt{None}, \texttt{None}, \texttt{None}, \texttt{None}, \texttt{None}, \texttt{None}).$$



By (7.2.1) and (7.2.2), the code for incoming and outgoing signals is

```
def incoming(xminus,x,w,i):
    if inputs(w)[i] or biases(w)[i]: return None
    elif xminus[i] != None: return xminus[i]
    else: return sum([ outgoing(xminus,x,w,j) * w[i][j] for j in
      ↪ range(len(w)) if edges(w)[i][j] ])

def outgoing(xminus,x,w,i):
    if outputs(w)[i]: return None
    elif x[i] != None: return x[i]
    else: return w[i][i](incoming(xminus,x,w,i))
```

Then forward propagation is coded as follows. The code doesn't depend on the node ordering. To see this, uncomment `rng.shuffle(nodes)` below.

```
from numpy.random import default_rng
rng = default_rng()

def forward_prop(xminus,x,w):
   nodes = arange(len(w))
   # rng.shuffle(nodes)
   for i in nodes:
      xminus[i] = incoming(xminus,x,w,i)
      x[i]      = outgoing(xminus,x,w,i)
   return xminus, x
```

After injecting the sample at the input nodes, we propagate x to all nodes,

```
xminus, x = inject_source(source,w)
xminus, x = forward_prop(xminus,x,w)

print(xminus)
print(x)
```

This returns the incoming and outgoing signals

$$\begin{aligned} x^- &= (\texttt{None}, \texttt{None}, 0.4, -8.0, -1.88, -1.88, 0.132, -0.954), \\ x &= (1.5, 2.5, 0.4, -8.0, 0.132, -0.954, \texttt{None}, \texttt{None}). \end{aligned} \tag{7.2.6}$$



Let $J(x^-, y)$ be a function of x^- and the target y, measuring the error between the targets y and the actual outputs x^-. We consider two error functions.

The first case is *mean square error* or *mean square loss*. For Figure 7.4, mean square error is

$$J(x^-, y) = \frac{1}{2}(x_6^- - y_0)^2 + \frac{1}{2}(x_7^- - y_1)^2, \tag{7.2.7}$$

The second case is *mean logistic error* or *mean logistic loss*. *Logistic error assumes the targets are probability vectors* p. For Figure 7.4, mean logistic error is

$$\begin{aligned} (q_0, q_1) &= \sigma(x_6^-, x_7^-), \\ J(x^-, p) &= I(p, q) = p_0 \log\left(\frac{p_0}{q_0}\right) + p_1 \log\left(\frac{p_1}{q_1}\right). \end{aligned} \tag{7.2.8}$$

Here σ and $I(p, q)$ are the softmax function and the relative information (§5.6). First σ squashes the output signal x^- into a probability vector q, then $I(p, q)$ compares q with the target probability vector p.

The code for mean error, covering both cases, is

```
from scipy.special import softmax as sigma
from scipy.stats import entropy as I

# both cases

def J(xminus, target, w, case = "square"):
   xminus = xminus[outputs(w)].astype("float")
   if case == "log": return I(target,sigma(xminus))
   else: return sum((xminus - target)**2) / 2

target = array([.427,-.288])
J(xminus, target, w)
```

The default case for J is mean square loss. For mean logistic loss, insert `case = "log"`.

With sources, targets, and weights as above, the mean square loss is $J(x^-, y) = 0.266$. This code works for any configuration of output nodes.

To match the actual outputs with the targets as closely as possible, in both cases, we seek weights that minimize J. When there is no error, $J = 0$. For mean square error, $J = 0$ implies the target y equals the output x^- at the output nodes. For mean logistic error, $J = 0$ implies the target p equals the squashed output $\sigma(x^-)$ at the output nodes.

The rest of the section deals only with mean square error. A network using mean logistic error is in Exercise **7.2.5**. Mean square error is the basis for linear regression, §7.5. Mean logistic error is the basis for logistic regression §7.6.



For Figure 7.4, since $x_6^- = w_{64}x_4$ and $x_7^- = w_{75}x_5$, we have

$$\begin{aligned}
J &= \frac{1}{2}(w_{64}x_4 - y_0)^2 + \frac{1}{2}(w_{75}x_5 - y_1)^2, \\
x_5 &= f_5(x_5^-) = f_5(w_{52}x_2 + w_{53}x_3), \\
x_4 &= f_4(x_4^-) = f_4(w_{42}x_2 + w_{43}x_3), \\
x_3 &= f_3(x_3^-) = f_3(w_{30}x_0 + w_{31}x_1), \\
x_2 &= f_2(x_2^-) = f_2(w_{20}x_0 + w_{21}x_1).
\end{aligned}$$

Therefore J is a function of the weights w_{64}, w_{75}, w_{52}, w_{53}, w_{42}, w_{43}, w_{30}, w_{31}, w_{20}, w_{21}. For gradient descent, we will need the derivatives of J with respect to these weights. Let

$$\delta_3 = \frac{\partial J}{\partial x_3^-}, \qquad \delta_4 = \frac{\partial J}{\partial x_4^-}, \qquad \delta_5 = \frac{\partial J}{\partial x_5^-},$$

and let

$$loc_3 = \frac{\partial x_3}{\partial x_3^-} = f_3'(x_3^-), \qquad loc_5 = \frac{\partial x_5}{\partial x_5^-} = f_5'(x_5^-).$$

Since w_{31} appears in x_3^-, by the chain rule,

$$\frac{\partial J}{\partial w_{31}} = \frac{\partial J}{\partial x_3^-} \cdot \frac{\partial x_3^-}{\partial w_{31}} = x_1 \cdot \delta_3.$$

Since x_3 appears in x_4^- and x_5^-, by the chain rule,

$$\delta_3 = \frac{\partial J}{\partial x_3^-} = \frac{\partial J}{\partial x_4^-}\frac{\partial x_4^-}{\partial x_3}\frac{\partial x_3}{\partial x_3^-} + \frac{\partial J}{\partial x_5^-}\frac{\partial x_5^-}{\partial x_3}\frac{\partial x_3}{\partial x_3^-}.$$

Since $x_4^- = w_{43}x_3 + w_{42}x_2$ and $x_5^- = w_{53}x_3 + w_{52}x_2$, we obtain

$$\delta_3 = \delta_4 w_{43} loc_3 + \delta_5 w_{53} loc_3 = (\delta_4 w_{43} + \delta_5 w_{53}) \cdot loc_3.$$

Similarly

$$\frac{\partial J}{\partial w_{52}} = x_2 \cdot \delta_5,$$

and

$$\delta_5 = \delta_7 w_{75} \cdot loc_5.$$

The goal is to code these formulas in general, see (7.2.12) and (7.2.13), to be used in gradient descent network training.

We already know how to compute the outgoing signals x using forward propagation. To code the derivatives δ, we use back propagation.



Since J is a function of all nodes, at each node i, we have the derivatives

$$\frac{\partial J}{\partial x_i^-}, \qquad f_j'(x_i^-), \qquad \frac{\partial J}{\partial x_i}. \tag{7.2.9}$$

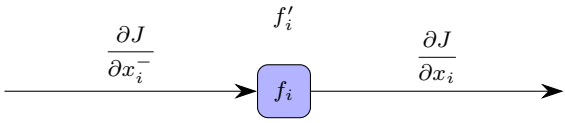


Fig. 7.7 Downstream, local, and upstream derivatives at node i.

These are the *downstream derivative*, *local derivative*, and *upstream derivative* at node j. The terminology reflects the fact that derivatives are computed backward.

From (7.2.2),

$$\frac{\partial x_i}{\partial x_i^-} = f_i'(x_i^-). \tag{7.2.10}$$

By the chain rule and (7.2.10), the key relation between these derivatives is

$$\frac{\partial J}{\partial x_i^-} = \frac{\partial J}{\partial x_i} \cdot f_i'(x_i^-), \tag{7.2.11}$$

or

$$\text{downstream} = \text{upstream} \times \text{local}.$$

```
def local(xminus,x,w,i):
   if neurons(w)[i]: return
    ↪ der_dict[w[i][i]](incoming(xminus,x,w,i))
   else: return None
```

For x, x^-, and W as above, the local derivatives are

$$(\texttt{None}, \texttt{None}, 1, 1, 0.115, 0.089, \texttt{None}, \texttt{None}).$$

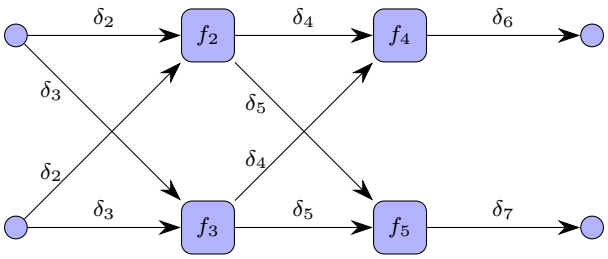


Fig. 7.8 Neural network: downstream derivatives.

Let

$$\delta_i = \frac{\partial J}{\partial x_i^-}, \qquad i = 0, 1, \ldots, d-1.$$

If i is an input node, δ_i is `None`. Then we have the *downstream gradient vector* $\delta = (\delta_0, \delta_1, \ldots, \delta_{d-1})$. Strictly speaking, we should write δ_i^- for the downstream derivatives. However, we won't be using upstream derivatives. Because of this, we will write δ_i.

Once we have the incoming vector x^- and outgoing vector x, we can differentiate J and compute the downstream derivatives with respect to each output node. For example, in Figure 7.4, there are two output nodes 6, 7, and we compute

$$\delta_6 = \frac{\partial J}{\partial x_6^-}, \qquad \delta_7 = \frac{\partial J}{\partial x_7^-}$$

as follows. Using (7.2.6) and (7.2.7),

$$\frac{\partial J}{\partial x_6^-} = (x_6^- - y_0) = -0.294.$$

Similarly,

$$\delta_7 = \frac{\partial J}{\partial x_7^-} = (x_7^- - y_1) = -0.666.$$

The code for injecting the downstream derivatives at the output nodes is

```
# returns downstream derivatives of J at output nodes

# both cases

def inject_target(xminus, target, w, case = "square"):
   delta = full(len(w),None)
   xminus = xminus[outputs(w)].astype("float")
   if case == "log": q = sigma(xminus)
   else: q = xminus
   delta[outputs(w)] = q - target
   return delta
```

When `case = "log"`, the rationale for this code is (7.6.5), and Exercise **7.2.6**.

For mean square loss, this returns

$$\delta = (\delta_0, \delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_6, \delta_7)$$
$$\delta = (\texttt{None}, \texttt{None}, \texttt{None}, \texttt{None}, \texttt{None}, \texttt{None}, -0.294, -0.666),$$



We compute δ recursively via back propagation as in §4.4. From (7.2.1) and (7.2.10),

$$\frac{\partial J}{\partial x_j^-} = \sum_{j \to i} \frac{\partial J}{\partial x_i^-} \cdot \frac{\partial x_i^-}{\partial x_j} \cdot \frac{\partial x_j}{\partial x_j^-} = \left(\sum_{j \to i} \frac{\partial J}{\partial x_i^-} \cdot w_{ij} \right) \cdot f_j'(x_j^-).$$

This yields the downstream derivative at node j,

$$\delta_j = \left(\sum_{j \to i} \delta_i \cdot w_{ij} \right) \cdot f_j'(x_j^-). \tag{7.2.12}$$

The code is

```
def downstream(xminus,x,delta,w,j):
   if inputs(w)[j] or biases(w)[j]:  delta[j] = None
   elif delta[j] != None: return delta[j]
   else:
      upstream = sum([ downstream(xminus,x,delta,w,i) * w[i][j]
    ↪ for i in range(len(w)) if edges(w)[i][j] ])
      return upstream * local(xminus,x,w,j)
```

Using this, we have the third version of back propagation,

```
def backward_prop(xminus,x,delta,w):
   nodes = arange(len(w))
   for j in nodes:
      delta[j] = downstream(xminus,x,delta,w,j)
   return delta
```

With W, sources and targets as above, the code

```
delta = inject_target(xminus, target, w)
delta = backward_prop(xminus, x, delta, w)

print(delta)
```

returns

$$\delta = (\texttt{None}, \texttt{None}, 0.0279, -0.020, -0.033, -0.059, -0.294, -0.666).$$

Note x must be computed prior to δ: first forward then backward propagation.



Above we computed the upstream, downstream, and local derivatives of J at a given node (7.2.9). Since the incoming signals x_j^- depend also on the weights w_{ij}, J also depends on w_{ij}. By (7.2.1),

$$\frac{\partial x_i^-}{\partial w_{ij}} = x_j,$$

see also Table 7.1. From this,

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial x_i^-} \cdot \frac{\partial x_i^-}{\partial w_{ij}} = \delta_i \cdot x_j.$$

We have shown

Weight Gradient of Output

If (i, j) is an edge, then

$$\frac{\partial J}{\partial w_{ij}} = \delta_i \cdot x_j. \tag{7.2.13}$$

This result is key for neural network training (§7.4).



Perceptrons can be assembled in parallel (Figure 7.1). If a network has only one layer of neurons, the network is *shallow* (Figure 7.9).

A shallow network is *dense* if all input nodes point to all neurons. A shallow network can always be assumed dense by inserting zero weights at missing edges.

Neural networks can also be assembled in series, with each component a *layer* (Figure 7.10). Usually each layer is a dense shallow network. For example, Figure 7.4 consists of two dense shallow networks in layers. We say a network is *deep* if there are two or more neuron layers.

The weight matrix W (7.2.4) is 8×8, while the weight matrices W_1, W_2 of each of the two dense shallow network layers in Figure 7.4 are 2×2.

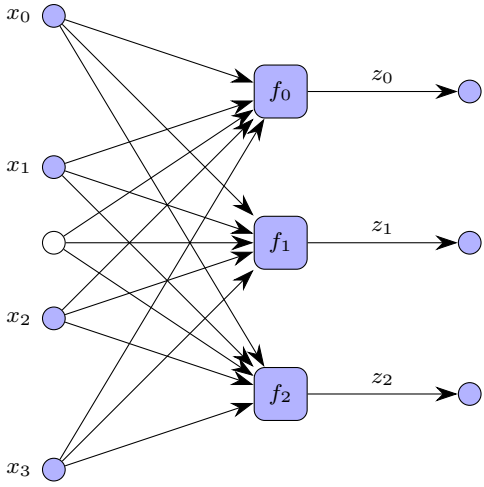


Fig. 7.9 A shallow dense layer with a bias input.

In Figure 7.9, there are 4 input nodes, 3 neurons, 3 output nodes, and a bias input. If x and z are the layer's incoming and outgoing signal vectors, then x and z are in $\mathbf{R}^4$ and $\mathbf{R}^3$ respectively. It is assumed weights are only on edges incoming to the neurons, and the weights on outgoing edges are fixed equal to 1.

Let $M = (m_{ij})$ be the weight matrix along the edges joining the four inputs to the three neurons, and let $b = (b_1, b_2, b_3)$ be the weights along the edges joining the bias input to the three neurons. Then M is a 4×3 matrix and b, the *bias vector*, is in $\mathbf{R}^3$.

With our convention, w_{ij} denotes the weight on the edge from node j to node i. With this convention, the formulas (7.2.1), (7.2.2) reduce to the matrix multiplication formulas

$$z^- = Mx + b = W\tilde{x}, \qquad z = f(Mx + b) = f(W\tilde{x}), \tag{7.2.14}$$

with $W = (M, b)$ and $\tilde{x} = \begin{pmatrix} x \\ 1 \end{pmatrix}$ the augmented sample.

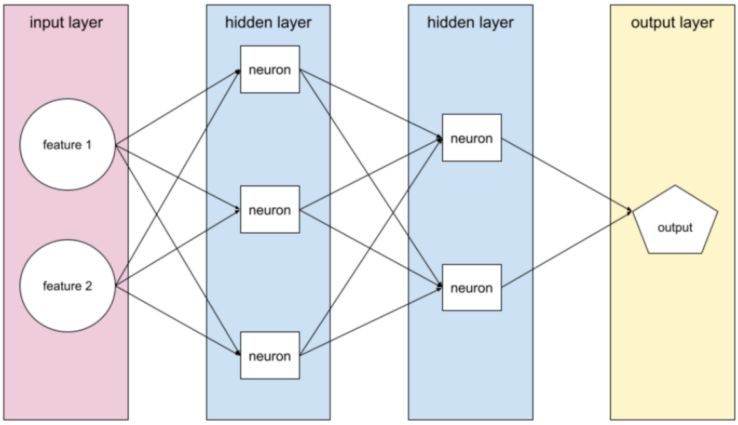


Fig. 7.10 Layered neural network [11].

From this point of view, a dense shallow network is a vector-valued perceptron, and the layered network in Figure 7.10 is a composition of vector-valued perceptrons in series.

Exercises

Exercise 7.2.1 Write code returning `num_inputs(w)`, `num_edges(w)`, `num_outputs(w)`, `num_biases(w)`, `num_neurons(w)`.

Exercise 7.2.2 Show that a dataset lies in a hyperplane (4.5.7) in the sample space iff the augmented dataset (7.2.3) does not span the augmented sample space (see §2.7).

Exercise 7.2.3 With the nodes in Figure 7.4 re-ordered from 0, 1, 2, 3, 4, 5, 6, 7 to 4, 0, 3, 7, 1, 6, 5, 2, rebuild the network matrix.

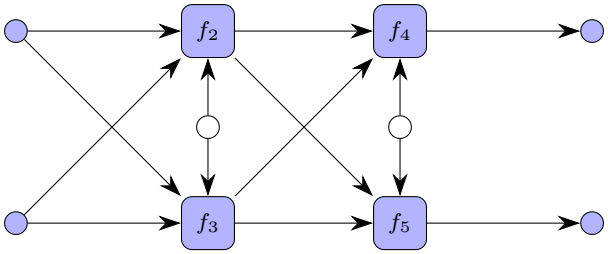


Fig. 7.11 Neural network with biases

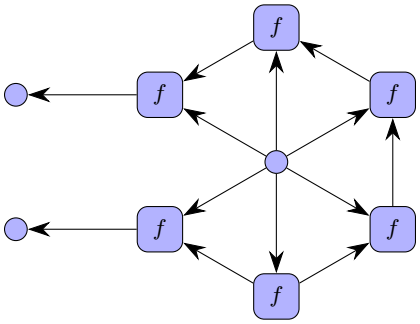


Fig. 7.12 A network with six neurons, two outputs, and one input.

Exercise 7.2.4 Compute the incoming signals x^-, outgoing signals x, and downstream derivatives δ for the network in Figure 7.11, using sources, targets, weights, and activations as in the text. Here J is mean square loss (7.2.7).

Exercise 7.2.5 In Figure 7.12, $f(x) = \tanh(x)$, the source is 1.0, the target is $(.427, .573)$, and all weights are $+1$. Assume J is mean logistic loss (7.2.8). Compute the incoming signals x^-, outgoing signals x, and downstream derivatives δ.

Exercise 7.2.6 Suppose we have a neuron followed by an output node as in Figure 7.13. Fix a target probability $0 \leq p \leq 1$, and let J be the relative

information $I(p,q)$ (4.2.9). If σ is the logistic function (5.2.16), show the upstream, local, and downstream derivatives at the neuron are

$$\frac{\partial J}{\partial q} = \frac{q-p}{q(1-q)}, \qquad \sigma' = \sigma(1-\sigma), \qquad \delta = \frac{\partial J}{\partial q^-} = q - p.$$

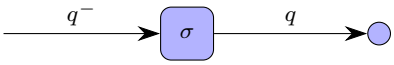


Fig. 7.13 Downstream, local, and upstream derivatives at termination.

7.3 Gradient Descent

Let $f(w)$ be a scalar function of a vector $w = (w_1, w_2, \ldots, w_d)$ in $\mathbf{R}^d$. A basic problem is to minimize $f(w)$, that is, to find or compute a w^* satisfying

$$f(w) \geq f(w^*), \qquad \text{for every } w.$$

Such a w is a *minimizer*.

This goal is so general, that any insight one provides towards this goal is widely useful in many settings. The setting we have in mind is $f = J$, where J is the mean error from §7.1.

Usually $f(w)$ is a measure of cost or lack of compatibility. Because of this, $f(w)$ is called the *loss function* or *error function*.

A neural network is a black box with inputs x and outputs y, depending on unknown *weights* w. To *train* the network is to select weights w in response to training data (x, y). The optimal weights w^* are selected as minimizers of a loss function $f(w)$ measuring the error between predicted outputs and actual outputs, corresponding to given training inputs.

For neural network training, the most common loss functions are the mean square loss (7.5.4) and the mean logistic loss (7.6.2).

From §4.3, if the loss function $f(w)$ is continuous and proper, there is a global minimizer w^*. If $f(w)$ is in addition strictly convex, w^* is unique (§4.5). When this happens, if the gradient of the loss function is $g = \nabla f(w)$, then w^* is the unique point satisfying $g^* = \nabla f(w^*) = 0$.



Let $g(w)$ be any function of a scalar variable w. From the definition of derivative (4.1.3), if b is close to a, we have the approximation

$$\frac{g(b) - g(a)}{b - a} \approx g'(a).$$

Inserting $a = w$ and $b = w^+$,

$$g(w^+) \approx g(w) + g'(w)(w^+ - w).$$

Assume w^* is a root of $g(w) = 0$, so $g(w^*) = 0$. If w^+ is close to w^*, then $g(w^+)$ is close to zero, so

$$0 \approx g(w) + g'(w)(w^+ - w).$$

Solving for w^+,

$$w^+ \approx w - \frac{g(w)}{g'(w)}.$$

Since the global minimizer w^* satisfies $f'(w^*) = 0$, we insert $g(w) = f'(w)$ in the above approximation,

$$w^+ \approx w - \frac{f'(w)}{f''(w)}.$$

This leads to *Newton's method* of computing approximations w_0, w_1, w_2, ... of w^* using the recursion

$$w_{n+1} = w_n - \frac{f'(w_n)}{f''(w_n)}, \qquad n = 1, 2, \ldots$$

Because calculating $f''(w)$ is computationally expensive, *first-order descent methods* replace the second derivative terms $f''(w_n)$ by constants, known as learning rates.

In the multi-variable case, Newton's method becomes

$$w_{n+1} = w_n - D^2 f(w_n)^{-1} \nabla f(w_n), \qquad n = 1, 2, \ldots,$$

and the second-derivative term is even more expensive to compute.

These first-order methods, collectively known as *gradient descent*, are the subject of this chapter.



Here is code for Newton's method.

```
from numpy import *

def newton(loss, grad, curv, w, num_iter):
   g = grad(w)
   c = curv(w)
   trajectory = array([[w],[loss(w)]])
   for _ in range(num_iter):
      w -= g/c
```

```
        trajectory = column_stack([trajectory,[w,loss(w)]])
        g = grad(w)
        c = curv(w)
        if allclose(g,0): break
    return trajectory
```

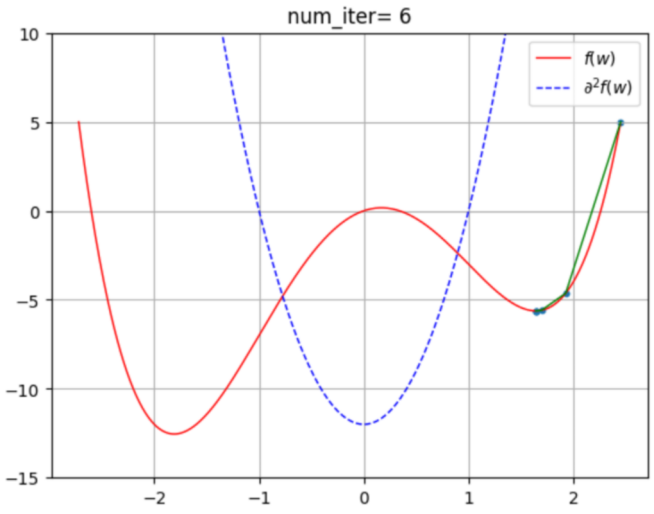


Fig. 7.14 Double well newton descent.

When applied to the function

$$f(w) = w^4 - 6w^2 + 2w,$$

the code returns `trajectory`

```
def loss(w): return w**4 - 6*w**2 + 2*w     # f(w)
def grad(w): return 4*w**3 - 12*w + 2       # f'(w)
def curv(w): return 12*w**2 - 12            # f''(w)

u0 = -2.72204813
w0 = 2.45269774
num_iter = 20
trajectory = newton(loss, grad, curv, w0, num_iter)
```

which can be plotted using the code

```
from matplotlib.pyplot import *

def plot_descent(a, b, loss, curv, delta, trajectory):
   w = arange(a,b,delta)
   plot(w,loss(w),color = 'red',linewidth = 1)
   plot(w,curv(w),"--",color = 'blue',linewidth = 1)
   plot(*trajectory,color = 'green',linewidth = 1)
   scatter(*trajectory,s = 10)
   title("num_iter= " + str(len(trajectory.T)))
   grid()
   show()
```

Then

```
ylim(-15,10)
delta = .01
plot_descent(u0, w0, loss, curv, delta, trajectory)
```

returns Figure 7.14.



A *descent sequence* is a sequence w_0, w_1, w_2, ... where the loss function decreases

$$f(w_0) \geq f(w_1) \geq f(w_2) \geq \dots .$$

In a descent sequence, the point after the current point $w = w_n$ is the successive point $w^+ = w_{n+1}$, and the point before the current point is the previous point $w^- = w_{n-1}$. Then $(w^-)^+ = w = (w^+)^-$.

Recall (§4.3) the gradient $\nabla f(w)$ at a given point w is the direction of greatest increase of the function, starting from w. Because of this, it is natural to construct a descent sequence by moving, at any given w, in the direction $-\nabla f(w)$ opposite to the gradient.

A *gradient descent sequence* is a descent sequence w_0, w_1, w_2, ... where each successive point w^+ is obtained from the previous point w by moving in the direction opposite to the gradient $\nabla f(w)$ at w,

Basic Gradient Descent Step (GD)

$$w^+ = w - t\nabla f(w). \tag{7.3.1}$$

The step size t, which determines how far to go in the direction opposite to the gradient, is the *learning rate* or *time step*. Starting with an initial weight w_0, this generates a descent sequence as follows,

$$\begin{aligned} w_1 = (w_0)^+ &= w_0 - t_1 \nabla f(w_0), \\ w_2 = (w_1)^+ &= w_1 - t_2 \nabla f(w_1), \\ w_3 = (w_2)^+ &= w_2 - t_3 \nabla f(w_2), \\ \dots &= \dots \end{aligned} \tag{7.3.2}$$

The learning rate may vary with the steps, $t_1, t_2, \dots$, or the learning rate may be a fixed number t. Under reasonable assumptions, a gradient descent sequence with a constant learning rate t converges to a minimizer. However, for stochastic gradient descent, to obtain convergence to w^*, the learning rate must vary, see below.



Let us unpack (7.3.1), so we understand how it applies to weights in networks (§4.4). In a neural network, weights $w_1, w_2, \dots$ are attached to edges, and the final outputs are combined into a loss function. As a result, the loss function is a function of the weights,

$$f(w) = f(w_1, w_2, \dots).$$

In (7.3.1), $w = (w_1, w_2, \dots)$ is the weight vector, consisting all of weights combined into a single vector. By the gradient formula (4.3.2), (7.3.1) is equivalent to

$$\begin{aligned} w_1^+ &= w_1 - t \frac{\partial f}{\partial w_1}, \\ w_2^+ &= w_2 - t \frac{\partial f}{\partial w_2}, \\ \dots &= \dots . \end{aligned}$$

In other words,

Each Weight is Updated Separately

To update a weight in a specific edge using gradient descent, one needs only the derivative of the loss function relative to the weight on that specific edge.



In practice, the learning rate is selected by trial and error. Which learning rate does the theory recommend?

Given an initial point w_0, the *sublevel set at* w_0 (see §4.5) consists of all points w where $f(w) \le f(w_0)$. Only the part of the sublevel set that is connected to w_0 counts.

In Figure 7.15, the sublevel set at w_0 is the interval $[u_0, w_0]$. The sublevel set at w_1 is the interval $[b, w_1]$. Notice we do not include any points to the left of b in the sublevel set at w_1, because points to the left of b are separated from w_1 by the gap at the point b.

Suppose the second derivative $D^2 f(w)$ is never greater than a constant L on the sublevel set. This means

$$D^2 f(w) \le L, \qquad \text{on } f(w) \le f(w_0), \tag{7.3.3}$$

in the sense the eigenvalues of $D^2 f(w)$ are never greater than L.

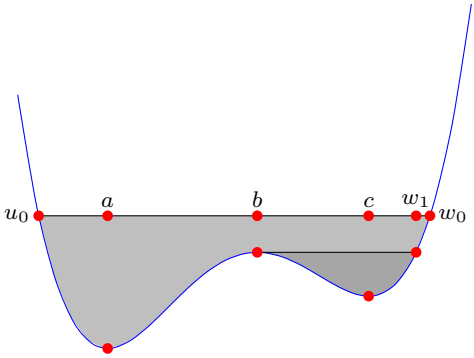


Fig. 7.15 Double well function and sublevel sets at w_0 and at w_1.

For example, if $f(w) = w \cdot Qw/2$, then we can take L equal to the top eigenvalue λ of Q. By choosing $Q = A^t A$, for $f(w) = |Aw|^2/2$, we can take L equal to the square σ^2 of the top singular value σ of A.

Because the second derivative is the derivative of the first derivative, $D^2 f(w)$ measures how fast the gradient $\nabla f(w)$ changes from point to point. From this point of view, $D^2 f(w)$ is a measure of the curvature of the function $f(w)$, and (7.3.3) says the rate of change of the gradient is never greater than L.

Given such a bound L on the curvature, If the learning rate t is no larger than $1/L$, we say we are doing *short-step* gradient descent. Then we have

Short Step Gradient Descent

Let L be as above and w^+ as in (7.3.1). If $t \le 1/L$, then

$$f(w^+) \le f(w) - \frac{t}{2}|\nabla f(w)|^2. \tag{7.3.4}$$

To see this, fix w and let S be the sublevel set $\{w' : f(w') \le f(w)\}$. Since the gradient pushes f down, for $t > 0$ small, w^+ stays in S. Insert $x = w^+$

and $a = w$ into the right half of (4.5.15) and simplify. This leads to

$$f(w^+) \le f(w) - t|\nabla f(w)|^2 + \frac{t^2 L}{2}|\nabla f(w)|^2.$$

Since $tL \le 1$ when $0 \le t \le 1/L$, we have $t^2 L \le t$. This derives (7.3.4).

The curvature of the loss function and the learning rate are inversely proportional. Where the curvature of the graph of $f(w)$ is large, the learning rate $1/L$ is small, and gradient descent proceeds in small time steps.



When the sublevel set is bounded, there is a bound L satisfying (7.3.3). From §4.3, the sublevel set is bounded when $f(w)$ is proper: Large $|w|$ implies high cost $f(w)$. The graphs in Figures 4.3, 4.4, 7.15, are proper.

In practice, when the loss function is not proper, it is modified by an extra term that forces properness. This is called *regularization*. If the extra term is proportional to $|w|^2$, it is *ridge* regularization, and if the extra term is proportional to $|w|$, it is *LASSO* regularization.



Now let w_0 w_1, w_2 ... be a short-step gradient descent sequence, with short-step learning rates t_n all equal to $1/L$. By (7.3.4), w_n remains in the sublevel set $f(w) \le f(w_0)$. If this sublevel set is bounded, w_n subconverges to a limit w^* (§A.8). Inserting $w^+ = w_n$, $t = t_n$, $w = w_{n-1}$ in (7.3.4),

$$f(w_n) \le f(w_{n-1}) - \frac{t_n}{2}|\nabla f(w_{n-1})|^2.$$

Since $f(w_n)$ and $f(w_{n+1})$ both converge to $f(w^*)$ and $\nabla f(w_n)$ converges to $\nabla f(w^*)$,

$$f(w^*) \le f(w^*) - \frac{1}{2L}|\nabla f(w^*)|^2.$$

Since this implies $\nabla f(w^*) = 0$, we have derived the following.

Gradient Descent Converges to a Critical Point

Fix any initial weight w_0 and let L be as above. If the short-step gradient descent sequence starting from w_0, with constant learning rate $t = 1/L$, converges to some point w^*, then w^* is a critical point of the loss function. In particular, if the loss function is proper, the short-step gradient descent sequence starting from w_0 converges to a critical point of the loss function.

For example, let $f(w) = w^4 - 6w^2 + 2w$ (Figures 7.14, 7.15, 7.16). Then

$$f'(w) = 4w^3 - 12w + 2, \qquad f''(w) = 12w^2 - 12.$$

Thus the inflection points (where $f''(w) = 0$) are ± 1 and, in Figure 7.15, the critical points are a, b, c.

Let u_0 and w_0 be the points satisfying $f(w) = 5$ as in Figure 7.16. Then $u_0 = -2.72204813$ and $w_0 = 2.45269774$, so $f''(u_0) = 76.914552$ and $f''(w_0) = 60.188$. Thus we may choose $L = 76.914552$. With this L, the short-step gradient descent starting at w_0 is guaranteed to converge to one of the three critical points. In fact, the sequence converges to the right-most critical point c (Figure 7.16).

This exposes a flaw in basic gradient descent. Gradient descent may converge to a local minimizer, and miss the global minimizer. In §7.9, modified gradient descent will address some of these shortcomings.

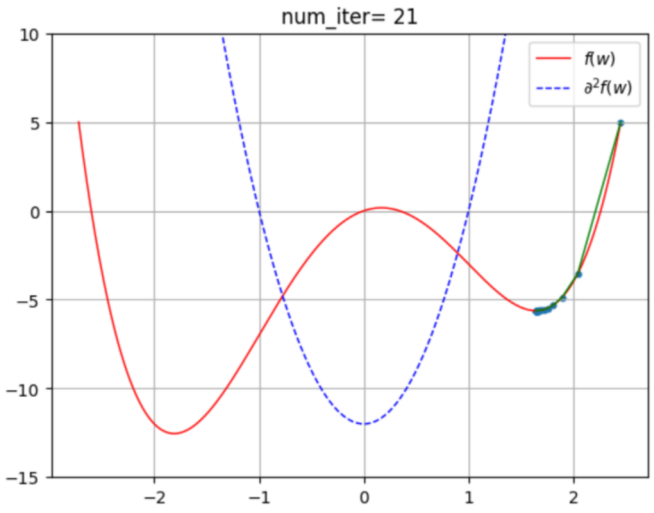


Fig. 7.16 Double well gradient descent.

The code for gradient descent is

```
from numpy import *
from matplotlib.pyplot import *

def gd(loss, grad, w, learning_rate, num_iter):
   g = grad(w)
   trajectory = array([[w],[loss(w)]])
   for _ in range(num_iter):
      w -= learning_rate * g
      trajectory = column_stack([trajectory,[w,loss(w)]])
      g = grad(w)
```

```
        if allclose(g,0): break
    return trajectory
```

When applied to the double well function $f(w)$,

```
u0 = -2.72204813
w0 = 2.45269774
L = 76.914552
learning_rate = 1/L
num_iter = 100
trajectory = gd(loss, grad, w0, learning_rate, num_iter)

ylim(-15,10)
delta = .01
plot_descent(u0, w0, loss, curv, delta, trajectory)
```

the code returns Figure 7.16.



We now turn to an important modification of gradient descent, *stochastic gradient descent* (SGD). Here the loss function depends not only on a weight w, but also on a source-target sample vector v. In this setting, given a dataset $v_1, v_2, \ldots, v_N$, the goal is, following §7.1, to minimize the mean loss

$$f(w) = \frac{1}{N}\sum_{k=1}^{N} F(w, v_k). \tag{7.3.5}$$

The mean loss $f(w)$ is an average of sampled loss functions $F(w, v_k)$.

To apply gradient descent to (7.3.5), we must compute the gradient

$$\nabla f(w) = \frac{1}{N}\sum_{k=1}^{N} \nabla F(w, v_k). \tag{7.3.6}$$

However, in practice, the computational cost of evaluating the sums (7.3.5) and (7.3.6) is high. For example, for the MNIST dataset, $N = 60,000$. When training neural networks, these sums must be evaluated repeatedly, at each weight updating, resulting in prohibitive cost.

Because of this, a more feasible approach is to avoid evaluating these sums, select samples $v_1, v_2, \ldots$ randomly from the dataset, and generate a descent sequence according to

$$w_n = w_{n-1} - t_n \nabla F(w_{n-1}, v_n), \qquad n = 1, 2, \ldots. \tag{7.3.7}$$

It turns out that with the correct tuning of learning rates, this modified descent sequence *does* converge to a minimizer of the mean loss (7.3.5).

This is a reflection of the robustness of gradient descent: Even if an approximate or noisy gradient is used to compute a descent sequence, under appropriate assumptions, there is convergence to a minimizer.

However, because the gradient in (7.3.7) is only an approximation to the true gradient $\nabla f(w)$, the convergence here is slower.

The theoretical basis for this convergence was established in the 1951 paper of Robbins-Monroe [28]. Their result is a type of law of large numbers for randomly sampled minimization problems.

To view this with the proper perspective, by the discussion (5.3.9), the mean loss (7.3.5) is an expectation of a random variable: If V is a random variable with equally likely values $v_1, v_2, \ldots, v_N$, then the mean loss is an expectation

$$f(w) = E(F(w, V)).$$

Let $V_1, V_2, \ldots$ be an i.i.d. sequence of samples of V, fix an initial weight w_0, and generate a descent sequence by

$$W_n = W_{n-1} - t_n \nabla F(W_{n-1}, V_n), \qquad W_0 = w_0. \tag{7.3.8}$$

Because V_n are random variables, so are W_n.

Here we take the simplest possible example, that of minimizing a parabola. In §7.8, we derive SGD for strongly convex mean loss functions.

Fix a scalar w^*. We assume the samples v are scalars, and the sampled loss

$$F(w, v) = \frac{1}{2}(w - w^* - v)^2. \tag{7.3.9}$$

is a scalar parabola minimized at $w^* + v$. Then

$$f(w) = E(F(w, V)) = \frac{1}{2}(w - w^*)^2 - (w - w^*)E(V) + \frac{1}{2}E(V^2).$$

If V has mean zero and variance one, then $f(w) = (w - w^*)^2/2 + 1/2$, and $f(w)$ is minimized at w^*. Here is the SGD result.

Stochastic Gradient Descent (SGD) for Parabolas

Assume V is a standard scalar random variable. If $V_1, V_2, \ldots$ is an i.i.d. sequence of samples of V, let $W_1, W_2, \ldots$ be given by (7.3.8), where $F(w, v)$ is given by (7.3.9). If the learning rates satisfy

$$t_1 + t_2 + \cdots = \infty, \qquad \text{and} \qquad t_1^2 + t_2^2 + \cdots < \infty, \tag{7.3.10}$$

then

$$E\left(|W_n - w^*|^2\right) \to 0, \qquad n \to \infty. \tag{7.3.11}$$

Again, the point of SGD is to converge to the minimizer *without* evaluating the expectation in the mean loss.

The standard learning rates for SGD are the *harmonic learning rates*

$$t_1 = \frac{\alpha}{1+\beta}, \quad t_2 = \frac{\alpha}{2+\beta}, \quad t_3 = \frac{\alpha}{3+\beta}, \quad \ldots, \tag{7.3.12}$$

where α and β are constants. Using standard manipulations, these learning rates satisfy (7.3.10), see Exercises **A.7.4** and **A.7.6**.

To derive the SGD convergence result for parabolas, start by computing the gradient

$$\nabla F(w, v) = F'(w, v) = w - w^* - v.$$

Then (7.3.8) becomes

$$W_n = W_{n-1} - t_n(W_{n-1} - w^* - V_n),$$

which simplifies to a convex combination

$$W_n - w^* = (1 - t_n)(W_{n-1} - w^*) + t_n V_n. \tag{7.3.13}$$

Taking expectations, since $E(V_n) = 0$,

$$E(W_n) - w^* = (1 - t_n)\left(E(W_{n-1}) - w^*\right).$$

Iterating this equality leads to

$$E(W_n) - w^* = p_n \cdot (w_0 - w^*), \qquad n = 1, 2, \ldots,$$

with p_n the product

$$p_n = \prod_{k=1}^{n}(1 - t_k) = (1 - t_1)(1 - t_2)\ldots(1 - t_n). \tag{7.3.14}$$

By Exercise **A.7.7** with $m = 1/2$ and $L = 0$, p_n converges to zero, hence the means $E(W_n)$ converge to w^* as $n \to \infty$.

Since W_{n-1} depends only on $V_1, V_2, \ldots, V_{n-1}$, the random variables W_{n-1} and V_n are independent. Taking variances in (7.3.13), and using $Var(W_n) = Var(W_n - w^*)$,

$$Var(W_n) = (1 - t_n)^2\, Var(W_{n-1}) + t_n^2\, Var(V_n).$$

Assume[1] $t_n \leq 1$. Since $Var(V_n) = 1$,

$$Var(W_n) \leq (1 - t_n)\, Var(W_{n-1}) + t_n^2.$$

Since $Var(W_0) = Var(w_0) = 0$, iterating this inequality leads to

$$Var(W_n) \leq s_n, \qquad n = 1, 2, \ldots,$$

[1] By (7.3.10), $t_n \to 0$, so this holds for n large.

with s_n the sum

$$s_n = \sum_{k=1}^{n} t_k^2 \cdot \frac{p_n}{p_k}.$$

To show $s_n \to 0$, use the second condition in (7.3.10), as follows. Since

$$\sum_{k=1}^{\infty} t_k^2 = \lim_{N\to\infty} \sum_{k=1}^{N} t_k^2$$

is finite, the limit of the error

$$e_N = \sum_{k=N+1}^{\infty} t_k^2 = \sum_{k=1}^{\infty} t_k^2 - \sum_{k=1}^{N} t_k^2,$$

as $N \to \infty$, is zero. Now fix a cut-off N and break s_n into two parts, $n \le N$ and $n > N$,

$$s_n = \sum_{k=1}^{N} t_k^2 \cdot \frac{p_n}{p_k} + \sum_{k=N+1}^{n} t_k^2 \cdot \frac{p_n}{p_k}.$$

Since $p_n \le p_k$ when $n \ge k$,

$$s_n \le \left(\sum_{k=1}^{N} \frac{t_k^2}{p_k}\right) p_n + \sum_{k=N+1}^{n} t_k^2 \le \left(\sum_{k=1}^{N} \frac{t_k^2}{p_k}\right) p_n + e_N.$$

With N fixed, let $n \to \infty$. Since $p_n \to 0$,

$$0 \le s_\infty = \lim_{n\to\infty} s_n \le e_N.$$

By choosing N large enough, we can make e_N arbitrarily small. Hence $s_\infty = 0$, and $Var(W_n) \to 0$.

Using the triangle inequality, combine $E(W_n) \to w^*$ and $Var(W_n) \to 0$ to complete the proof of (7.3.11).

In §7.8, we examine GD and SGD in the context of strongly convex loss functions.

Exercises

Exercise 7.3.1 Let A be a matrix and b a vector with the residual

$$f(w) = |Aw - b|^2$$

well-defined. Starting with w_0 in the row space of A, show gradient descent, with constant short-step learning rate, converges to a unique w^* in the row space of A. (Exercise **4.3.3**)

Exercise 7.3.2 Continuing the previous exercise, starting with any w_0, show gradient descent converges to a unique w^* with $w^* - w_0$ in the row space of A.

Exercise 7.3.3 Let $f(w) = \frac{1}{2}aw^2+bw+c$, $a > 0$, be a convex parabola. Then $f'(w) = aw + b$, and the minimizer w^* satisfies $f'(w^*) = 0$, so $aw^* + b = 0$. If w_0 is any starting weight, and $t > 0$ is the learning rate, the gradient descent sequence (7.3.1) is

$$w_{n+1} = w_n - tf'(w_n), \qquad n = 0, 1, 2, \ldots.$$

Show by direct calculation

$$w_{n+1} - w^* = (1 - at)(w_n - w^*), \qquad n = 0, 1, 2, \ldots,$$

so

$$w_n - w^* = (1 - at)^n(w_0 - w^*), \qquad n = 1, 2, \ldots.$$

If the learning rate t is small enough so that $at \leq 1$, this shows w_n converges to w^* as $n \to \infty$.

7.4 Network Training

A neural network with network matrix W defines an *input-output map*

$$source \qquad \to \qquad target.$$

Here the inputs are the sources at the input nodes, and the outputs are the targets at the output nodes.

Given sources x and targets y, we seek to modify W so that the output propagated through the network starting from x equals y. This is *network training.*

The weights are modified using gradient descent (§7.3). If J measures the error between the network outputs and the targets, W is updated to W^+ using (7.3.1),

Weight Gradient Descent

If t is the learning rate, the updating equation is

$$W^+ = W - t\nabla_W J. \tag{7.4.1}$$

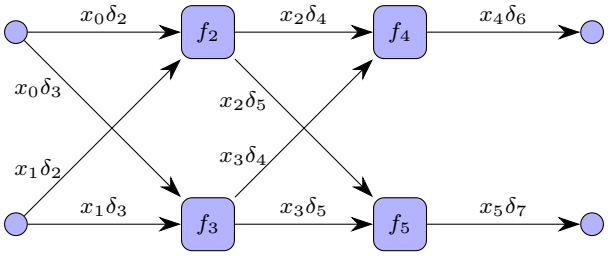


Fig. 7.17 Neural network: weight gradients.

How do we compute $\nabla_W J$? From (7.2.13),

The Weight Gradient is a Tensor Product

The weight gradients are

$$\frac{\partial J}{\partial w_{ij}} = \delta_i x_j,$$

or

$$\nabla_W J = \delta \otimes x. \tag{7.4.2}$$

For the network in Figure 7.4, the weight gradients are as in Figure 7.17. To write the code for the weight gradient, first we import the previous section's code (see Appendix B)

```
%%capture

import import_ipynb
from neural_networks import *
```

then we code the tensor product

```
def tensor(delta,x):
   x[x == None] = 0
   delta[delta == None] = 0
   return outer(delta,x)
```

We combine the input source and output target into a single vector

```
source = array([ 1.5, 2.5 ])
target = array([ 0.427, -0.288 ])
vector = hstack([source, target])
vector
```

We do this to handle datasets with multiple samples below. We also count inputs and outputs using

```
def num_inputs(w): return sum(inputs(w))
def num_outputs(w): return sum(outputs(w))
def num_biases(w): return sum(biases(w))
def num_neurons(w): return sum(neurons(w))

num_inputs(w), num_outputs(w), num_biases(w), num_neurons(w)
```

There are two cases, mean square error and mean logistic error. The default case is mean square error. The weight gradient code is

```
def single_weight_gradient(vector, w, case = "square"):
   num_in = num_inputs(w)
   num_out = num_outputs(w)
   source, target = vector[:num_in], vector[-num_out:]
   xminus, x = inject_source(source, w)
   xminus, x = forward_prop(xminus, x, w)
   delta = inject_target(xminus, target, w, case = case)
   delta = backward_prop(xminus, x, delta, w)
   loss = J(xminus, target, w, case = case)
   # gradJ has same shape as w
   gradJ = tensor(delta,x)
   # cast loss to array same shape as w
   loss = full(w.shape, loss)
   return array([loss, gradJ])
```

For mean square error, it's `case = "square"`. For mean logistic error, it's `case = "log"`. In this code, for convenience below, the scalar `loss` is cast to an array with shape `w.shape`:

If the shape of `w` is (d, d), then the shape of `gradJ` is (d, d) and the shape of `single_weight_gradient(vector,w)` is $(2, d, d)$.

After this, weights are updated as in (7.4.1),

```
# update only weights on edges

def update_weights(w, gradJ, learning_rate):
   w[edges(w)] -= learning_rate * gradJ[edges(w)]
   return w
```



To save space, below `lr` is the learning rate, and `bs` is the minibatch size. Also below `random = "no"` determines whether or not to randomize the initial W (see §7.2 for `initial_weights`).

```
def single_sample_training(vector, w, lr, iters, random = "no",
    ↪ case = "square"):
   W = initial_weights(w,random = random)
   trajectory = [ ]
   for iter in range(iters):
      loss, gradJ = single_weight_gradient(vector, W, case = case)
      W = update_weights(W, gradJ, lr)
      # recover scalar loss
      loss = loss[0,0]
      if isclose(0,loss): break
      trajectory.append(loss)
   return W, trajectory
```

Here `iters` is the maximum number of iterations, and the iterations terminate if the loss J is close to zero.

Starting with the network weight matrix (7.2.5), the code

```
lr = .045
iters = 100
wstar, trajectory = single_sample_training(vector, w, lr, iters)
```

stops after `len(trajectory)= 45` iterations, and returns an optimal weight matrix W^*.

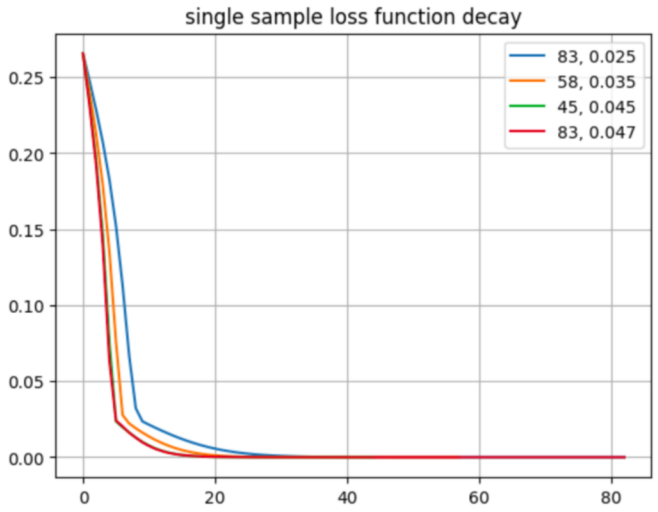


Fig. 7.18 Loss decay as learning rate varies: single sample training.

Forward propagation using W^* results in outputs

$$x_6^- = 0.4269028859959264, \qquad x_7^- = -0.28799475228709476,$$

in close agreement with `target = array([0.427,-0.288])`. Since you may be starting your network training with a different W, your results may be different.

The loss `trajectory` can be plotted using the code

```
from matplotlib.pyplot import *

for lr in [.025,.035, .045,.047]:
   wstar, trajectory = single_sample_training(vector, w, lr,
     ↪ iters)
   n = len(trajectory)
   label = str(n) + ", " + str(lr)
   plot(range(n), trajectory,label = label)

title("single sample loss function decay")
grid()
legend()
show()
```

resulting in Figure 7.18.



The convergence here is surprisingly easy to attain. However, this is a mirage. It is a reflection of overfitting, in the sense that we trained the weights to obtain the input-output map corresponding to a single sample: There is no reason the trained weights reproduce the input-output map for other samples.

To make this point explicit, focus on the output node at top right in Figure 7.17, let y be the target at node 6, and let

$$x = x_4, \qquad w = w_{46}.$$

Then

$$x_6^- = wx, \qquad \delta_6 = \frac{\partial J}{\partial x_6^-} = wx - y, \qquad \frac{\partial J}{\partial w_{46}} = x(wx - y).$$

Starting with weight w and learning rate t, the updated weight is

$$w^+ = w - tx(wx - y) = w(1 - tx^2) + txy.$$

Since $f_4 = \sigma$, x is close to 1 as soon as there is a substantial incoming x_4^-. If we start the training with the network adjacency matrix, then all weights equal 1 and $x_4^- = 8$, leading to x_4 close to 1. Taking $t = 1$ yields $w^+ = y$, which implies we have convergence in two iterations. Only the weight w played a substantial role, the weights at the other edges were minimally involved.

When we train against several samples, we must match several equations, and all weights must contribute. Only after we train the weights repeatedly against all samples in a training dataset, can we hope to achieve training with some predictive power.



We repeat weight gradient descent when the loss function is the sum J of loss functions $J(x_k^-, y_k)$ over all samples x_1, x_2, ..., x_N in a dataset, with corresponding targets y_1, y_2, ..., y_N. To this end, we set

$$J = \frac{1}{N}\sum_{k=1}^{N} J(x_k^-, y_k).$$

By (7.3.6), the weight gradient $\nabla_W J$ equals the average of the weight gradients $\nabla_W J(x_k^-, y_k)$.

As with single sample training, we assume `dataset` consists of sample-target vectors. We use `map` to parallelize the computation.

```
def batch_weight_gradient(dataset,w, case = "square"):
   gradient_map = lambda vector: single_weight_gradient(vector,
    ↪ w, case = case)
   weight_gradients = array(list(map(gradient_map, dataset)))
   return mean(weight_gradients, axis = 0)
```

Here `weight_gradients` has shape $(N, 2, d, d)$, and `batch_weight_gradient` has shape $(2, d, d)$, as before.

If the `dataset` consists of a single sample `array([vector])`, then

```
batch_weight_gradient(dataset,w)
```

is identical to

```
single_weight_gradient(vector,w).
```

An *iteration* is the updating of the weight matrix against a single sample. An *epoch* is the updating of the weight matrix against all samples in the dataset. The distinction will become clearer below.

```
def batch_sample_training(dataset,w,lr,epochs,random = "no", case
    ↪ = "square"):
   W = initial_weights(w,random = random)
   trajectory = [ ]
   for epoch in range(epochs):
      loss, gradJ = batch_weight_gradient(dataset, W, case = case)
      W = update_weights(W,gradJ,lr)
      # recover scalar loss
      loss = loss[0,0]
```

```
      if isclose(0,loss): break
      trajectory.append(loss)
   return W, trajectory
```

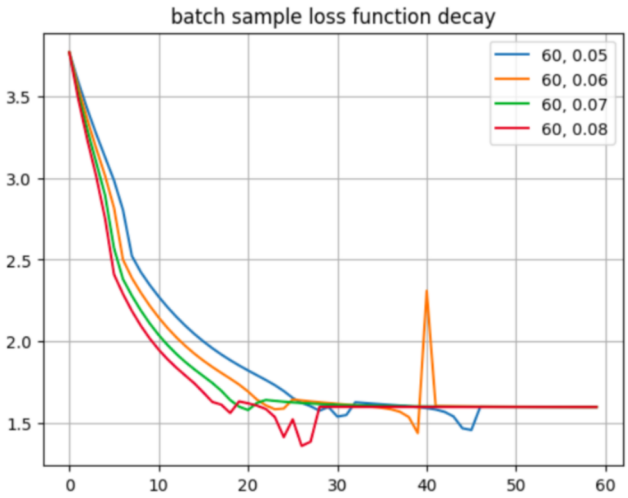


Fig. 7.19 Loss decay as learning rate varies: batch sample training.

The code for Figure 7.19 is

```
sources = array([ [ 1.5, 2.5 ], [1.2,3.1], [7.1,8.2] ])
targets = array([ [ 0.427, -0.288 ], [1.1,-.4], [1.2,3.4 ] ])
dataset = hstack([sources, targets])

epochs = 60

from matplotlib.pyplot import *

for lr in [.05,.06,.07,.08]:
   wstar, trajectory = batch_sample_training(dataset,w,lr,epochs)
   n = len(trajectory)
   label = str(n) + ", " + str(lr)
   plot(range(n),trajectory,label = label)

title("batch sample loss function decay")
grid()
legend()
show()
```

In batch sample training, we updated the weight gradients only after evaluating the gradients against all samples in the dataset. An alternate approach is to use stochastic gradient descent (§7.3) and update the weights with each sample. To prevent cycles, this is coupled with a random shuffling of the dataset at the start of each epoch. This leads to the code

```
from numpy.random import default_rng
rng = default_rng()

def stochastic_sample_training(dataset,w,lr,epochs,random = "no",
    ↪ case = "square"):
   W = initial_weights(w,random = random)
   trajectory = [ ]
   for n,epoch in enumerate(range(epochs),start = 1):
      losses = [ ]
      rng.shuffle(dataset)
      for vector in dataset:
         loss, gradJ = single_weight_gradient(vector, W, case =
    ↪ case)
         W = update_weights(W,gradJ,lr/n)
         # append scalar loss
         losses.append(loss[0,0])
      loss = mean(losses)
      if isclose(0,loss): break
      trajectory.append(loss)
   return W, trajectory
```

The above two dataset training approaches, batch sample and stochastic sample, are two extremes. A third approach, intermediate between the two, is to divide the dataset into minibatches, each with minibatch size `bs`, and do batch sample training on each minibatch. This is *minibatch sample training*. Here is the code.

```
from numpy.random import default_rng
rng = default_rng()

def minibatch_sample_training(dataset, w, lr, bs, epochs, random
    ↪ = "no", case = "square"):
   W = initial_weights(w,random = random)
   trajectory = [ ]
   N = len(dataset)
   for n,epoch in enumerate(range(epochs),start = 1):
      losses = [ ]
      rng.shuffle(dataset)
      minibatches = arange(0,N,bs)
      for start in minibatches:
         end = start + bs
         minibatch = dataset[start:end]
```

```
        loss, gradJ = batch_weight_gradient(minibatch, W, case =
    ↪ case)
        W = update_weights(W,gradJ,lr/n)
        # append scalar loss
        losses.append(loss[0,0])
    loss = mean(losses)
    if isclose(0,loss): break
    trajectory.append(loss)
  return W, trajectory
```

Note the code records the loss function once each epoch.



In batch sample training, the weights are updated once each pass over the dataset, so 1 iteration equals 1 epoch. In stochastic sample training, the weights are updated with each sample, so 1 epoch equals N iterations, where N is the number of samples in the dataset. In minibatch sample training, the weights are updated with each batch, so 1 epoch equals N/s iterations.

When $s = 1$, minibatch sample training is stochastic sample training, and when $s = N$, minibatch sample training is batch sample training. As of the current writing of this text, minibatch sample training is the standard training algorithm for neural networks.

When generalized to the general gradient descent setting of §7.3, stochastic sample training is stochastic gradient descent.

For stochastic and minibatch gradient descent, as explained in §7.3, the learning rates t_n must vary according to something close to $t_n = t/n$. This is reflected in the stochastic sample training and minibatch sample training codes above.

Exercises

Exercise 7.4.1 Train the network in Figure 7.17, starting with the network adjacency matrix. Do this with mean square loss and for a single sample, then for three samples. Find the learning rates leading to the fewest number of iterations.

Exercise 7.4.2 With the nodes in Figure 7.4 re-ordered as in Exercise **7.2.3**, verify the training code returns the same results.

Exercise 7.4.3 Train the network in Figure 7.11 using mean square loss and the three samples and three targets above. Start with a random W.

Exercise 7.4.4 Train the network in Figure 7.12 against the dataset

```
dataset = array([
[ 0.99335999,  1.        ,  0.        ],
[-0.8943543 ,  0.        ,  1.        ],
[ 0.87709524,  1.        ,  0.        ],
[-0.61427175,  0.        ,  1.        ],
[ 0.53202877,  1.        ,  0.        ],
[ 1.10156379,  1.        ,  0.        ],
[ 0.51760267,  1.        ,  0.        ],
[-1.30845517,  0.        ,  1.        ],
[ 0.47808674,  1.        ,  0.        ],
[-1.13024748,  0.        ,  1.        ]])
```

This dataset consists of normally distributed scalar sources with mean $+1$ or -1 according to whether the one-hot encoded targets are $(1, 0)$ or $(0, 1)$. The code for `dataset` is

```
from numpy.random import default_rng
rng = default_rng()

mu,  N = 0, 50
#  sdev = 0.5, 1.0, 2.0, 10.0
sdev = 0.5
n, p = 1, 0.5

targets = rng.binomial(n,p,N)
# source mean equals pm1 according to target=1 or 0
sources = rng.normal(mu,sdev,N) + (2*targets - 1)
# one-hot encoded targets
targets = array([targets,1-targets]).T

dataset = column_stack([sources, targets])
```

Here J is mean logistic loss (7.2.8). Start with the network adjacency matrix W and do batch sample training with 100 epochs and learning rate 0.2. Using the resulting optimal weight W^*, compare predicted targets with actual targets, for `sdev = 0.5, 1.0, 2.0, 10.0`. What is your percentage success rate as `sdev` varies?

7.5 Linear Regression

Let $x_1, x_2, \ldots, x_N$ be a dataset, with corresponding labels or targets $y_1, y_2, \ldots, y_N$ in $\mathbf{R}^d$. As in §7.1, the loss function is

$$J(W) = \frac{1}{N} \sum_{k=1}^{N} J(x_k, y_k, W). \tag{7.5.1}$$

In this and the next section, we focus on a single-layer perceptron (Figure 7.20),

$$J(x, y, W) = J(z, y), \qquad z = Mx + b. \tag{7.5.2}$$

Here the weight matrix is $W = (M, b)$, b, z, and y are in $\mathbf{R}^d$, and M has d rows.

When the outputs are scalar, $d = 1$, M is a single row, b, z, and y are scalars, and $w = W^t$ is a column vector. Then

$$z = Mx + b = (M, b)\begin{pmatrix} x \\ 1 \end{pmatrix} = W\tilde{x} = w \cdot \tilde{x},$$

which is consistent with augmented vectors (7.2.3) in §7.2.

Because $W = (M, b)$, the loss function may be written $J(W)$ or $J(M, b)$.



A basic attribute of a neural network is its trainability. Can a given network be trained to achieve desired input-output behavior? As stated, this question is imprecise and not clearly defined. In fact, for deep networks, it is not at all clear how to turn this vague idea into an actionable definition.

In the case of a single-layer perceptron, the situation is straightforward enough to be able to both make the question precise, and to provide actionable criteria that guarantee trainability. This we do in the two cases

- linear regression, and
- logistic regression.

In both cases, as we shall see, the loss functions have continuous gradients $\nabla_W J(W)$.

With any loss function $J(W)$, the goal is to minimize $J(W)$. With this in mind, from §4.3 and §4.5, we recall

Ideal Loss Function

Assume $J(W)$ is strictly convex. Then $J(W)$ is proper iff $J(W)$ has a unique minimizing weight W^*,

$$J(W^*) \leq J(W), \qquad \text{for every } W. \tag{7.5.3}$$

Often, in machine learning, $J(W)$ is neither convex nor proper. Nevertheless, this result is an important benchmark to start with. Lack of properness is often addressed by *regularization*, which is the modification of J by a proper forcing term. Lack of convexity is addressed by using some type of accelerated gradient descent.

We say the regression is *trainable* if the loss function (7.5.1) has a minimizing weight W^* (7.5.3).

In this section, we determine conditions on the dataset that guarantee trainability for linear regression, and, in the next section, we determine conditions on the dataset that guarantee trainability for logistic regression. The results in this section rely on the analysis of convex sets in §4.5.

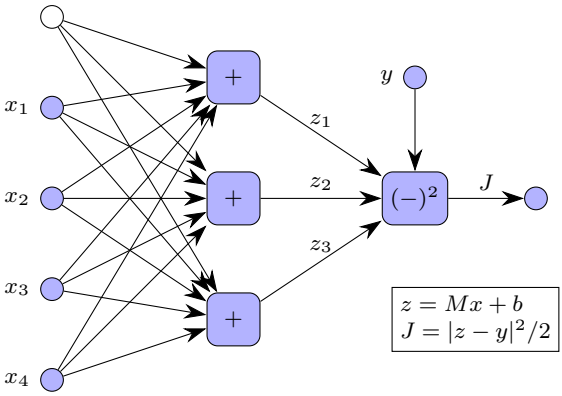


Fig. 7.20 Linear regression neural network.

For linear regression, the loss function is (7.5.1), (7.5.2) with

$$J(z, y) = \frac{1}{2}|z - y|^2. \tag{7.5.4}$$

Then (7.5.1) is the *mean square error* or *mean square loss*, and the problem of minimizing (7.5.1) is *linear regression* (Figure 7.20).

A weight $W = (M, b)$ with $M \neq 0$ is a *hyperplane weight*. Below we need the following.

Hyperplane Weight

A dataset $x_1, x_2, \ldots, x_N$ lies in a hyperplane iff

$$Mx_k + b = 0, \qquad k = 1, 2, \ldots, N, \tag{7.5.5}$$

for some hyperplane weight (M, b),

This is Exercise **7.5.1**.



We use the identities (2.2.10) and (2.2.15) to compute the gradient of $J(x, y, W)$.

Let $V = (M_1, b_1)$ be a weight and let $v = M_1 x + b_1$, $z = Mx + b$. Then

$$z + tv = (M + tM_1)x + (b + tb_1) = (W + tV)\tilde{x},$$

and the directional derivative is

$$\begin{aligned} \frac{d}{dt} J(x, y, W + tV) &= \frac{d}{dt} \frac{1}{2} |z + tv - y|^2 = v \cdot (z + tv - y) \\ &= \operatorname{trace}\left(V(\tilde{x} \otimes (z + tv - y))\right). \end{aligned} \tag{7.5.6}$$

By (4.3.6), inserting $t = 0$, and recalling $(u \otimes v)^t = v \otimes u$, (7.5.6) implies the weight gradient for mean square loss is

$$\nabla_W J(x, y, W) = (z - y) \otimes \tilde{x} = (z - y) \otimes \begin{pmatrix} x \\ 1 \end{pmatrix} = \begin{pmatrix} (z - y) \otimes x \\ z - y \end{pmatrix}. \tag{7.5.7}$$

Note this result is a special case of (7.4.2).

Differentiating (7.5.6) with respect to t and inserting $t = 0$,

$$\left.\frac{d^2}{dt^2}\right|_{t=0} J(x, y, W + tV) = |v|^2 = |M_1 x + b_1|^2. \tag{7.5.8}$$

Since this is nonnegative, by (4.5.13), $J(x, y, W)$ is a convex function of W.

Since $J(W)$ is the average of $J(x, y, W)$ over all samples, (7.5.7) shows $\nabla_W J(W)$ is a continuous function of (M, b). Since $J(W)$ is the average of $J(x, y, W)$ over all samples, (7.5.8) shows $J(W)$ is a convex function of (M, b).

To check strict convexity of $J(W)$, suppose

$$\left.\frac{d^2}{dt^2}\right|_{t=0} J(W + tV) = 0.$$

Then (7.5.8) vanishes for all samples $x = x_k$, $y = y_k$, which implies

$$M_1 x_k + b_1 = 0, \qquad k = 1, 2, \ldots, N. \tag{7.5.9}$$

If the dataset does not lie in a hyperplane, by Exercise **7.5.1**, $M_1 = 0$ and $b_1 = 0$. Hence J is strictly convex.

Linear Regression: Strict Convexity

Suppose the dataset $x_1, x_2, \ldots, x_N$ does not lie in a hyperplane. Then mean square loss is strictly convex.

Let K be the convex hull of the dataset. When the dataset does not lie in a hyperplane, the mean μ lies in the interior of K (Exercise **4.5.15**). Thus (§4.5) there is a ball B centered at μ wholly contained in K. Let $r > 0$ be the radius of B. Then the point $x = \mu + rv$ lies in B for every unit vector v.

We now show $J(W)$ is proper when the dataset does not lie in a hyperplane. To check properness of $J(W)$, by definition (4.3.7), given a level c, we show there is a bound C with

$$J(W) \le c \qquad \Longrightarrow \qquad \|W\| \le C.$$

Here $\|W\|$ is the norm of the matrix $W = (M, b)$ (2.2.16).

The exact form of the bound C does not matter. What matters, for properness, is the existence of *some* bound C depending on the level c and the dataset.

Let $m_1, m_2, \ldots, m_d$ be the rows of M, let $b = (b_1, b_2, \ldots, b_d)$, and let

$$C = \frac{1}{r}\left(\sqrt{2Nc} + \max_k |y_k|\right)(|\mu| + r + 1).$$

We establish properness in the form

$$J(M, b) \le c \qquad \Longrightarrow \qquad |m_i| + |b_i| \le C, \qquad i = 1, 2, \ldots, d. \tag{7.5.10}$$

Since

$$\sum_{i=1}^{d} (|m_i|^2 + b_i^2) = \|W\|^2, \tag{7.5.11}$$

this is equivalent to properness as stated above.

Assume $J(M, b) \le c$. By (7.5.1), (7.5.2), and the triangle inequality,

$$|Mx_k + b| \le \sqrt{2Nc} + |y_k|, \qquad k = 1, 2, \ldots, N.$$

By taking convex combinations of samples,

$$|Mx + b| \le \sqrt{2Nc} + \max_k |y_k|, \qquad \text{for } x \text{ in } K.$$

In particular, since μ is in K,

$$|m_i \cdot \mu + b_i| \le \sqrt{2Nc} + \max_k |y_k|, \qquad i = 1, 2, \ldots, d.$$

With B as above, insert $x = \mu \pm rv$ with v a unit vector,

$$|M(\mu \pm rv) + b| \le \sqrt{2Nc} + \max_k |y_k|.$$

Since $2rv = (\mu + rv) - (\mu - rv)$, by the triangle inequality,

$$r|Mv| \le \sqrt{2Nc} + \max_k |y_k|.$$

From this, since v is any unit vector, we have

$$|m_i| \le \frac{1}{r}\left(\sqrt{2Nc} + \max_k |y_k|\right), \qquad i = 1, 2, \ldots, d,$$

which implies

$$|m_i \cdot \mu| \le \frac{|\mu|}{r}\left(\sqrt{2Nc} + \max_k |y_k|\right), \qquad i = 1, 2, \ldots, d.$$

By the triangle inequality, $|b_i| \le |m_i \cdot \mu + b_i| + |m_i \cdot \mu|$, so

$$|b_i| \le \frac{1}{r}\left(\sqrt{2Nc} + \max_k |y_k|\right)(|\mu| + r).$$

Since this establishes (7.5.10), the loss function is proper. Since this implies trainability,

Linear Regression: Trainability

Suppose the dataset $x_1, x_2, \ldots, x_N$ does not lie in a hyperplane. Then linear regression is trainable.

This is a simple geometric criterion for convergence of gradient descent to the global minimum of J, valid for linear regression.



From §7.3, a gradient descent sequence is *short-step* if the learning rates are no greater than $1/L$, where L is an upper bound for the second derivative of the loss function. As such, it is important to obtain a reasonable formula for L.

Let μ and Q be the mean and variance of the dataset, and let

$$L = 1 + |\mu|^2 + \operatorname{trace}(Q).$$

With $V = (M, b)$, by the Cauchy-Schwarz inequality,

$$|Mx + b|^2 = |V\tilde{x}|^2 \le \|V\|^2 \left(|x|^2 + 1\right).$$

By (7.5.8), we have

$$\left.\frac{d^2}{dt^2}\right|_{t=0} J(W + tV) \le \|V\|^2 \left(\frac{1}{N}\sum_{k=1}^{N}(|x_k|^2 + 1)\right) = L\|V\|^2.$$

We have shown

Linear Regression Gradient Descent

A gradient descent sequence with learning rates equal to $1/L$ is short-step. Moreover, when the dataset does not lie in a hyperplane, a gradient descent sequence with learning rates $1/L$ converges to the unique minimizer of the loss function.

The second statement follows because the sequence converges to a critical point, and a strictly convex function has a unique critical point equal to its minimizer.

Exercises

Exercise 7.5.1 Show a dataset $x_1, x_2, \ldots, x_N$ lies in a hyperplane iff there is a hyperplane weight (M, b) satisfying (7.5.5).

7.6 Logistic Regression

A dataset is a *two-class dataset* if it is composed of two disjoint classes. More generally, a dataset is a *multi-class dataset* if it is composed of $d \geq 2$ disjoint classes. Separability of two-class datasets was first discussed in §4.5.

Recall (§5.3) a vector $p = (p_1, p_2, \ldots, p_d)$ is a probability vector if each component $p_1, p_2, \ldots, p_d$ is nonnegative (positive or zero), and the components sum to one, $p_1 + p_2 + \cdots + p_d = 1$.

A probability vector p is *strict* if the components are all positive (none are zero). A probability vector p is *one-hot encoded* if one of the components is one. When this is the i-th component, $p_i = 1$, we say p is *one-hot encoded at slot i*. When this happens, all other components are zero.

Let $x_1, x_2, \ldots, x_N$ be a dataset with corresponding labels or targets $p_1, p_2, \ldots, p_N$. If each target p is a probability vector, the dataset is a *soft-class dataset*.

A two-class dataset is a soft-class dataset: Assign to each sample x the one-hot encoded probability vector

$$p = \begin{cases} (1,0), & \text{if } x \text{ is class 1,} \\ (0,1), & \text{if } x \text{ is class 2.} \end{cases}$$

More generally, a multi-class dataset is a soft-class dataset: Assign to each sample x the one-hot encoded probability vector $p = (p_1, p_2, \ldots, p_d)$ satisfying

$$p_i = 1 \qquad \Longleftrightarrow \qquad x \text{ is in class } i, \qquad i = 1, 2, \ldots, d.$$

For example, if there are three classes, as in the Iris dataset, the probability vector p_k corresponding to sample x_k is one of

$$(1, 0, 0), \qquad (0, 1, 0), \qquad (0, 0, 1),$$

according to the class of the sample x_k.

On the other hand, in a soft-class dataset, we can assign classes: For each $i = 1, 2, \ldots, d$, let class i consist of all samples x whose corresponding targets $p = (p_1, p_2, \ldots, p_d)$ satisfy $p_i > 0$. Because the resulting classes may overlap and not be disjoint, they are *soft classes*. There are other ways of assigning classes in a soft-class dataset. For example, we may set class i to be the samples whose targets satisfy $p_i = \max p$.

In what follows, given a soft-class dataset, *we assume the i-th class is some class of samples satisfying $p_i > 0$, for each $i = 1, 2, \ldots, d$.* The i-th class need not be all samples satisfying $p_i > 0$. In particular, this condition is satisfied by multi-class datasets with one-hot encoded targets as above.

Summarizing, a *soft-class dataset* is a dataset $x_1, x_2, \ldots, x_N$ with probability vector targets $p_1, p_2, \ldots, p_N$. A soft-class dataset is a multi-class dataset iff the targets are one-hot encoded.



Let $x_1, x_2, \ldots, x_N$ be a dataset, with corresponding labels or targets $p_1, p_2, \ldots, p_N$ consisting of probability vectors. The loss function is

$$J(W) = \frac{1}{N} \sum_{k=1}^{N} J(x_k, p_k, W). \tag{7.6.1}$$

As in the previous section, we focus on a single-layer perceptron

$$J(x, p, W) = J(p, y), \qquad y = Mx + b. \tag{7.6.2}$$

Here the weight matrix is $W = (M, b)$, b, y, and p are in $\mathbf{R}^d$, and M has d rows.

For logistic regression, the loss function is (7.6.1), (7.6.2) with

$$J(p, y) = I(p, q), \qquad q = \sigma(y), \tag{7.6.3}$$

where $q = \sigma(y)$ is the softmax function, squashing the network's output y into the probability vector q, and $I(p, q)$ is the relative information (§5.6). In §5.6, $I(p, \sigma(y))$ is the *information error* (5.6.16).

Let $\mathbf{1} = (1, 1, \ldots, 1)$. If (m, b) is a hyperplane, let

$$V = \mathbf{1} \otimes (m, b), \qquad \tilde{x} = (x, 1), \qquad y = m \cdot x + b.$$

Then $V\tilde{x} = y\mathbf{1}$, and, by (5.6.4), the loss function satisfies

$$J(W+V) = J(W) \qquad \text{for all } W.$$

To avoid this redundancy, we restrict the loss function to centered weights.

To explain this, from §5.6, a vector b is centered if $b\cdot\mathbf{1} = 0$. We say a matrix M is *centered* if the columns of M are centered, and a weight $W = (M, b)$ is *centered* if M is centered and b is centered. Then M is centered iff $M^t\mathbf{1} = 0$, and $y = Mx + b$ is centered when W is centered.

Since $V^t\mathbf{1} = d\cdot(m, b)$, V is centered only if $V = 0$. If we restrict the loss function to centered weights, it follows the redundancy is avoided.

Because of this, in logistic regression we work only with centered weights W, and we say logistic regression is *trainable* if there is a centered weight W^* minimizing $J(W)$ over all centered weights W.

When $d = 1$, W is centered iff $W = 0$, hence there is no scalar case $d = 1$ for logistic regression with centered weights. For the $d = 2$ case, when W is centered, we can write (compare Figures 7.27 and 7.28)

$$W = \begin{pmatrix} m_1 & b_1 \\ m_2 & b_2 \end{pmatrix} = \begin{pmatrix} m & b \\ -m & -b \end{pmatrix}, \qquad \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} y \\ -y \end{pmatrix}, \qquad y = m\cdot x + b. \quad (7.6.4)$$

In this case, the loss $J(w) = J(m, b)$ is a function of a single hyperplane (m, b).

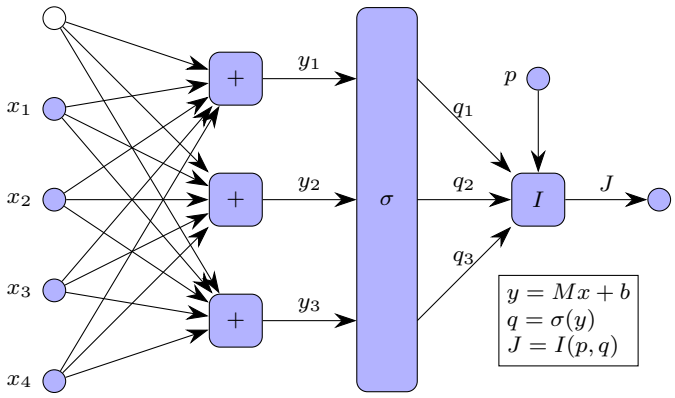


Fig. 7.21 Logistic regression neural network.

Then (7.6.1) is *mean logistic error* or *mean logistic loss*, and the problem of minimizing (7.6.1) is *logistic regression* (Figure 7.21).

Because $I(p, q)$ is only defined for probability vectors p, q, logistic regression applies only to soft-class datasets.

We compute the gradient $\nabla_W J(x, p, W)$. By (5.6.3) and (5.6.16),

$$\nabla_y I(p, \sigma(y)) = \nabla_y Z(y) - p = q - p, \qquad q = \sigma(y), \tag{7.6.5}$$

and, by (5.6.11),

$$D_y^2 I(p, \sigma(y)) = D^2 Z(y) = \operatorname{diag}(q) - q \otimes q, \qquad q = \sigma(y). \tag{7.6.6}$$

Let $V = (M_1, b_1)$ be a weight matrix and bias vector, and let $v = V\tilde{x} = M_1 x + b_1$, $y = W\tilde{x} = Mx + b$. Then

$$y + tv = (W + tV)\tilde{x} = (M + tM_1)x + (b + tb_1),$$

and, by (7.6.5), the directional derivative is

$$\begin{aligned} \left.\frac{d}{dt}\right|_{t=0} J(x, y, W + tV) &= \left.\frac{d}{dt}\right|_{t=0} I(p, \sigma(y + tv)) \\ &= v \cdot (q - p) \\ &= \operatorname{trace}\left(V\tilde{x} \otimes (q - p)\right). \end{aligned}$$

By (4.3.6), this shows the gradient for log loss is

$$\nabla_W J(x, p, W) = (q - p) \otimes \tilde{x} = (q - p) \otimes \begin{pmatrix} x \\ 1 \end{pmatrix} = \begin{pmatrix} (q - p) \otimes x \\ q - p \end{pmatrix}. \tag{7.6.7}$$

This result is a special case of (7.4.2). Since q and p are probability vectors, $p \cdot \mathbf{1} = 1 = q \cdot \mathbf{1}$, the gradient is centered.

Since $J(W)$ is the average of $J(x, y, W)$ over all samples, (7.6.7) shows $\nabla_W J(W)$ is a continuous function of (M, b). If we initiate gradient descent with a centered weight $W = (M, b)$, since the gradient is also centered, all successive weight matrices will be centered.



Turning to convexity, as for linear regression, we establish strict convexity when the dataset does not lie in a hyperplane.

Logistic Regression: Strict Convexity

Suppose the soft-class dataset $x_1, x_2, \ldots, x_N$ does not lie in a hyperplane. Then mean logistic loss is strictly convex on centered weights.

To see this, given a vector v and probability vector q, set $\bar{v} = \sum_{j=1}^{d} v_j q_j$. Then

$$\sum_{j=1}^{d} v_j^2 q_j - \left(\sum_{j=1}^{d} v_j q_j\right)^2 = \sum_{j=1}^{d} (v_j - \bar{v})^2 q_j.$$

If either side is zero, and q is strict, then $v = \bar{v}\mathbf{1}$, so v is a multiple of $\mathbf{1}$.

From this identity, and by (4.5.13) and (7.6.6), the second derivative of $I(p, \sigma(y))$ in the direction of a vector v is

$$\left.\frac{d^2}{dt^2}\right|_{t=0} I(p, \sigma(y + tv)) = \sum_{j=1}^{d} (v_j - \bar{v})^2 q_j, \qquad q = \sigma(y).$$

Let $V = (M_1, b_1)$ be a centered weight and let $v = M_1 x + b_1 = V\tilde{x}$. Then the second derivative of $J(x, p, W)$ in the direction of V is

$$\left.\frac{d^2}{dt^2}\right|_{t=0} J(x, p, W + tV) = \sum_{j=1}^{d} (v_j - \bar{v})^2 q_j, \qquad v = M_1 x + b_1. \tag{7.6.8}$$

This shows the second derivative of $J(x, p, W)$ is nonnegative, establishing the convexity of $J(x, p, W)$. Since $J(W)$ is the average of $J(x, p, W)$ over all samples, we conclude $J(W)$ is convex.

Moreover, if (7.6.8) vanishes, then, by the previous paragraph, since $q = \sigma(y)$ is strict, v is a multiple of $\mathbf{1}$. Since v is centered, it follows $v = 0$ (Exercise **5.6.1**). Since $v = M_1 x + b_1$, this implies $M_1 x + b_1 = 0$.

If

$$\left.\frac{d^2}{dt^2}\right|_{t=0} J(W + tV) = \frac{1}{N} \sum_{k=1}^{N} \left.\frac{d^2}{dt^2}\right|_{t=0} J(x_k, p_k, W + tV)$$

vanishes, then, since the summands are nonnegative, (7.6.8) vanishes, for every sample $x = x_k$, $p = p_k$, hence

$$M_1 x_k + b_1 = 0, \qquad k = 1, 2, \ldots, N.$$

As with linear regression, if the dataset does not lie in a hyperplane, by Exercise **7.5.1**, $M_1 = 0$ and $b_1 = 0$. Hence J is strictly convex. This establishes strict convexity of $J(W)$ on centered weights.



Now we turn to properness of $J(W) = J(M, b)$. The reader may wish to review the properness of the cumulant-generating function (5.6.12) before continuing.

If the rows of M are m_1, m_2, …, m_d, and $b = (b_1, b_2, \ldots, b_d)$, then $y = Mx + b$ is equivalent to levels corresponding to d hyperplanes (§4.5)

$$\begin{aligned} y_1 &= m_1 \cdot x + b_1, \\ y_2 &= m_2 \cdot x + b_2, \\ \dots &= \dots \\ y_d &= m_d \cdot x + b_d. \end{aligned} \tag{7.6.9}$$

The scalars $y_1, y_2, \dots, y_d$ are the outputs corresponding to the sample x and weight $W = (M, b)$.

Let $x_1, x_2, \dots, x_N$ be a soft-class dataset with associated target probability vectors $p_1, p_2, \dots, p_N$. We assume there are d possibly overlapping classes, and, for each sample x in class i, the corresponding target p satisfies $p_i > 0$.

In §4.5, we defined separating hyperplanes for two-class datasets. There are at least two generalizations of separability to soft-class datasets. They are strong separability ("all-against-all"), and weak separability ("some-against-some"). Let $y_1, y_2, \dots, y_d$ be the outputs (7.6.9).

A soft-class dataset is *strongly separable* if there is a hyperplane (m_i, b_i) separating class i from the rest of the dataset, for every $i = 1, 2, \dots, d$. By Exercise **7.6.2**, this is the same as saying there is a weight $W = (M, b)$ such that

$$\begin{aligned} y_i &\geq 0, \quad \text{for } x \text{ in class } i, \\ y_i &\leq 0, \quad \text{for } x \text{ in class } j, \end{aligned} \qquad \text{for every } i = 1, 2, \dots, d \text{ and every } j \neq i. \tag{7.6.10}$$

Here again the hyperplanes are *decision boundaries*.

On the other hand, a soft-class dataset is *weakly separable* if there is a hyperplane (m_i, b_i) separating some class i and some class $j \neq i$. By Exercise **7.6.1**, this is the same as saying there is a weight $W = (M, b)$ such that

$$\begin{aligned} y_i &\geq 0, \quad \text{for } x \text{ in class } i, \\ y_i &\leq 0, \quad \text{for } x \text{ in class } j, \end{aligned} \qquad \text{for some } i = 1, 2, \dots, d \text{ and some } j \neq i. \tag{7.6.11}$$

Clearly strong separability implies weak separability. If a dataset lies in a hyperplane (4.5.7), the dataset is separable, in both strong and weak senses. Thus the question of separability is only interesting when the dataset does not lie in a hyperplane.

We say a dataset is *strongly inseparable* if the dataset is not weakly separable, and *weakly inseparable* if the dataset is not strongly separable. Clearly strong inseparability implies weak inseparability.

When the dataset is two-class, $d = 2$, and $W = (M, b)$ is centered, strong separability equals weak separability, and, by (7.6.4), both are equivalent to separability as defined in (4.5.6).



The first trainability result is for two-class datasets. Here separability is as in §4.5 with a weight $w = (m, b)$, or as above with a centered weight $W = (M, b)$ with two rows.

Fundamental Theorem of Linear Binary Classifiers

Assume neither class of a two-class dataset lies in a hyperplane. Then logistic regression is trainable iff the dataset is inseparable.

For this two-class dataset, by (7.6.4), trainability is equivalent to the existence of a minimizing weight $w = (m, b)$. When the means of the classes are distinct, $m \neq 0$, and w is a hyperplane (Exercise **7.6.5**). In this case, we say the weight w is the *LR hyperplane*.

The two-class case, derived on its own in [16], follows from

Multi-Class Logistic Regression: Trainability

Suppose none of the classes of a multi-class dataset lie in a hyperplane.

- If the dataset is strongly separable, logistic regression is not trainable.
- If the dataset is strongly inseparable, logistic regression is trainable.

For soft-class datasets, strong separability does not imply non-trainability. However, if the soft-class dataset is strongly separable and the inequalities in (7.6.10) are strict, then the classes are disjoint, and the dataset is multi-class.

In any case, for any soft-class dataset, strong inseparability still implies trainability,

Soft-Class Logistic Regression: Trainability

Suppose none of the classes of a soft-class dataset lie in a hyperplane. If the dataset is strongly inseparable, logistic regression is trainable.

Again, as above, $M^* = 0$ only happens iff the means of the classes agree (Exercise **7.6.5**). These are the main results of the section.



To establish these results, suppose the dataset is multi-class and $W = (M, b)$ satisfies (7.6.10). Then (Exercise **7.6.3**)

$$\begin{matrix} y_i \geq 0, \\ y_j \leq 0, \end{matrix} \quad \text{for every } j \neq i, \text{ for every } x \text{ in class } i, \tag{7.6.12}$$

for every $i = 1, 2, \ldots, d$.

Since the dataset is multi-class, the targets are one-hot encoded. From (7.6.12), one obtains $I(p, \sigma(y)) \le \log d$ for every sample x (Exercise **7.6.4**), which implies $J(W) \le \log d$.

When $W = (M, b)$ satisfies (7.6.10), $tW = (tM, tb)$ also satisfies (7.6.10) for all $t > 0$, hence $J(tW) \le \log d$ for all $t > 0$. This shows the loss function is not proper, hence not trainable.

This establishes one direction: multi-class strong separability implies non-trainability. We now establish the other direction, for any soft-class dataset.

Recall (§4.5) a set K has interior if there is a ball B in K. For each $i = 1, 2, \ldots, d$, let K_i be the convex hull of the samples in class i. By the full-rank dataset theorem in §4.5, K_i has interior iff class i does not lie in a hyperplane.

Assume none of the classes lie in a hyperplane and the dataset is strongly inseparable. By the hyperplane separation theorem (4.5.8), $K_i \cap K_j$ has interior for all i and all $j \ne i$.

Let x^*_{ij} be the centers of balls in $K_i \cap K_j$ for each $i \ne j$. By making the balls small enough, we may assume the radii of the balls equal the same $r > 0$.

Let $W = (M, b)$, with $b = (b_1, b_2, \ldots, b_d)$ and $m_1, m_2, \ldots, m_d$ be the rows of M, both centered, and set

$$C_i = \frac{cN + \log d}{r\epsilon} \left(1 + r + \frac{1}{d} \sum_{j \ne i} |x^*_{ij}| \right).$$

As in the previous section, we establish properness of the loss function by showing

$$J(M, b) \le c \qquad \Longrightarrow \qquad |m_i| + |b_i| \le C_i, \qquad i = 1, 2, \ldots, d. \tag{7.6.13}$$

Let $Z(y)$ be the cumulant-generating function (5.6.2). Then $Z(y) > y_j$, $j = 1, 2, \ldots, d$. Suppose p is a probability vector satisfying $p_i \ge \epsilon$. Then

$$\epsilon(y_j - y_i) < \epsilon(Z(y) - y_i) \le p_i(Z(y) - y_i) \le \sum_{k=1}^{d} p_k(Z(y) - y_k) = Z(y) - p \cdot y,$$

hence

$$\epsilon(y_j - y_i) < Z(y) - p \cdot y, \qquad j \ne i.$$

Let ϵ_i be the minimum of p_i over all targets p corresponding to samples x in soft-class i. By the definition of the soft-classes, $\epsilon_i > 0$. Let ϵ be the least of $\epsilon_1, \epsilon_2, \ldots, \epsilon_d$. Then $p_i \ge \epsilon > 0$ for x in class i, for every $i = 1, 2, \ldots, d$.

Now suppose x is in class i and let $y = Mx + b$. If $J(M, b) \le c$, then $I(p, q) \le cN$ for each sample x. By (5.6.16),

$$Z(y) - p \cdot y = I(p, \sigma(y)) - I(p) \le cN + \log d.$$

Combining the last two inequalities,

$$\epsilon(y_j - y_i) < cN + \log d, \qquad j \neq i, \text{for samples } x \text{ in class } i.$$

By taking convex combinations of samples,

$$\epsilon(y_j - y_i) \leq cN + \log d, \qquad j \neq i, \text{for all } x \text{ in } K_i.$$

Repeating the same argument for x in K_j,

$$\epsilon(y_i - y_j) \leq cN + \log d, \qquad j \neq i, \text{for all } x \text{ in } K_j.$$

Combining the last two inequalities,

$$\epsilon|y_i - y_j| \leq cN + \log d, \qquad j \neq i, \text{for all } x \text{ in } K_i \cap K_j. \tag{7.6.14}$$

Let v be a unit vector, and let

$$x^{\pm} = x^*_{ij} \pm rv, \qquad y_i^{\pm} = m_i \cdot x^{\pm} + b_i, \qquad y_j^{\pm} = m_j \cdot x^{\pm} + b_j.$$

Since $y_i - y_j = (m_i - m_j) \cdot x + (b_i - b_j)$, and $x^{\pm}$ are in $K_i \cap K_j$, by (7.6.14),

$$2r\epsilon|(m_i - m_j) \cdot v| = \epsilon|(y_i^+ - y_j^+) - (y_i^- - y_j^-)| \leq 2(cN + \log d).$$

Optimizing over all v, or choosing $v = (m_i - m_j)/|m_i - m_j|$, we obtain

$$r\epsilon|m_i - m_j| \leq cN + \log d.$$

Let

$$y_i = m_i \cdot x^*_{ij} + b_i, \qquad y_j = m_j \cdot x^*_{ij} + b_j.$$

Since x^*_{ij} is in $K_i \cap K_j$, by (7.6.14),

$$\begin{aligned} r\epsilon|b_i - b_j| &\leq r\epsilon|y_i - y_j| + r\epsilon|(m_i - m_j) \cdot x^*_{ij}| \\ &\leq r(cN + \log d) + r\epsilon|w_i - w_j|\,|x^*_{ij}| \leq (cN + \log d)\left(r + |x^*_{ij}|\right). \end{aligned}$$

Hence

$$r\epsilon|m_i - m_j| + r\epsilon|b_i - b_j| \leq (cN + \log d) \cdot (1 + r + |x^*_{ij}|). \tag{7.6.15}$$

Since M is centered,

$$dm_i = (d-1)m_i + m_i = (d-1)m_i - \sum_{j \neq i} m_j = \sum_{j \neq i}(m_i - m_j).$$

Similarly, since b is centered,

$$db_i = \sum_{j \neq i}(b_i - b_j).$$

Hence

$$|m_i| + |b_i| \le \frac{1}{d} \sum_{j \neq i} |m_i - m_j| + |b_i - b_j|.$$

Combining this with (7.6.15) results in (7.6.13), this completes the proof of the main results.

Let μ and Q be the mean and variance of the dataset, and let $L = 1 + |\mu|^2 + \operatorname{trace}(Q)$. By Exercise 5.6.7 and (7.6.8),

$$\left.\frac{d^2}{dt^2}\right|_{t=0} J(x, p, W + tV) \le \left(1 + |x|^2\right) \|V\|^2.$$

Averaging over all samples $(x, p) = (x_k, p_k)$, we obtain $D_W^2 J(W) \le L$. We conclude a result similar to that presented for linear regression.

Two-class Logistic Regression Gradient Descent

A gradient descent sequence with learning rates equal to $1/L$ is short-step. Moreover, when neither class in a two-class dataset lies in a hyperplane, a gradient descent sequence with learning rates $1/L$ converges to the unique minimizer of the loss function, when the dataset is inseparable, or diverges to infinity, when the dataset is separable.

To establish the second statement, either the sequence diverges to infinity or not. If not, then the sequence subconverges (§A.8) to some weight W^*. From §7.3, W^* is a critical point of the loss function, hence a minimizer, hence the dataset is inseparable.



One way to find a separating hyperplane is to build a loss function that directly penalizes misclassification, and then to minimize that loss function.

The simplest such loss function is the average of the levels of samples, adjusted by $\pm$ according to class, and only counting mismatched samples. This leads us to the loss function

$$J_{\text{pa}}(m, b) = \frac{1}{N} \sum_{k=1}^{N} \begin{Bmatrix} \operatorname{relu}(y_k), & \text{if } p_k = 0, \\ \operatorname{relu}(-y_k), & \text{if } p_k = 1, \end{Bmatrix} \qquad y_k = m \cdot x_k + b. \quad (7.6.16)$$

The *perceptron algorithm* [29] iterates stochastic gradient descent along this loss function. When the dataset is separable, the perceptron convergence theorem [29] guarantees convergence in finitely many steps to a separating hyperplane.

Let $t > 0$ be a scale factor. Sample levels can be rescaled by using loss $J(tm, tb)$. Using (5.6.16), one shows [16] $J(m, b)$ is a softened version of $J_{\text{pa}}(m, b)$,

Perceptron Algorithm is Scaling Limit of Log Regression

For the two-class logistic regression loss function $J(m, b)$,

$$\lim_{t\to\infty} \frac{1}{t} J(tm, tb) = J_{\text{pa}}(m, b), \qquad \text{for every } (m, b). \tag{7.6.17}$$

Exercises

Exercise 7.6.1 Show a dataset $x_1, x_2, \ldots, x_N$ is weakly separable iff (7.6.11) holds.

Exercise 7.6.2 Show a dataset $x_1, x_2, \ldots, x_N$ is strongly separable iff (7.6.10) holds.

Exercise 7.6.3 Show a dataset $x_1, x_2, \ldots, x_N$ is strongly separable iff (7.6.12) holds.

Exercise 7.6.4 Let (M, b) be strongly separating, and let $y = Mx + b$. Assume p is one-hot encoded. Using (5.6.16) and (7.6.12), show $I(p, \sigma(y)) \leq \log d$.

Exercise 7.6.5 Suppose the two-class dataset does not lie in a hyperplane. Then the means of the two classes agree iff there is an optimal weight (m, b) with $m = 0$.

Exercise 7.6.6 Let $x_1, x_2, \ldots, x_N$ be any dataset with d features. The *double* of the dataset is the two-class dataset with both classes equal to the original dataset. Show the minimizer for the double dataset loss is $W = 0$.

Exercise 7.6.7 Given $\delta \geq 0$ and a dataset C in the interval $[-1, 1]$, let C_+ equal C shifted to the right by δ, and let C_- equal C shifted to the left by δ. Let $S(\delta)$ be the minimum value of the logistic loss function of the resulting two-class dataset (C_+, C_-). Use gradient descent to compute and plot $S(\delta)$ for $0 \leq \delta \leq 1$ and for $\delta > 1$. Note $S(0) = \log 2$ and $S(\delta) = 0$ for $\delta > 1$.

7.7 Regression Examples

Let x_k, $k = 1, 2, \ldots, N$, be a scalar dataset with corresponding targets y_k, $k = 1, 2, \ldots, N$. The simplest regression problem is to determine the line $y = mx + b$ minimizing the residual

$$J(m, b) = \frac{1}{N} \sum_{k=1}^{N} (mx_k + b - y_k)^2. \tag{7.7.1}$$

This line is the *regression line*. For the dataset in Figure 7.2, the regression line is in Figure 7.22.

GNP.deflator	GNP	Unemployed	Armed Forces	Population	Year	Employed
83	234.289	235.6	159	107.608	1947	60.323
88.5	259.426	232.5	145.6	108.632	1948	61.122
88.2	258.054	368.2	161.6	109.773	1949	60.171
89.5	284.599	335.1	165	110.929	1950	61.187
96.2	328.975	209.9	309.9	112.075	1951	63.221
98.1	346.999	193.2	359.4	113.27	1952	63.639
99	365.385	187	354.7	115.094	1953	64.989
100	363.112	357.8	335	116.219	1954	63.761
101.2	397.469	290.4	304.8	117.388	1955	66.019
104.6	419.18	282.2	285.7	118.734	1956	67.857
108.4	442.769	293.6	279.8	120.445	1957	68.169
110.8	444.546	468.1	263.7	121.95	1958	66.513
112.6	482.704	381.3	255.2	123.366	1959	68.655
114.2	502.601	393.1	251.4	125.368	1960	69.564
115.7	518.173	480.6	257.2	127.852	1961	69.331
116.9	554.894	400.7	282.7	130.081	1962	70.551

Table 7.2 Longley Economic Data [19].

More generally, given a vector dataset $x_1, x_2, \ldots, x_N$, and scalar targets $y_1, y_2, \ldots, y_N$, we want to minimize

$$J(m,b) = \frac{1}{N}\sum_{k=1}^{N}(m \cdot x_k + b - y_k)^2$$

over all weight vectors m and scalars b.

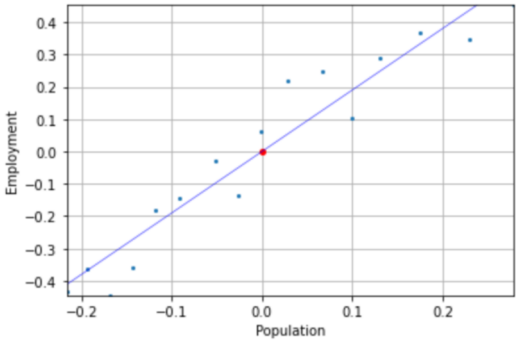


Fig. 7.22 Population versus employed: linear regression.

Figure 7.2 is a dataset and Figure 7.22 is a plot of population versus employed, with the mean and the regression line shown.

Here we are fitting a *regression hyperplane*

$$y = m \cdot x + b,$$

and this corresponds to (7.5.1), (7.5.2), (7.5.4), with $W = (m, b)$.

Since

$$0 = m \cdot x - y + b = (m, -1) \begin{pmatrix} x \\ y \end{pmatrix} + b,$$

the regression hyperplane is not a hyperplane in x space, it is a hyperplane in (x, y) space.

If the number of targets is d, then $y = (y_1, y_2, \ldots . y_d)$ and $b = (b_1, b_2, \ldots, b_d)$ are column vectors in $\mathbf{R}^d$, the weight matrix M has d rows m_1, m_2, …, m_d, and we are fitting a *regression subspace*

$$y = Mx + b.$$

This corresponds to (7.5.1), (7.5.2), (7.5.4), with $W = (M, b)$ and

$$J(W) = J(M, b) = \frac{1}{N} \sum_{k=1}^{N} |Mx_k + b - y_k|^2. \tag{7.7.2}$$

With b, and y column vectors in $\mathbf{R}^d$, and I the $d \times d$ identity matrix, we have

$$0 = Mx - y + b = (M, -I) \begin{pmatrix} x \\ y \end{pmatrix} + b.$$

Thus the regression subspace is not a subspace in x space, it is a subspace in (x, y) space.



With the dataset x_1, x_2, …, x_N, let

$$\tilde{x}_1 = \begin{pmatrix} x_1 \\ 1 \end{pmatrix}, \qquad \tilde{x}_2 = \begin{pmatrix} x_2 \\ 1 \end{pmatrix}, \qquad \ldots \qquad \tilde{x}_N = \begin{pmatrix} x_N \\ 1 \end{pmatrix}, \tag{7.7.3}$$

be the augmented dataset, and let X and Y *be the transposes of the matrices*

$$\left(\tilde{x}_1 \ \tilde{x}_2 \ \ldots \ \tilde{x}_N\right), \qquad \left(y_1 \ y_2 \ \ldots \ y_N\right).$$

With this understood, we can rewrite (7.7.2) as the residual

$$J(W) = \frac{1}{N} \|WX^t - Y^t\|^2 = \frac{1}{N} \|XW^t - Y\|^2. \tag{7.7.4}$$

Following the method in §2.6, any weight W^* minimizing (7.7.4) is a solution the regression equation

$$X^t X W^t = X^t Y. \tag{7.7.5}$$

Here the regression equation involves W^t because the matrix-matrix multiplication in the residual is in the reverse order, and $\|\cdot\|^2$ is given by (2.2.17). From §2.6, the solution of the regression equation is X^+Y. By taking transposes, we obtain

Linear Regression

The weight W^* satisfying $(W^*)^t = X^+Y$ minimizes the residual (7.7.4) and solves the regression equation (7.7.5).

Of course, when N is large, the only practical solution method of (7.7.5) involves gradient descent. In particular, when N is large, this is the only practical method for obtaining the pseudo-inverse and the SVD decomposition.



We work out the regression equation when both x and y are scalar. In this case, $w = W^t = \begin{pmatrix} m \\ b \end{pmatrix}$ and the regression equation (7.7.5) is 2×2. Let

$$\mu = \frac{1}{N}\sum_{k=1}^{N} x_k, \qquad \nu = \frac{1}{N}\sum_{k=1}^{N} y_k.$$

be the means of the dataset and targets, and let $x = (x_1, x_2, \ldots, x_N)$ and $y = (y_1, y_1, \ldots, y_N)$.

Then

$$X^tX = \begin{pmatrix} x\cdot x & N\mu \\ N\mu & N \end{pmatrix}, \qquad X^tY = \begin{pmatrix} x\cdot y \\ N\nu \end{pmatrix},$$

and the regression equation reduces to

$$\begin{aligned} (x\cdot x)m + N\mu b &= x\cdot y, \\ m\mu + b &= \nu. \end{aligned} \tag{7.7.6}$$

The second equation says the regression line passes through the mean (μ, ν).

Let u and v be the centered dataset and targets,

$$u_k = x_k - \mu, \qquad v_k = y_k - \nu, \qquad k = 1, 2, \ldots, N.$$

Then

$$u\cdot u = x\cdot x - N\mu^2, \qquad u\cdot v = x\cdot y - N\mu\nu.$$

Subtracting $N\mu$ times the second equation in (7.7.6) from the first equation in (7.7.6) leads to

Scalar Linear Regression

The regression line passes through the mean (μ, ν). If the centered dataset and targets are u and v, then the slope of the regression line is

$$m = \frac{u \cdot v}{u \cdot u}.$$



Now we use linear regression to do *polynomial regression*. Starting with the dataset (x_k, y_k) in $\mathbf{R}^2$ (Figure 7.22), we can expand or "lift" the dataset from $\mathbf{R}^2$ to $\mathbf{R}^6$ by working with the vectors $(1, x_k, x_k^2, x_k^3, x_k^4, y_k)$ instead of (x_k, y_k).

Assuming the data is given by Figure 7.2, we build the code for Figures 7.22 and 7.23. We begin by assuming the data is given as arrays,

```
from numpy import *
from pandas import read_csv

df = read_csv("longley.csv")

x = df["Population"].to_numpy()
y = df["Employed"].to_numpy()
```

After this, we compute the optimal weight w^* and construct the polynomial. The regression equation is solved using the pseudo-inverse (§2.3).

Then we standardize the data

```
x = x - mean(x)
y = y - mean(y)

varx = sum(x**2)/len(x)
vary = sum(y**2)/len(y)

x = x/sqrt(varx)
y = y/sqrt(vary)
```

```
from scipy.linalg import pinv

# polynomial function - degree d-1
def poly(t,d):
   X = column_stack([ x**i for i in range(d) ])  # Nxd
   Xplus = pinv(X)
   Y = y # Nx1
```

```
    wstar = dot(Xplus,Y)
    return sum([ t**i*wstar[i] for i in range(d) ],axis = 0)
```

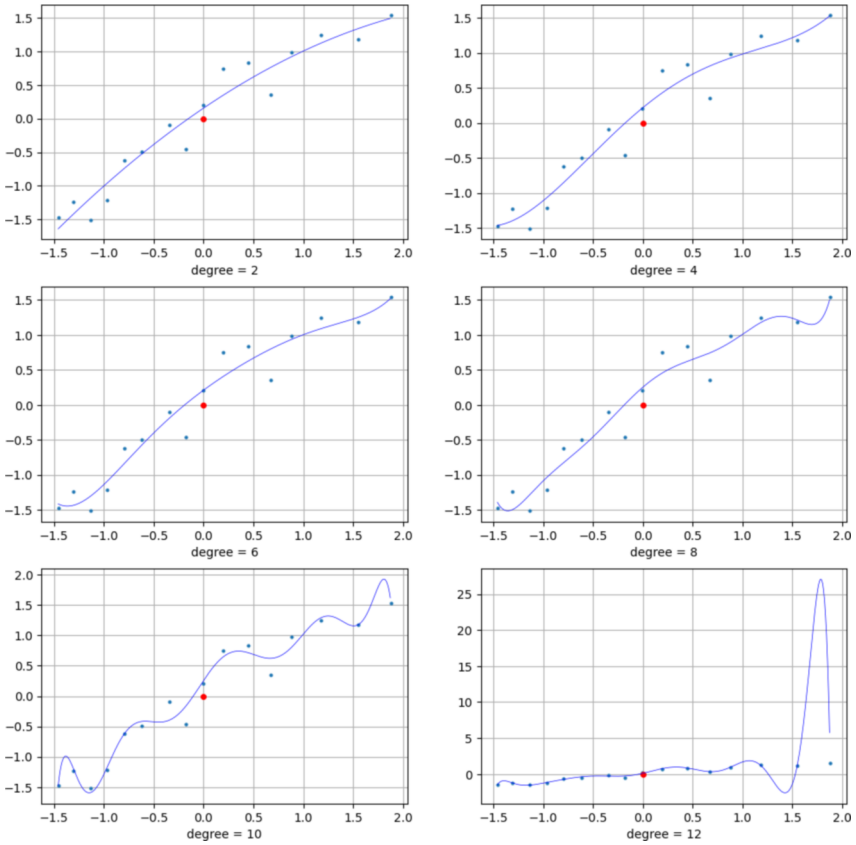


Fig. 7.23 Polynomial regression: Degrees 2, 4, 6, 8, 10, 12.

Then we plot the data and the polynomial in six subplots.

```
from matplotlib.pyplot import *

xmin,ymin = amin(x), amin(y)
xmax, ymax = amax(x), amax(y)

figure(figsize=(12,12))
# six subplots
rows, cols = 3,2
```

```
# t interval
t = arange(xmin,xmax,.01)

for i in range(6):
   d = 3 + 2*i # degree = d-1
   subplot(rows, cols,i+1)
   scatter(x,y,s=5)
   scatter([0],[0],s=7,color = "red")
   plot(t,poly(t,d),color = "blue",linewidth = .5)
   title("degree = " + str(d-1))
   grid()

show()
```

Running this code with degree 1 returns Figure 7.22. Taking too high a power can lead to overfitting, for example for degree 12.



Here is an example of a simple logistic regression problem. A group of hikers trains to scale Mount Rainier. For each hiker, we know the number of months they train, as well as whether or not (p equal 1 or 0) they successfully scale the peak.

More generally, we may only know the amount of training time x, and the *probability* p that the hiker succeeded, where now $0 \leq p \leq 1$.

For example, the data may be as in Figure 7.3, where p_k equals 1 or 0.

x	p	x	p	x	p	x	p	x	p
0.5	0	.75	0	1.0	0	1.25	0	1.5	0
1.75	0	1.75	1	2.0	0	2.25	1	2.5	0
2.75	1	3.0	0	3.25	1	3.5	0	4.0	1
4.25	1	4.5	1	4.75	1	5.0	1	5.5	1

Table 7.3 Months trained and outcomes.

As stated, the samples of this dataset are scalars, and the dataset is one-dimensional (Figure 7.24).

Plotting the dataset on the (x, q) plane, the goal is to fit a curve

$$q = \sigma(mx + b) \tag{7.7.7}$$

as in Figure 7.25.

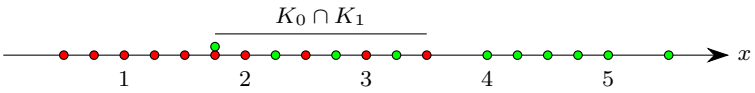


Fig. 7.24 Hikers dataset: x.

Since this is logistic regression, we can apply the two-class result from the previous section: The dataset is one-dimensional, so a hyperplane is just a point, a threshold. Neither class lies in a hyperplane, and the dataset is inseparable (Figure 7.24). Hence logistic regression is trainable, and gradient descent is guaranteed to converge to the unique optimal weight (m^*, b^*).

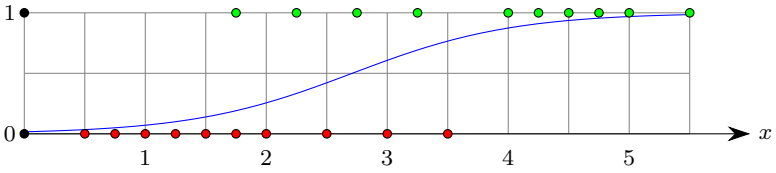


Fig. 7.25 Hikers dataset: (x, q) [37].

Based on (7.6.1) and (7.6.7), here is the descent code. Here `reshape` modifies `dataset` from an array of shape `(N,)` (a vector) to an array of shape `(N,1)` (a matrix with a single row).

```
from scipy.special import expit as sigma
from numpy.random import default_rng as rng

dataset = array([0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 1.75, 2.0,
    ↪ 2.25, 2.5, 2.75, 3.0, 3.25, 3.5, 4.0, 4.25, 4.5, 4.75,
    ↪ 5.0, 5.5])
N = len(dataset)
dataset = dataset.reshape(N,1)
labels = array([0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1])

# w = (m, b)

def q(x,w):
   m, b = w[:-1], w[-1]
   return sigma(dot(m,x) + b)

def gradient(dataset,labels,w):
   G = array( [ (q(x,w)-p) * hstack([x,1]) for x,p in
     ↪ zip(dataset, labels) ])
   return mean(G, axis = 0)

def logreg(dataset,labels):
```

```
    Q = cov(dataset,bias = True)
    mu = mean(dataset,axis = 0)
    L = 1 + dot(mu,mu) + trace(Q)
    t = 1/L # short-step learning rate
    d = len(mu)
    w = rng().random(d+1) # initial m,b
    g = gradient(dataset,labels,w)
    num_iter = 0
    print("starting m,b: ",w)
    while not allclose(g,0):
       w -=  t * g
       g = gradient(dataset,labels,w)
       num_iter += 1
    print("minimizer m,b: ",w)
    print("gradient at minimizer: ",g)
    print("num iter: ", num_iter)

logreg(dataset,labels)
```

This code returns $m^* = 1.50464542$, $b^* = -4.0777134$. These values are used to graph the sigmoid in Figure 7.25.



Even though we are done, we take the long way and incorporate the bias. With the bias treated as an extra input, the augmented dataset is

$$(x_1, 1), (x_2, 1), \ldots, (x_N, 1), \qquad N = 20,$$

resulting in Figure 7.26. In Figure 7.26, the shaded area is bounded by the vectors corresponding to the overlap between the classes.

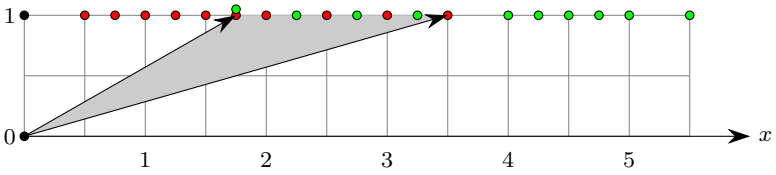


Fig. 7.26 Hikers dataset: (x, x_0).

Let $\sigma(y)$ be the sigmoid function (5.2.16). Then, as in the previous section, the goal is to minimize the loss function

$$J(m, b) = \frac{1}{N} \sum_{k=1}^{N} I(p_k, q_k), \qquad q_k = \sigma(y_k), \qquad y_k = mx_k + b, \qquad (7.7.8)$$

Once we have the minimizer (m^*, b^*), we have the best-fit curve (7.7.7). If the targets p are one-hot encoded, the dataset is as in Figure 7.4.

Each sample $(x, 1)$ in the augmented dataset is in $\mathbf{R}^2$, and each target is one-hot encoded as $(p, 1-p)$. Since the weight matrix is centered, we have

$$W = (M, b) = \begin{pmatrix} m & b \\ -m & -b \end{pmatrix}$$

and

$$\begin{pmatrix} y \\ -y \end{pmatrix} = W \begin{pmatrix} x \\ 1 \end{pmatrix} = \begin{pmatrix} mx + b \\ -(mx + b) \end{pmatrix}.$$

x	p	x	p	x	p	x	p	x	p
0.5	(1,0)	.75	(1,0)	1.0	(1,0)	1.25	(1,0)	1.5	(1,0)
1.75	(1,0)	1.75	(0,1)	2.0	(1,0)	2.25	(0,1)	2.5	(1,0)
2.75	(0,1)	3.0	(1,0)	3.25	(0,1)	3.5	(1,0)	4.0	(0,1)
4.25	(0,1)	4.5	(0,1)	4.75	(0,1)	5.0	(0,1)	5.5	(0,1)

Table 7.4 Months trained and one-hot encoded outcomes.

Since $y = Wx$, the outputs must satisfy $y_1 = y$ and $y_2 = -y$. This leads to a neural network with two inputs and two outputs (Figure 7.27).

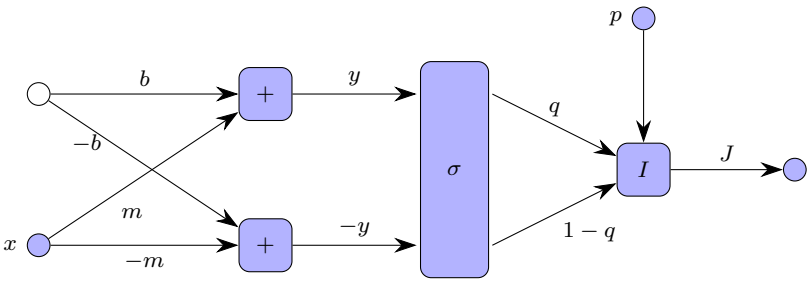


Fig. 7.27 Neural network for hiker outcomes.

Since here $d = 2$, the networks in Figures 7.27 and 7.28 are equivalent. In Figure 7.27, σ is the softmax function, I is given by (5.6.7), and p, q are probability vectors. In Figure 7.28, σ is the sigmoid function, I is given by (4.2.2), and p, q are probability scalars.

Figure 7.25 is a plot of x against q. However, the dataset, with the bias input included, has two inputs x, 1 and one output q, and should be plotted in three dimensions $(x, 1, q)$. Then (Figure 7.29) the augmented samples lie on the line $(x, 1)$ in the horizontal plane, and q is on the vertical axis.

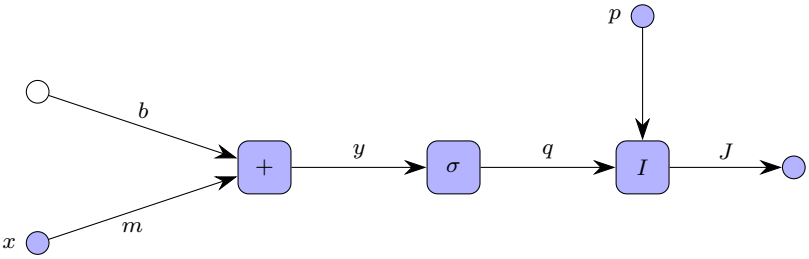


Fig. 7.28 Equivalent neural network for hiker outcomes.

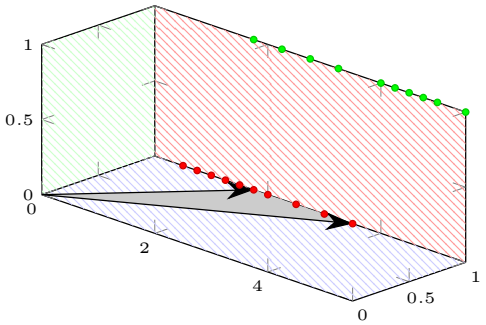


Fig. 7.29 Hiker dataset: (x, x_0, q).

The Iris dataset consists of 150 samples divided into three classes, leading to three convex hulls K_0, K_1, K_2 in $\mathbf{R}^4$. The two-dimensional projection (onto the top two principal components) of the dataset is weakly inseparable but not strongly inseparable (Figure 7.30). It follows we have no guarantee the mean logistic loss is proper.

On the other hand, the MNIST dataset consists of 60,000 samples divided into ten classes. The two-dimensional projection (onto the top two principal components) of the dataset is strongly inseparable (Figure 7.31). Therefore the mean logistic loss of the two-dimensional projection is proper.

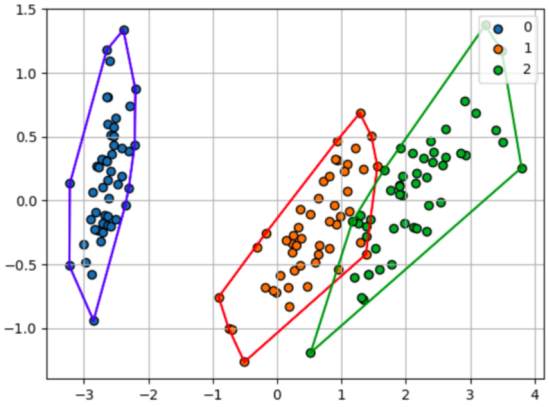


Fig. 7.30 Convex hulls of Iris classes in $\mathbf{R}^2$.

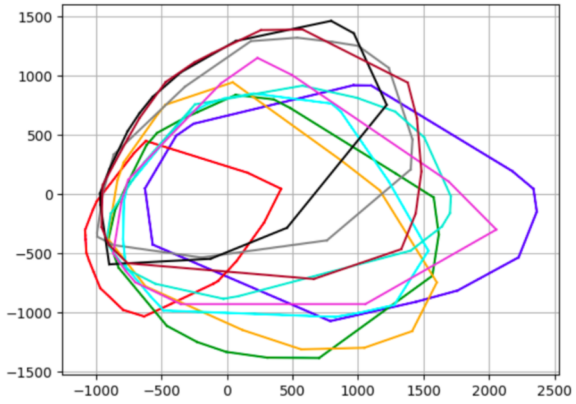


Fig. 7.31 Convex hulls of MNIST classes in $\mathbf{R}^2$.

7.8 Strong Convexity

In this section, we work with loss functions that are strongly convex. While this is not always the case, this assumption is a base case against which we can test different optimization or training models.

By strongly convex, we mean there are positive constants m and L satisfying

$$m \leq D^2 f(w) \leq L, \qquad \text{for every } w. \tag{7.8.1}$$

Recall this means the eigenvalues of the symmetric matrix $D^2 f(w)$ are between L and m. In this situation, the *condition number* $r = m/L$ is between

zero and one: $0 < r \leq 1$. (In the literature, the condition number is often defined as L/m.)

In §7.3, we saw that basic gradient descent converged to a critical point. If $f(w)$ is strongly convex, then $f(w)$ is proper and there is exactly one critical point, the global minimum w^*.

Let w_0, w_1, w_2, ... be a gradient descent sequence. Recall the learning rates t_n are *short-step* if they satisfy $t_n \leq 1/L$. Then from §7.3 we have

Gradient Descent on a Strongly Convex Function

Every gradient descent sequence with constant short-step learning rate converges to the global minimum w^*.

The simplest example of a strongly convex loss function is the *quadratic case*

$$f(w) = \frac{1}{2} w \cdot Qw - b \cdot w. \tag{7.8.2}$$

Here Q is a symmetric matrix with positive eigenvalues, and m and L are the least and greatest eigenvalues of Q.

By (4.3.5), the gradient for this example is $Qw - b$. Hence the minimizer is the unique solution $w^* = Q^{-1}b$ of the linear system $Qw = b$. Thus gradient descent is a natural tool for solving linear systems and computing inverses, at least for variance matrices Q.

Let $f(w)$ be a loss function satisfying (7.8.1). By (4.5.18), $f(w)$ lies between two quadratics,

$$\frac{m}{2}|w - w^*|^2 \leq f(w) - f(w^*) \leq \frac{L}{2}|w - w^*|^2. \tag{7.8.3}$$

How far we are from our goal w^* can be measured by the error $|w - w^*|^2$. Another measure of error is $f(w) - f(w^*)$. The goal is to drive the error between w and w^* to zero.

When $f(w)$ is strongly convex, the estimate (7.8.3) shows these two error measures are equivalent. We use both measures below.



Suppose the learning rate t satisfies $tL \leq 1$. Inserting $x = w$ and $a = w^*$ in the left half of (4.5.20) and using $\nabla f(w^*) = 0$ implies

$$f(w) \leq f(w^*) + \frac{1}{2m}|\nabla f(w)|^2. \tag{7.8.4}$$

Let $f_1(w) = f(w) - f(w^*)$. Combining this inequality with the short-step inequality (7.3.4) leads to

$$f_1(w^+) \leq (1 - mt) f_1(w). \tag{7.8.5}$$

Now assume $t = 1/L$ and set $r = m/L$. Iterating this implies

$$f_1(w_2) \le (1-r)f_1(w_1) \le (1-r)(1-r)f_1(w_0) = (1-r)^2 f_1(w_0).$$

Continuing in this manner leads to the basic gradient descent result GD-I.

Gradient Descent I (GD-I)

Let $r = m/L$ and set $f_1(w) = f(w) - f(w^*)$. Then the descent sequence $w_0, w_1, w_2, \ldots$ given by basic gradient descent (7.3.1) with learning rate

$$t = \frac{1}{L}$$

converges to w^* at the rate

$$f_1(w_n) \le (1-r)^n f_1(w_0), \qquad n = 1, 2, \ldots. \tag{7.8.6}$$



It turns out (7.8.6) extends naturally to the case of variable learning rates $t_1, t_2, \ldots$. For gradient descent, there is no need for this, since constant short-step learning rates provide adequate convergence (7.8.6). However, for stochastic gradient descent below, learning rates must be tuned to converge to zero with the time step.

Because of this, we generalize (7.8.6) to the setting of varying learning rates $t_1, t_2, \ldots$ converging to zero. In this case, a natural condition on the learning rates is

$$t_1 + t_2 + t_3 + \cdots = \infty. \tag{7.8.7}$$

This condition enables the gradient descent sequence to reach a minimizer w^*, no matter how far w^* is from the initial weight w_0.

As we saw in §7.3, the simplest learning rates converging to zero and satisfying (7.8.7) are the harmonic learning rates (7.3.12).

Assume the learning rates t_n are short-step, $t_n \le 1/L$, for $n = 1, 2, \ldots$, and let p_n be the product

$$p_n = \prod_{k=1}^{n} (1 - mt_k) = (1 - mt_1)(1 - mt_2)\ldots(1 - mt_n). \tag{7.8.8}$$

Then $p_n \ge (1-r)^n$, $n = 1, 2, \ldots$.

By repeating the calculation leading to (7.8.6), we obtain

$$f_1(w_n) \le p_n f_1(w_0), \qquad n = 1, 2, \ldots \tag{7.8.9}$$

From here, (7.8.7) implies w_n converges to w^*, see Exercise **7.8.5**.



Using coercivity of the gradient (4.5.17), we can obtain an improved result GD-II. Let $g = \nabla f(w)$.

By basic gradient descent (7.3.1), $w^+ = w - tg$, hence

$$|w^+ - w^*|^2 = |w - w^*|^2 - 2tg \cdot (w - w^*) + t^2|g|^2.$$

Inserting $x = w$ and $a = w^*$ in (4.5.17) and using $\nabla f(w^*) = 0$ implies

$$g \cdot (w - w^*) \geq \frac{mL}{m+L}|w - w^*|^2 + \frac{1}{m+L}|g|^2.$$

Inserting this inequality in the last equation,

$$|w^+ - w^*|^2 \leq \left(1 - 2t\frac{mL}{m+L}\right)|w - w^*|^2 + \left(t^2 - \frac{2t}{m+L}\right)|g|^2.$$

Let $r = m/L$. Setting the learning rate at $t = 2/(m+L)$, the rightmost term vanishes, yielding

$$|w^+ - w^*|^2 \leq \left(1 - 2t\frac{mL}{m+L}\right)|w - w^*|^2 = \left(\frac{1-r}{1+r}\right)^2 |w - w^*|^2.$$

Iterating this, we obtain

Gradient Descent II (GD-II)

Let $r = m/L$. Then the descent sequence w_0, w_1, w_2, ... given by basic gradient descent (7.3.1) with learning rate

$$t = \frac{2}{m+L}$$

converges to w^* at the rate

$$|w_n - w^*|^2 \leq \left(\frac{1-r}{1+r}\right)^{2n} |w_0 - w^*|^2 \qquad n = 1, 2, \ldots. \tag{7.8.10}$$

GD-II improves GD-I in two ways: Since $m < L$, the learning rate is larger,

$$\frac{2}{m+L} > \frac{1}{L},$$

and the convergence rate is smaller,

$$\left(\frac{1-r}{1+r}\right)^2 < (1-r),$$

implying faster convergence.

For example, if $L = 6$ and $m = 2$, then $r = 1/3$, the learning rates are $1/6$ versus $1/4$, and the convergence rates are $2/3$ versus $1/4$. Even though GD-II improves GD-I, the improvement is not substantial. In the next section, we use momentum to derive better convergence rates.



When the loss function is quadratic, we can be more explicit. Let $g = \nabla f(w)$ be the gradient of the loss function at a point w. Then the line passing through w in the direction of g is $w - tg$. When the loss function is quadratic (7.8.2), $f(w - tg)$ is a quadratic function of the scalar variable t. In this case, the minimizer t along the line $w - tg$ is explicitly computable as

$$t^* = \frac{g \cdot g}{g \cdot Qg}. \tag{7.8.11}$$

This choice of learning rate leads to gradient descent with varying rates t_1, t_2, ... As a consequence, in this case, one can show the error $f_1(w) = f(w) - f(w^*)$ is lowered as follows,

$$f_1(w^+) = \left(1 - \frac{1}{(u \cdot Qu)(u \cdot Q^{-1}u)}\right) f_1(w), \qquad u = \frac{g}{|g|}. \tag{7.8.12}$$

To estimate the rate of descent, we use

Kantorovich's Inequality [20]

If Q is a symmetric $d \times d$ matrix with positive top and bottom eigenvalues L and m, then

$$1 \leq (u \cdot Qu)(u \cdot Q^{-1}u) \leq \frac{(m + L)^2}{4mL}$$

for every unit vector u.

Using this, one can show that here the convergence rate is also (7.8.10). Thus, after all this work, there is no advantage here, it simpler to stick with GD-II! Nevertheless, the idea here, the *line-search for a minimizer*, is a sound one, and is useful in some situations. Kantorovich's inequality is derived in Exercise **4.5.9**.



We turn now to the analysis of the stochastic gradient descent (SGD) algorithm. Here we have a sampled loss function $F(w, v)$, depending on a sample v, and the mean loss function is

$$f(w) = E(F(w, V)), \tag{7.8.13}$$

for some random variable V. Motivation for this set-up is in §7.3. We assume strong convexity of $f(w)$.

Then, under reasonable conditions, by linearity of the expectation (5.3.4), the mean of $\nabla F(w, V)$ is $\nabla f(w)$,

$$\nabla f(w) = E(\nabla F(w, V)), \qquad \text{for every } w. \tag{7.8.14}$$

For example, this holds for (7.3.5) and (7.3.6).

In addition to (7.8.1), we assume the (total) variance of $\nabla F(w, V)$ is bounded,

$$Var(\nabla F(w, V)) \le \sigma^2, \qquad \text{for every } w. \tag{7.8.15}$$

Since the second moment equals the variance plus the mean squared, and by (7.8.14) the mean is $\nabla f(w)$, (7.8.15) and strong convexity (4.5.21) imply

$$E(|\nabla F(w, V)|^2) \le L^2|w - w^*|^2 + \sigma^2, \qquad \text{for every } w. \tag{7.8.16}$$

This holds for the parabola case in §7.3, and for (7.3.5) and (7.3.6).

Thus (7.8.14) and (7.8.16) hold for stochastic sample training and mini-batch sample training, as explained in §7.4: For those settings, the only real assumption we are making is strong convexity of the mean loss $f(w)$.

The basic SGD descent step is

$$W^+ = W - t\nabla F(W, V). \tag{7.8.17}$$

Here V, W, and W^+ are in uppercase because they are random variables.

At every step in (7.8.17), we assume W and V are independent, so we have the conditional expectations (see the Exercises in §5.3)

$$E(\nabla F(W, V) \mid W = w) = E(\nabla F(w, V)) \tag{7.8.18}$$

and

$$E\left(|\nabla F(W, V)|^2 \mid W = w\right) = E\left(|\nabla F(w, V)|^2\right). \tag{7.8.19}$$

We proceed as in the derivation of GD-II, and start with

$$|W^+ - w^*|^2 = |W - w^*|^2 + 2(W^+ - W) \cdot (W - w^*) + |W^+ - W|^2.$$

Inserting (7.8.17) leads to

$$|W^+ - w^*|^2 = |W - w^*|^2 - 2t\nabla F(W, V) \cdot (W - w^*) + t^2|\nabla F(W, V)|^2.$$

Taking the expectation, conditional on $W = w$, and appealing to (7.8.16), (7.8.18), and (7.8.19),

$$E\left(|W^+ - w^*|^2 \mid W = w\right) \le |w - w^*|^2(1 + t^2L^2) - 2t(w - w^*) \cdot \nabla f(w) + t^2\sigma^2.$$

Since $\nabla f(w^*) = 0$, by (4.5.16) with $x = w$ and $a = w^*$,

$$E\left(|W^+ - w^*|^2 \mid W = w\right) \le |w - w^*|^2(1 - 2mt + t^2L^2) + t^2\sigma^2.$$

Inserting $w = W$ and taking the expectation,

$$E\left(|W^+ - w^*|^2\right) \le E\left(|W - w^*|^2\right)(1 - 2mt + t^2L^2) + t^2\sigma^2. \tag{7.8.20}$$

Let $V_1, V_2, \ldots$ be an i.i.d. sequence of samples of V, and let $W_1, W_2, \ldots$, be the SGD sequence generated by

$$W_n = W_{n-1} - t_n \nabla F(W_{n-1}, V_n), \qquad W_0 = w_0. \tag{7.8.21}$$

Then W_{n-1} depends only on $V_1, V_2, \ldots, V_{n-1}$, so W_{n-1} and V_n are independent.

Inserting $W = W_{n-1}$, $W^+ = W_n$, $t = t_n$, in (7.8.20), we obtain

$$E\left(|W_n - w^*|^2\right) \le E\left(|W_{n-1} - w^*|^2\right)(1 - 2mt_n + t_n^2L^2) + t_n^2\sigma^2, \tag{7.8.22}$$

for $n = 1, 2, \ldots$.

Let p_n and s_n be the product and sum

$$p_n = \prod_{k=1}^{n}(1 - 2mt_k + L^2t_k^2), \qquad s_n = \sum_{k=1}^{n} t_k^2 \cdot \frac{p_n}{p_k}.$$

Under the assumptions

$$t_1 + t_2 + \cdots = \infty \qquad \text{and} \qquad t_1^2 + t_2^2 + \cdots < \infty, \tag{7.8.23}$$

$p_n \to 0$ (Exercise **A.7.7**). Repeating the argument at the end of §7.3, $p_n \to 0$ implies $s_n \to 0$.

When t_k is short-step, $p_n > 0$. Hence we can iterate (7.8.22),

$$E\left(|W_n - w^*|^2\right) \le p_n \cdot |w_0 - w^*|^2 + \sigma^2 \cdot s_n, \qquad n = 1, 2, \ldots. \tag{7.8.24}$$

We conclude

Stochastic Gradient Descent (SGD)

Assume the loss function $f(w)$, given by (7.8.13) for some random variable V, is strongly convex, and assume (7.8.14), and (7.8.16). Let $V_1, V_2, \ldots$ be an i.i.d. sequence, and let $W_1, W_2, \ldots$ be generated by (7.8.21). If the learning rates are short-step and satisfy (7.8.23), then

$$E\left(|W_n - w^*|^2\right) \to 0 \qquad \text{as} \qquad n \to \infty. \tag{7.8.25}$$

Here w^* is the global minimum of the loss function $f(w)$.

The takeaway here is one only needs strong convexity for the mean loss function $f(w)$, no convexity assumptions are made on the individual sample loss functions $F(w, v)$. The sample loss functions can have bumps and kinks in them, and need not be convex everywhere.

Exercises

Exercise 7.8.1 With $f(w)$ given by (7.8.2) and $g = Qw - b$, show $f(w) - f(w^*) = g \cdot Q^{-1}g/2$.

Exercise 7.8.2 With $f(w)$ given by (7.8.2) and $g = Qw - b$, show $f(w - tg)$ is minimized at t^* given by (7.8.11).

Exercise 7.8.3 Verify (7.8.12).

Exercise 7.8.4 Use Kantorovich's inequality to show the error decrease (7.8.12) is no greater than $(1-r)^2/(1+r)^2$, which is the rate in (7.8.10).

Exercise 7.8.5 Suppose the short-step learning rates $t_1, t_2, \ldots$ satisfy

$$t_1 + t_2 + t_3 + \cdots = \infty.$$

If $w_0, w_1, w_2, \ldots$ follows basic gradient descent (7.3.1), use (7.8.9) and Exercise **A.7.7** to show $f(w_n) \to f(w^*)$. Conclude $w_n \to w^*$.

Exercise 7.8.6 Show (7.8.16) follows from (7.8.15).

7.9 Accelerated Gradient Descent

In this section, we modify the gradient descent method by adding a term incorporating previous gradients, leading to *gradient descent with momentum*.

Recall in a descent sequence, the current point is w, the next point is w^+, and the previous point is w^-.

In gradient descent with momentum, we add a *momentum term* to the current point w, obtaining the *lookahead point*

$$w^\circ = w + s(w - w^-). \tag{7.9.1}$$

Here $s > 0$ is the *decay rate*. The momentum term reflects the direction induced by the previous step. Because this mimics the behavior of a ball rolling downhill, gradient descent with momentum is also called *heavy ball descent*.

Then the descent sequence $w_0, w_1, w_2, \ldots$ is generated by

Momentum Gradient Descent Step

$$w^+ = w - t\nabla f(w) + s(w - w^-). \tag{7.9.2}$$

Here we have two hyperparameters, the learning rate and the decay rate.



We establish convergence of momentum gradient descent in the above form, due to Polyak [26], and in the modified form due to Nesterov [22]. The first result is valid for strongly convex quadratics, and the second for all strongly convex functions.

Assume $f(x)$ is quadratic (7.8.2). In this case, $\nabla f(w) = Qw - b$, and the sequence satisfies the recursion

$$w_{n+1} = w_n - t(Qw_n - b) + s(w_n - w_{n-1}), \qquad n = 0, 1, 2, \dots. \tag{7.9.3}$$

To initialize the recursion, we set $w_{-1} = w_0^- = w_0$. This implies $w_1 = w_0 - t(Qw_0 - b)$.

We measure the convergence using the error $E(w) = |w - w^*|^2$, and we assume $m < Q < L$ strictly, in the sense every eigenvalue λ of Q satisfies $m < \lambda < L$; then

$$C = \max_{\lambda} \frac{(L - m)(L - m)}{(L - \lambda)(\lambda - m)} \tag{7.9.4}$$

is finite. As before, we set $r = m/L$.

Let v be an eigenvector of Q with eigenvalue $\lambda > 0$. To solve (7.9.3), we assume a solution of the form

$$w_n = w^* + \rho^n v, \qquad Qv = \lambda v. \tag{7.9.5}$$

Inserting this into (7.9.3) and using $Qw^* = b$ leads to the quadratic equation

$$\rho^2 = (1 - t\lambda + s)\rho - s.$$

By the quadratic formula,

$$\rho = \rho_\pm = \frac{(1 - \lambda t + s) \pm \sqrt{(1 - \lambda t + s)^2 - 4s}}{2}.$$

Assume the discriminant $(1 - \lambda t + s)^2 - 4s$ is negative. This happens when

$$\frac{(1 - \sqrt{s})^2}{\lambda} < t < \frac{(1 + \sqrt{s})^2}{\lambda}. \tag{7.9.6}$$

If we assume

$$\frac{(1 - \sqrt{s})^2}{m} \le t \le \frac{(1 + \sqrt{s})^2}{L}, \tag{7.9.7}$$

then (7.9.6) holds for every eigenvalue λ of Q.

Multiplying (7.9.7) by λ,

$$\begin{aligned} 4s - (1 - \lambda t + s)^2 &= (2\sqrt{s} - 1 + \lambda t - s)(2\sqrt{s} + 1 - \lambda t + s) \\ &= (\lambda t - (1 - \sqrt{s})^2)(-\lambda t + (1 + \sqrt{s})^2), \end{aligned}$$

so

$$\begin{aligned} 4s - (1 - \lambda t + s)^2 &\geq (1 - \sqrt{s})^2 (1 + \sqrt{s})^2 \left(\frac{\lambda}{m} - 1\right)\left(1 - \frac{\lambda}{L}\right) \\ &= (1 - s)^2 \frac{(L - \lambda)(\lambda - m)}{mL} \geq \frac{(1 - s)^2 (L - m)^2}{mLC}. \end{aligned} \tag{7.9.8}$$

When (7.9.6) holds, the roots are conjugate complex numbers ρ, $\bar{\rho}$, where

$$\rho = x + iy = \frac{(1 - \lambda t + s) + i\sqrt{-(1 - \lambda t + s)^2 + 4s}}{2}. \tag{7.9.9}$$

It follows the absolute value of ρ equals

$$|\rho| = \sqrt{x^2 + y^2} = \sqrt{s}.$$

To obtain the fastest convergence, we choose s and t to minimize $|\rho| = \sqrt{s}$, while still satisfying (7.9.7). This forces (7.9.7) to be an equality,

$$\frac{(1 - \sqrt{s})^2}{m} = t = \frac{(1 + \sqrt{s})^2}{L}. \tag{7.9.10}$$

These are two equations in two unknowns s, t. Solving, we obtain

$$\sqrt{s} = \frac{1 - \sqrt{r}}{1 + \sqrt{r}}, \qquad t = \frac{1}{L} \cdot \frac{4}{(1 + \sqrt{r})^2}. \tag{7.9.11}$$

Let $\tilde{w}_n = w_n - w^*$. Since $Qw_n - b = Q\tilde{w}_n$, (7.9.3) is a 2-step *linear* recursion in the variables $\tilde{w}_n$. Therefore the general solution depends on two constants A, B.

Let $\lambda_1, \lambda_2, \ldots, \lambda_d$ be the eigenvalues of Q and let $v_1, v_2, \ldots, v_d$ be the corresponding orthonormal basis of eigenvectors.

Since (7.9.3) is a 2-step *vector* linear recursion, A and B are vectors, and the general solution depends on $2d$ constants A_k, B_k, $k = 1, 2, \ldots, d$.

If ρ_k, $k = 1, 2, \ldots, d$, are the corresponding roots (7.9.9), then (7.9.5) is a solution of (7.9.3) for each of $2d$ roots $\rho = \rho_k$, $\rho = \bar{\rho}_k$, $k = 1, 2, \ldots, d$. Therefore the linear combination

$$w_n = w^* + \sum_{k=1}^{d} \left(A_k \rho_k^n + B_k \bar{\rho}_k^n\right) v_k, \qquad n = 0, 1, 2, \ldots \tag{7.9.12}$$

is the general solution of (7.9.3). Inserting $n = 0$ and $n = 1$ into (7.9.12), then taking the dot product of the result with v_k, we obtain two linear equations for two unknowns A_k, B_k,

$$\begin{aligned} A_k + B_k &= (w_0 - w^*) \cdot v_k, \\ A_k\rho_k + B_k\bar{\rho}_k &= (w_1 - w^*) \cdot v_k = (1 - t\lambda_k)(w_0 - w^*) \cdot v_k, \end{aligned}$$

for each $k = 1, 2, \ldots, d$. Solving for A_k, B_k yields

$$A_k = \left(\frac{1 - t\lambda_k - \bar{\rho}_k}{\rho_k - \bar{\rho}_k} \right) (w_0 - w^*) \cdot v_k, \qquad B_k = \bar{A}_k.$$

Using (7.9.8), (7.9.10), and (7.9.11), one can verify

$$\left| \frac{1 - t\lambda_k - \bar{\rho}_k}{\rho - \bar{\rho}} \right|^2 \leq \frac{C\lambda}{4L(1 + \sqrt{r})^2} \leq \frac{C}{4}, \tag{7.9.13}$$

hence

$$(|A_k| + |B_k|)^2 \leq C\,|(w_0 - w^*) \cdot v_k|^2.$$

Now use (2.9.5) twice, first with $v = w_n - w^*$, then with $v = w_0 - w^*$. By (7.9.12) and the triangle inequality,

$$\begin{aligned} |w_n - w^*|^2 &= \sum_{k=1}^{d} |(w_n - w^*) \cdot v_k|^2 = \sum_{k=1}^{d} |A_k\rho_k^n + B_k\bar{\rho}_k^n|^2 \\ &\leq \sum_{k=1}^{d} (|A_k| + |B_k|)^2 |\rho_k|^{2n} \\ &\leq Cs^n \sum_{k=1}^{d} |(w_0 - w^*) \cdot v_k|^2 = Cs^n |w_0 - w^*|^2. \end{aligned}$$

This derives the following result.

Momentum Gradient Descent - Heavy Ball

Suppose the loss function $f(w)$ is quadratic (7.8.2) and strongly convex, and let $r = m/L$. Let C be given by (7.9.4). Then the descent sequence $w_0, w_1, w_2, \ldots$ given by (7.9.2) with learning rate and decay rate

$$t = \frac{1}{L} \cdot \frac{4}{(1 + \sqrt{r})^2}, \qquad s = \left(\frac{1 - \sqrt{r}}{1 + \sqrt{r}} \right)^2,$$

converges to w^* at the rate

$$|w_n - w^*|^2 \le C\left(\frac{1-\sqrt{r}}{1+\sqrt{r}}\right)^{2n} |w_0 - w^*|^2, \qquad n = 1, 2, \dots \quad (7.9.14)$$

This heavy ball descent, due to Polyak [26], is an improvement over GD-II (7.8.10), because $\sqrt{r}$ is substantially larger than r when r is small. The downside of this momentum method is that the convergence (7.9.14) is only guaranteed for $f(w)$ quadratic (7.8.2). In fact, there are examples of non-quadratic $f(w)$ where heavy ball descent does not converge to w^*. Nevertheless, this method is widely used.



The momentum method can be modified by evaluating the gradient at the lookahead point w° (7.9.1),

Momentum Descent Step With Lookahead Gradient

$$\begin{aligned} w^\circ &= w + s(w - w^-), \\ w^+ &= w^\circ - t\nabla f(w^\circ). \end{aligned} \quad (7.9.15)$$

This leads to *accelerated gradient descent*, or *momentum descent with lookahead gradient*. This result, due to Nesterov [22], is valid for any strongly convex function, not just quadratics.

The iteration (7.9.15) is in two steps, a momentum step followed by a basic gradient descent step. The momentum step takes us from the current point w to the lookahead point w°, and the gradient descent step takes us from w° to the successive point w^+.

Starting from w_0, and setting $w_{-1} = w_0$, here it turns out the loss sequence $f(w_0)$, $f(w_1)$, $f(w_2)$, ... is not always decreasing. Because of this, we seek another function $V(w)$ where the corresponding sequence $V(w_0)$, $V(w_1)$, $V(w_2)$, ... is decreasing.

To explain this, it's best to assume $w^* = 0$ and $f(w^*) = 0$. This can always be arranged by translating the coordinate system. Then it turns out

$$V(w) = f(w) + \frac{L}{2}|w - \rho w^-|^2, \quad (7.9.16)$$

with a suitable choice of ρ, does the job. With the choices

$$t = \frac{1}{L}, \qquad s = \frac{1-\sqrt{r}}{1+\sqrt{r}}, \qquad \rho = 1 - \sqrt{r},$$

we will show

$$V(w^+) \le \rho V(w). \quad (7.9.17)$$

In fact, we see below (7.9.24), (7.9.25) that V is reduced by an additional quantity proportional to the momentum term.

The choice $t = 1/L$ is a natural choice from basic gradient descent (7.3.4). The derivation of (7.9.17) below forces the choices for s and ρ.

Given a point w, while w^+ is well-defined by (7.9.15), it is not clear what w^- means. There are two ways to insert meaning here. Either evaluate $V(w)$ along a sequence $w_0, w_1, w_2, \ldots$ and set, as before, $w_n^- = w_{n-1}$, or work with the function $W(w) = V(w^+)$ instead of $V(w)$. If we assume $(w^+)^- = w$, then $W(w)$ is well-defined. With this understood, we nevertheless stick with $V(w)$ as in (7.9.16) to simplify the calculations.

We first show how (7.9.17) implies the result. Using $(w_0)^- = w_0$ and (7.8.3),

$$V(w_0) = f(w_0) + \frac{L}{2}|w_0 - \rho w_0|^2 = f(w_0) + \frac{m}{2}|w_0|^2 \le 2f(w_0).$$

Moreover $f(w) \le V(w)$. Iterating (7.9.17), we obtain

$$f(w_n) \le V(w_n) \le \rho^n V(w_0) \le 2\rho^n f(w_0).$$

This derives

Momentum Descent - Lookahead Gradient

Let $r = m/L$ and set $f_1(w) = f(w) - f(w^*)$. Then the sequence w_0, $w_1, w_2, \ldots$ given by (7.9.15) with short-step learning rate and decay rate

$$t = \frac{1}{L}, \qquad s = \frac{1 - \sqrt{r}}{1 + \sqrt{r}}$$

converges to w^* at the rate

$$f_1(w_n) \le 2\left(1 - \sqrt{r}\right)^n f_1(w_0), \qquad n = 1, 2, \ldots. \tag{7.9.18}$$

While the convergence rate for accelerated descent is slightly worse than heavy ball descent, the value of accelerated descent is its validity for all strongly convex functions satisfying (7.8.1), and the fact, also due to Nesterov [22], that this convergence rate is best-possible among all such functions.

Now we derive (7.9.17). Assume $(w^+)^- = w$ and $w^* = 0$, $f(w^*) = 0$. We know $w^\circ = (1 + s)w - sw^-$ and $w^+ = w^\circ - tg^\circ$, where $g^\circ = \nabla f(w^\circ)$.

Assume $t \le 1/L$. Since $w^+ = w^\circ - tg^\circ$, (7.3.4) implies

$$f(w^+) \le f(w^\circ) - \frac{t}{2}|g^\circ|^2. \tag{7.9.19}$$

By (4.5.15) with $x = w$ and $a = w^\circ$,

$$f(w^\circ) \le f(w) - g^\circ \cdot (w - w^\circ) - \frac{m}{2}|w - w^\circ|^2. \tag{7.9.20}$$

By (4.5.15) with $x = w^* = 0$ and $a = w^\circ$,

$$f(w^\circ) \le g^\circ \cdot w^\circ - \frac{m}{2}|w^\circ|^2. \tag{7.9.21}$$

Now assume $t = 1/L$. Multiply (7.9.20) by ρ and (7.9.21) by $1-\rho$ and add, then insert the sum into (7.9.19). After some simplification, this yields

$$f(w^+) \le \rho f(w) + g^\circ \cdot (w^\circ - \rho w) - \frac{r}{2t}\left(\rho|w - w^\circ|^2 + (1-\rho)|w^\circ|^2\right) - \frac{t}{2}|g^\circ|^2. \tag{7.9.22}$$

Since

$$(w^\circ - \rho w) - tg^\circ = w^+ - \rho w,$$

we have

$$\frac{1}{2t}|w^+ - \rho w|^2 = \frac{1}{2t}|w^\circ - \rho w|^2 - g^\circ \cdot (w^\circ - \rho w) + \frac{t}{2}|g^\circ|^2.$$

Adding this to (7.9.22) leads to

$$V(w^+) \le \rho f(w) - \frac{r}{2t}\left(\rho|w - w^\circ|^2 + (1-\rho)|w^\circ|^2\right) + \frac{1}{2t}|w^\circ - \rho w|^2. \tag{7.9.23}$$

Let

$$R(a,b) = r\left(\rho s^2|b|^2 + (1-\rho)|a+sb|^2\right) - |(1-\rho)a + sb|^2 + \rho|(1-\rho)a + \rho b|^2.$$

Solving for $f(w)$ in (7.9.16) and inserting into (7.9.23) leads to

$$V(w^+) \le \rho V(w) - \frac{1}{2t}R(w, w - w^-). \tag{7.9.24}$$

If we can choose s and ρ so that $R(a,b)$ is a *positive* scalar multiple of $|b|^2$, then, by (7.9.24), (7.9.17) follows, completing the proof.

Based on this, we choose s, ρ to make $R(a,b)$ independent of a, which is equivalent to $\nabla_a R = 0$. But

$$\nabla_a R = 2(1-\rho)\left(\left(r - (1-\rho)^2\right)a + \left(\rho^2 - s(1-r)\right)b\right),$$

so $\nabla_a R = 0$ is two equations in two unknowns s, ρ. This leads to the choices for s and ρ made above. Once these choices are made, $s(1-r) = \rho^2$ and $\rho > s$. From this,

$$R(a,b) = R(0,b) = (rs^2 - s^2 + \rho^3)|b|^2 = \rho^2(\rho - s)|b|^2, \tag{7.9.25}$$

which is positive.

Appendix A
Auxiliary Material

Some of the material in these appendices is first seen in high school. Because repeating the exposure leads to a deeper understanding, we review it in a manner useful to us here.

We start with basic counting, and show how the factorial function leads directly to the exponential. Given its convexity and its importance for entropy (§5.2), the exponential is treated carefully (§A.3).

The other use of counting is in graph theory (§3.3), which lays the groundwork for neural networks (§7.2).

After this, we review two-dimensional geometry: Points and vectors in the plane, 2×2 matrices, and complex numbers, followed by two foundational appendices on convergence and minimizers. The last appendix has a different flavor, and covers coding SQL from within Python.

A.1 Permutations and Combinations

Suppose we have three balls in a bag, colored red, green, and blue. Suppose they are pulled out of the bag and arranged in a line. We then obtain six possibilities, listed in Figure A.1.

Why are there six possibilities? Because they are three ways of choosing the first ball, then two ways of choosing the second ball, then one way of choosing the third ball, so the total number of ways is

$$6 = 3 \times 2 \times 1.$$

In particular, we see that the number of ways *multiply*, $6 = 3 \times 2 \times 1$.

Similarly, there are $5 \times 4 \times 3 \times 2 \times 1 = 120$ ways of arranging five distinct balls. Since this pattern appears frequently, it has a name.

If n is a positive integer, then *n-factorial* is

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

O. Hijab, *Math for Data Science*, https://doi.org/10.1007/978-3-031-89707-8

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1.$$

The factorial function grows large rapidly, for example,

$$10! = 10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 3,628,800.$$

Notice also

$$(n+1)! = (n+1) \times n \times (n-1) \times \cdots \times 2 \times 1 = (n+1) \times n!,$$

and $(n+2)! = (n+2) \times (n+1)!$, and so on.

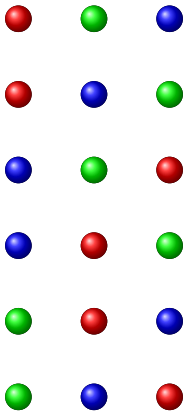


Fig. A.1 $6 = 3!$ arrangements of 3 balls.

Arrangements of n Objects

The number of arrangements of n objects is $n!$.

We also have

$$1! = 1, \qquad 0! = 1.$$

It's clear that $1! = 1$. It's less clear that $0! = 1$, but it's reasonable if you think about it: The number of arrangements of zero balls results in only one possibility — no arrangements. The code

```
from scipy.special import factorial

n = 10
factorial(n,exact = True)
```

returns 3,628,800.

More generally, we can consider the number of ways of selecting k balls from n balls, for some fixed number $k \leq n$. There are two varieties of selections that can be made: *Ordered selections* and *unordered selections*. These are *permutations* and *combinations*.



An ordered selection is a *permutation*, and the number of permutations of k objects, or a k-tuple of objects, from n objects is denoted $P(n,k)$. In particular, a permutation of n objects from n objects is an arrangement, so $P(n,n) = n!$.

The act of selecting a k-tuple from n objects can be decomposed into two parts: Selecting a $(k-1)$-tuple t, then selecting an additional object i not in t. This is the basis for the code

```
def perm_tuples(n,k):
   if k == 0: return [ () ]
   else: return [ (i,*t) for i in range(n) for t in
    ↪ perm_tuples(n,k-1) if i not in t ]

perm_tuples(5,2)
```

This code returns all k-tuples of integers x satisfying $0 \leq x < n$, and $P(n,k)$ equals `len(perm_tuples(n,k))`. For example, `perm_tuples(5,2)` returns

```
[ (0,1), (0,2), (0,3), (0,4), (1,0), (1,2), (1,3), (1,4), (2,0),
    ↪ (2,1), (2,3), (2,4), (3,0), (3,1), (3,2), (3,4), (4,0),
    ↪ (4,1), (4,2), (4,3) ]
```

In Python, $P(n,k)$ is

```
from numpy import *
set_printoptions(legacy = "1.25")

from scipy.special import perm

n, k = 5, 2
perm(n, k,exact = True)
```

so the code

```
perm(n,k,exact = True) == len(perm_tuples(n,k))
```

returns `True`.

For ordered selections, there are n choices for the first ball, $n-1$ choices for the second ball, and so on, until we have $n-(k-1)$ choices for the k-th

ball. Thus

$$P(n,k) = n \times (n-1) \times \cdots \times (n-k+1).$$

Because the selection order is taken into account, selecting ball #2 then ball #3 is considered distinct from selecting ball #3 then ball #2.

Permutation of k Objects from n Objects

The number of permutations of k objects from n objects is

$$P(n,k) = n(n-1)(n-2)\dots(n-k+1) = \frac{n!}{(n-k)!}.$$

The last formula follows by canceling,

$$\frac{n!}{(n-k)!} = \frac{n(n-1)\dots(n-k+1)(n-k)!}{(n-k)!} = n(n-1)\dots(n-k+1).$$

Note $P(x,k)$ is defined for any real number x by the same formula,

$$P(x,k) = x(x-1)(x-2)\dots(x-k+1).$$

An unordered selection is a *combination*, and the number of combinations of k objects from n objects is denoted $C(n,k)$. Here the selection order doesn't count, so the number of combinations of n objects from n objects is $C(n,n) = 1$.

When a selection of k objects is made, and the k selected objects are permuted, this results in the same unordered selection, but results in a different ordered selection. Since the number of permutations of k objects is $k!$, it follows $P(n,k)$ equals $k!$ times $C(n,k)$.

Combinations of k Objects from n Objects

The number of combinations of k objects from n objects is

$$C(n,k) = \frac{P(n,k)}{k!} = \frac{n!}{k!(n-k)!}.$$

Writing out the fraction $C(n,k)$,

$$C(n,k) = \frac{n(n-1)(n-2)\dots(n-k+1)}{1 \cdot 2 \cdot 3 \cdot \cdots \cdot k}. \tag{A.1.1}$$

The formula (A.1.1) is easy to remember: There are k terms in the numerator as well as the denominator, the factors in the denominator increase starting from 1, and the factors in the numerator decrease starting from n.

The act of selecting an unordered k-tuple from n objects is equivalent to restricting the selection to k-tuples with increasing entries. This is the basis for the code

```
def comb_tuples(n,k):
   if k == 0: return [ () ]
   elif k == 1: return perm_tuples(n,1)
   else: return [  (i, *t) for i in range(n) for t in
    ↪ comb_tuples(n,k-1) if i < t[0] ]

comb_tuples(5,2)
```

Then `comb_tuples(n,k)` returns all combinations of k integers x satisfying $0 \le x < n$, and $C(n,k)$ equals `len(comb_tuples(n,k))`.

The code for `comb_tuples(n,k)` is identical to that of `perm_tuples(n,k)`, except the tuple entries are now restricted to be in increasing order. For example, `comb_tuples(5,2)` returns

```
[ (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4),
    ↪ (3,4) ]
```

In Python, $C(n,k)$ is

```
from scipy.special import comb

n, k = 5, 2
comb(n, k,exact = True)
```

so the code

```
comb(n,k,exact = True) == len(comb_tuples(n,k))
```

returns `True`.

The number $C(n,k)$ is also called *n-choose-k* and the *binomial coefficient*.

Since $P(x,k)$ is defined for any real number x, so is $C(x,k)$:

$$C(x,k) = \frac{P(x,k)}{k!} = \frac{x(x-1)(x-2)\dots(x-k+1)}{1\cdot 2\cdot 3\cdot\dots\cdot k}.$$



An important question is the *rate of growth* of the factorial function $n!$. Attempting to answer this question leads to the exponential (§A.3) and to the entropy (§4.2). Here is how this happens.

Since $n!$ is a product of the n factors

$$1, 2, 3, \ldots, n-1, n,$$

each no larger than n, it is clear that

$$n! < n^n.$$

However, because half of the factors are less then $n/2$, we expect an approximation smaller than n^n, maybe something like $(n/2)^n$ or $(n/3)^n$.

To be systematic about it, assume

$$n! \quad \text{is approximately equal to} \quad e\left(\frac{n}{e}\right)^n \quad \text{for } n \text{ large}, \tag{A.1.2}$$

for some *constant* e. We seek the best constant e that fits here. In this approximation, we multiply by e so that (A.1.2) is an equality when $n = 1$.

Using the binomial theorem, in §A.3 we show

$$3\left(\frac{n}{3}\right)^n \le n! \le 2\left(\frac{n}{2}\right)^n, \qquad n \ge 1. \tag{A.1.3}$$

Based on this, a constant e satisfying (A.1.2) must lie between 2 and 3,

$$2 \le e \le 3.$$

To figure out the best constant e to pick, we see how much both sides of (A.1.2) increase when we replace n by $n+1$. Write (A.1.2) with $n+1$ replacing n, obtaining

$$(n+1)! \quad \text{is approximately equal to} \quad e\left(\frac{n+1}{e}\right)^{n+1} \quad \text{for } n \text{ large}. \tag{A.1.4}$$

Dividing the left sides of (A.1.2), (A.1.4) yields

$$\frac{(n+1)!}{n!} = (n+1).$$

Dividing the right sides yields

$$\frac{e((n+1)/e)^{n+1}}{e(n/e)^n} = (n+1)\cdot\frac{1}{e}\cdot\left(1+\frac{1}{n}\right)^n. \tag{A.1.5}$$

To make these quotients match as closely as possible, we should choose

$$e \approx \left(1 + \frac{1}{n}\right)^n, \qquad \text{for } n \text{ large.} \tag{A.1.6}$$

Choosing $n = 1, 2, 3, \ldots, 100, \ldots$ results in

$$e \approx 2, 2.25, 2.37, \ldots, 2.705, \ldots.$$

As $n \to \infty$, we obtain *Euler's constant* e (§A.3).

Equation (A.1.2) can be improved to *Stirling's approximation*

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n, \qquad \text{for } n \text{ large.} \tag{A.1.7}$$

This is an *asymptotic equality*. This means the ratio of the two sides approaches 1 for large n (see §A.7). Stirling's approximation is a consequence of the central limit theorem (Exercise **5.4.13**).

Stirling's approximation, meant for n large, is highly accurate: As soon as $n = 1$, Stirling's approximation is 90% accurate, and, as soon as $n = 9$, Stirling's approximation is 99% accurate.

Exercises

Exercise A.1.1 The n-th *Hermite number* is

$$H_n = \frac{(2n)!}{2^n n!}, \qquad n = 0, 1, 2, 3, \ldots$$

Use `scipy.special.factorial` to find the least n for which H_n is greater than a billion.

Exercise A.1.2 (Summation notation exercise) Let $n = 1, 2, \ldots$. Show

$$\sum_{k=n}^{\infty} (k - n) \cdot \frac{n^k}{k!} = \frac{n^{n+1}}{n!}. \tag{A.1.8}$$

(First break the sum into two sums, then write out the first few terms of each sum separately, and notice all terms but one cancel.)

Exercise A.1.3 (Summation notation exercise) Let $k = 2, 3, 4, \ldots$. Show the sum

$$\sum_{j=1}^{k-1} j = 1 + 2 + \cdots + (k - 1) = \frac{k(k-1)}{2}. \tag{A.1.9}$$

Hint - Write the sum in reverse order, then add the two sums term-by-term.

A.2 The Binomial Theorem

Let x and a be two variables. A *binomial* is an expression of the form

$$(a+x)^2, \qquad (a+x)^3, \qquad (a+x)^4, \qquad \dots$$

The *degree* of each of these binomials is 2, 3, and 4.

When binomials are expanded by multiplying out, one obtains a sum of terms. The *binomial theorem* specifies the exact pattern of the resulting sum.

Recall that

$$(a+b)(c+d) = a(c+d) + b(c+d) = ac + ad + bc + bd.$$

Similarly,

$$(a+b)(c+d+e) = a(c+d+e) + b(c+d+e) = ac + ad + ae + bc + bd + be.$$

Using this algebra, we can expand each binomial.

Expanding $(a+x)^2$ yields

$$(a+x)^2 = (a+x)(a+x) = a^2 + xa + ax + x^2 = a^2 + 2ax + x^2. \quad \text{(A.2.1)}$$

Similarly, for $(a+x)^3$, we have

$$\begin{aligned}(a+x)^3 &= (a+x)(a+x)^2 = (a+x)(a^2+2ax+x^2)\\ &= a^3 + 2a^2x + ax^2 + xa^2 + 2xax + x^3\\ &= a^3 + 3a^2x + 3ax^2 + x^3.\end{aligned} \quad \text{(A.2.2)}$$

For $(a+x)^4$, we have

$$\begin{aligned}(a+x)^4 &= (a+x)(a+x)^3 = (a+x)(a^3+3a^2x+3ax^2+x^3)\\ &= a^4 + 3a^3x + 3a^2x^2 + ax^3 + a^3x + 3a^2x^2 + 3ax^3 + x^4\\ &= a^4 + 4a^3x + 6a^2x^2 + 4ax^3 + x^4.\end{aligned} \quad \text{(A.2.3)}$$

Thus

$$\begin{aligned}(a+x)^2 &= a^2 + 2ax + x^2\\ (a+x)^3 &= a^3 + 3a^2x + 3ax^2 + x^3\\ (a+x)^4 &= a^4 + 4a^3x + 6a^2x^2 + 4ax^3 + x^4\\ (a+x)^5 &= \star a^5 + \star a^4x + \star a^3x^2 + \star a^2x^3 + \star ax^4 + \star x^5.\end{aligned} \quad \text{(A.2.4)}$$

Here $\star$ means we haven't found the coefficient yet.



There is a pattern in (A.2.4). In the first line, the powers of a are in decreasing order, 2, 1, 0, while the powers of x are in increasing order, 0, 1, 2. In the second line, the powers of a decrease from 3 to 0, while the powers of x increase from 0 to 3. In the third line, the powers of a decrease from 4 to 0, while the powers of x increase from 0 to 4.

This pattern of powers is simple and clear. Now we want to find the pattern for the coefficients in front of each term. In (A.2.4), these coefficients are $(1, 2, 1)$, $(1, 3, 3, 1)$, $(1, 4, 6, 4, 1)$, and $(\star, \star, \star, \star, \star, \star)$. These coefficients are the *binomial coefficients*.

Before we determine the pattern, we introduce a useful notation for these coefficients by writing

$$\binom{2}{0} = 1, \qquad \binom{2}{1} = 2, \qquad \binom{2}{2} = 1$$

and

$$\binom{3}{0} = 1, \qquad \binom{3}{1} = 3, \qquad \binom{3}{2} = 3, \qquad \binom{3}{3} = 1$$

and

$$\binom{4}{0} = 1, \qquad \binom{4}{1} = 4, \qquad \binom{4}{2} = 6, \qquad \binom{4}{3} = 4, \qquad \binom{4}{4} = 1$$

and

$$\binom{5}{0} = \star, \quad \binom{5}{1} = \star, \quad \binom{5}{2} = \star, \quad \binom{5}{3} = \star, \quad \binom{5}{4} = \star, \quad \binom{5}{5} = \star.$$

With this notation, the number

$$\binom{n}{k} \tag{A.2.5}$$

is the coefficient of $a^{n-k}x^k$ *when you multiply out* $(a+x)^n$. This is the binomial coefficient. Here n is the degree of the binomial, and k, which specifies the term in the resulting sum, varies from 0 to n (not 1 to n).

It is important to remember that, in this notation, the binomial $(a + x)^2$ expands into the sum of *three* terms a^2, $2ax$, x^2. These are term 0, term 1, and term 2. Alternatively, one says these are the *zeroth term*, the *first term*, and the *second term*. Thus the second term in the expansion of the binomial $(a+x)^4$ is $6a^2x^2$, and the binomial coefficient $\binom{4}{2} = 6$. In general, the binomial $(a + x)^n$ of degree n expands into a sum of $n + 1$ terms.

Since the binomial coefficient $\binom{n}{k}$ is the coefficient of $a^{n-k}x^k$ when you multiply out $(a + x)^n$, we have the binomial theorem.

Binomial Theorem

The binomial $(a+x)^n$ equals

$$\binom{n}{0}a^n + \binom{n}{1}a^{n-1}x + \binom{n}{2}a^{n-2}x^2 + \cdots + \binom{n}{n-1}ax^{n-1} + \binom{n}{n}x^n. \tag{A.2.6}$$

Using summation notation, the binomial theorem states

$$(a+x)^n = \sum_{k=0}^{n}\binom{n}{k}a^{n-k}x^k. \tag{A.2.7}$$

This result is only useful when we obtain usable formulas for $\binom{n}{k}$.



The binomial coefficient $\binom{n}{k}$ is called "n-choose-k", because it is the coefficient of the term corresponding to choosing k x's when multiplying the n factors in the product

$$(a+x)^n = (a+x)(a+x)(a+x)\ldots(a+x).$$

For example, the term $\binom{4}{2}a^2x^2$ corresponds to choosing two a's, and two x's, when multiplying the four factors in the product

$$(a+x)^4 = (a+x)(a+x)(a+x)(a+x).$$

The binomial coefficients may be arranged in a triangle, *Pascal's triangle* (Figure A.1). Can you figure out the numbers $\star$ in this triangle before peeking ahead?

Based on (A.2.10), here is code generating Pascal's triangle.

```
from numpy import *

N = 10
Comb = zeros((N,N),dtype=int)
Comb[0,0] = 1

for n in range(1,N):
   Comb[n,0] = Comb[n,n] = 1
   for k in range(1,n): Comb[n,k] = Comb[n-1,k] + Comb[n-1,k-1]

Comb
```

$n=0$:											1									
$n=1$:										1			1							
$n=2$:									1			2			1					
$n=3$:								1			3			3			1			
$n=4$:							1			4			6			4			1		
$n=5$:						1			5			10			10			5			1
$n=6$:					$\star$			6			15			20			15			6			$\star$
$n=7$:				1			$\star$			21			35			35			21			$\star$			1
$n=8$:			1			8			$\star$			56			70			56			$\star$			8			1
$n=9$:		1			9			36			$\star$			126			126			$\star$			36			9			1
$n=10$:	1			10			45			120			$\star$			252			$\star$			120			45			10			1

Table A.1 Pascal's triangle.

In Pascal's triangle, the very top row has one number in it: This is the *zeroth row* corresponding to $n = 0$ and the binomial expansion of $(a+x)^0 = 1$. The *first row* corresponds to $n = 1$; it contains the numbers $(1, 1)$, which correspond to the binomial expansion of $(a + x)^1 = 1a + 1x$. We say the *zeroth entry* ($k = 0$) in the *first row* ($n = 1$) is 1 and the *first entry* ($k = 1$) in the first row is 1. Similarly, the *zeroth entry* ($k = 0$) in the *second row* ($n = 2$) is 1, and the *second entry* ($k = 2$) in the *second row* ($n = 2$) is 1. The *second entry* ($k = 2$) in the *fourth row* ($n = 4$) is 6. For every row, the entries are counted starting from $k = 0$, and end with $k = n$, so there are $n + 1$ entries in row n. With this understood, the k-th entry in the n-th row is the binomial coefficient n-choose-k. So 10-choose-2 is

$$\binom{10}{2} = 45.$$



We can learn a lot about the binomial coefficients from this triangle. First, we have 1's all along the left edge. Next, we have 1's all along the right edge. Similarly, one step in from the left or right edge, we have the row number. Thus we have

$$\binom{n}{0} = 1 = \binom{n}{n}, \qquad \binom{n}{1} = n = \binom{n}{n-1}, \qquad n \geq 1.$$

Note also Pascal's triangle has a left-to-right symmetry: If you read off the coefficients in a particular row, you can't tell if you're reading them from left to right, or from right to left, it's the same either way: The fifth row is $(1, 5, 10, 10, 5, 1)$. In terms of our notation, this is written

$$\binom{n}{k} = \binom{n}{n-k}, \qquad 0 \le k \le n;$$

the binomial coefficients remain unchanged when k is replaced by $n - k$.

The key step in finding a formula for n-choose-k is to notice

$$(a+x)^{n+1} = (a+x)(a+x)^n.$$

Let's multiply this out when $n = 3$. From (A.2.4), we get

$$\begin{aligned}\binom{4}{0}a^4 + \binom{4}{1}a^3x + \binom{4}{2}a^2x^2 + \binom{4}{3}ax^3 + \binom{4}{4}x^4 \\ = \binom{3}{0}a^4 + \binom{3}{1}a^3x + \binom{3}{2}a^2x^2 + \binom{3}{3}ax^3 \\ + \binom{3}{0}a^3x + \binom{3}{1}a^2x^2 + \binom{3}{2}ax^3 + \binom{3}{3}x^4.\end{aligned}$$

Combining terms, this equals

$$\tbinom{3}{0}a^4 + \left(\tbinom{3}{1} + \tbinom{3}{0}\right)a^3x + \left(\tbinom{3}{2} + \tbinom{3}{1}\right)a^2x^2 + \left(\tbinom{3}{3} + \tbinom{3}{2}\right)ax^3 + \tbinom{3}{3}x^4.$$

Equating corresponding coefficients of x, we get,

$$\binom{4}{1} = \binom{3}{1} + \binom{3}{0}, \qquad \binom{4}{2} = \binom{3}{2} + \binom{3}{1}, \qquad \binom{4}{3} = \binom{3}{3} + \binom{3}{2}.$$

In general, a similar calculation establishes

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}, \qquad 1 \le k \le n. \tag{A.2.8}$$

This allows us to build Pascal's triangle (Figure A.1), where, apart from the ones on either end, each term ("the child") in a given row is the sum of the two terms ("the parents") located directly above in the previous row.



Insert $x = 1$ and $a = 1$ in the binomial theorem to get

$$2^n = \binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n-1} + \binom{n}{n}. \tag{A.2.9}$$

We conclude *the sum of the binomial coefficients along the n-th row of Pascal's triangle is* 2^n (remember n starts from 0).

Now insert $x = 1$ and $a = -1$. You get

$$0 = \binom{n}{0} - \binom{n}{1} + \binom{n}{2} - \cdots \pm \binom{n}{n-1} \pm \binom{n}{n}.$$

Hence: *the alternating*[1] *sum of the binomial coefficients along the n-th row of Pascal's triangle is zero.*



Let $C(n,k)$ be as in §A.1. We show

Binomial Coefficient

For $n \geq 0$,

$$\binom{n}{k} = C(n,k) = \frac{n!}{k!(n-k)!}, \qquad 0 \leq k \leq n.$$

To establish this, because

$$C(0,0) = 1 = \binom{0}{0}$$

and $\binom{n}{k}$ satisfies (A.2.8), it is enough to show $C(n,k)$ also satisfies (A.2.8),

$$C(n+1,k) = C(n,k) + C(n,k-1), \qquad 1 \leq k \leq n. \tag{A.2.10}$$

To establish (A.2.10), we simplify

$$\begin{aligned} C(n,k) + C(n,k-1) &= \frac{n!}{k!(n-k)!} + \frac{n!}{(k-1)!(n-k+1)!} \\ &= \frac{n!}{(k-1)!(n-k)!}\left(\frac{1}{k} + \frac{1}{n-k+1}\right) \\ &= \frac{n!(n+1)}{(k-1)!(n-k)!k(n-k+1)} \\ &= \frac{(n+1)!}{k!(n+1-k)!} = C(n+1,k). \end{aligned}$$

This establishes (A.2.10), and, consequently, equality of $\binom{n}{k}$ and $C(n,k)$.

For example,

[1] *Alternating* means the plus-minus pattern $+ - + - + - \ldots$.

$$\binom{7}{3} = \frac{7\cdot 6\cdot 5}{1\cdot 2\cdot 3} = 35 = \binom{7}{4} \qquad \text{and} \qquad \binom{10}{2} = \frac{10\cdot 9}{1\cdot 2} = 45 = \binom{10}{8}.$$

In Python, the code

```
from scipy.special import comb

n, k = 7, 3

comb(n,k), comb(n,k,exact = True)
```

returns the binomial coefficient.



The binomial coefficient $\binom{n}{k}$ makes sense even for fractional n. This can be seen from (A.1.1). For example, for $n = 1/2$ and $k = 3$,

$$\binom{1/2}{3} = \frac{\frac{1}{2}\left(\frac{1}{2}-1\right)\left(\frac{1}{2}-2\right)}{1\cdot 2\cdot 3} = \frac{(1/2)(-1/2)(-3/2)}{6} = \frac{3}{48}. \tag{A.2.11}$$

This works also for n negative,

$$\binom{-1/2}{3} = \frac{\left(-\frac{1}{2}\right)\left(-\frac{1}{2}-1\right)\left(-\frac{1}{2}-2\right)}{1\cdot 2\cdot 3} = \frac{(-1/2)(-3/2)(-5/2)}{6} = \frac{-15}{48}. \tag{A.2.12}$$

In fact, in (A.1.1), n may be any real number, for example $n = \sqrt{2}$.

A.3 The Exponential Function

In this section, our first goal is to derive (A.1.3), as promised in §A.1.

To begin, use the binomial theorem (A.1.1), (A.2.7) with $a = 1$ and $x = 1/n$, obtaining

$$\left(1+\frac{1}{n}\right)^n = \sum_{k=0}^{n}\binom{n}{k}1^{n-k}\left(\frac{1}{n}\right)^k = \sum_{k=0}^{n}\frac{1}{k!}\frac{n(n-1)(n-2)\dots(n-k+1)}{n\cdot n\cdot n\cdot\dots\cdot n}.$$

Rewriting this by pulling out the first two terms $k = 0$ and $k = 1$ leads to

$$\left(1+\frac{1}{n}\right)^n = 1+1+\sum_{k=2}^{n}\frac{1}{k!}\left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\dots\left(1-\frac{k-1}{n}\right). \tag{A.3.1}$$

From (A.3.1), we can tell a lot. First, since all terms are positive, we see

$$\left(1+\frac{1}{n}\right)^n \geq 2, \qquad n \geq 1.$$

Second, each factor in (A.3.1) is of the form

$$\left(1-\frac{j}{n}\right), \qquad 1 \leq j \leq k-1. \tag{A.3.2}$$

Since n is in the denominator, each such factor *increases* with n. Moreover, as n increases, the *number of terms* in (A.3.1) increases, hence so does the sum. We conclude

$$\left(1+\frac{1}{n}\right)^n \qquad \text{increases as } n \text{ increases.} \tag{A.3.3}$$

Third, when $k \geq 2$, we know

$$k! = k(k-1)(k-2)\dots 3 \cdot 2 \geq 2^{k-1}.$$

Since each factor in (A.3.2) is no greater than 1, by (A.3.1),

$$\left(1+\frac{1}{n}\right)^n \leq 1+1+\sum_{k=2}^{n}\frac{1}{k!} \leq 2+\sum_{k=2}^{n}\frac{1}{2^{k-1}}. \tag{A.3.4}$$

But we can show

$$\sum_{k=2}^{n}\frac{1}{2^{k-1}} = \frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots+\frac{1}{2^{n-1}} \leq 1,$$

as follows.

A *geometric sum* is a sum of the form

$$s_n = 1+a+a^2+\cdots+a^{n-1} = \sum_{k=0}^{n-1} a^k.$$

Multiplying s_n by a results in almost the same sum,

$$as_n = a+a^2+a^3+\cdots+a^{n-1}+a^n = s_n + a^n - 1,$$

yielding

$$(a-1)s_n = a^n - 1.$$

When $a \neq 1$, we may divide by $a-1$, obtaining

$$s_n = \sum_{k=0}^{n-1} a^k = 1 + a + a^2 + \cdots + a^{n-1} = \frac{a^n - 1}{a - 1}. \tag{A.3.5}$$

Inserting $a = 1/2$, we conclude

$$\sum_{k=2}^{n} \frac{1}{2^{k-1}} = \sum_{k=1}^{n-1} \frac{1}{2^k} = s_n - 1 = 2\left(1 - 2^{-n}\right) - 1 \le 1, \qquad n \ge 2.$$

By (A.3.4), we arrive at

$$2 \le \left(1 + \frac{1}{n}\right)^n \le 3, \qquad n \ge 1. \tag{A.3.6}$$

Now we use (A.3.6) to establish (A.1.3). Write (A.1.3) as $a_n \le b_n \le c_n$. When $n = 1$,

$$a_1 = b_1 = c_1.$$

Moreover, as n increases, a_n, b_n, c_n all increase. Therefore, to establish (A.1.3), it is enough to show b_n increases faster than a_n, and c_n increases faster than b_n, both as n increases.

To measure how a_n, b_n, c_n increase with n, divide the $(n+1)$-st term by the n-th term: It is enough to show

$$\frac{a_{n+1}}{a_n} \le \frac{b_{n+1}}{b_n} \le \frac{c_{n+1}}{c_n}.$$

But we already know

$$\frac{b_{n+1}}{b_n} = n + 1,$$

and, from (A.3.6),

$$\frac{a_{n+1}}{a_n} = \frac{3((n+1)/3)^{n+1}}{3(n/3)^n} = (n+1) \cdot \frac{1}{3} \cdot \left(1 + \frac{1}{n}\right)^n \le n + 1 = \frac{b_{n+1}}{b_n},$$

and, from (A.3.6) again,

$$\frac{b_{n+1}}{b_n} = n + 1 \le (n+1) \cdot \frac{1}{2} \cdot \left(1 + \frac{1}{n}\right)^n = \frac{2((n+1)/2)^{n+1}}{2(n/2)^n} = \frac{c_{n+1}}{c_n}.$$

Since we've shown b_n increases faster than a_n, and c_n increases faster than b_n, we have derived (A.1.3).



Since a bounded increasing sequence has a limit (§A.8), by (A.3.3), we have the following strengthening of (A.1.6).

Euler's Constant

The limit

$$e = \lim_{n\to\infty} \left(1 + \frac{1}{n}\right)^n \tag{A.3.7}$$

exists and satisfies $2 \le e \le 3$.

Standard properties of limits, such as

$$\lim_{n\to\infty} (a_n + b_n) = \lim_{n\to\infty} a_n + \lim_{n\to\infty} b_n, \qquad \lim_{n\to\infty} a_n b_n = \lim_{n\to\infty} a_n \lim_{n\to\infty} b_n,$$

are in §A.7. Nevertheless, the intuition is clear: (A.3.7) is saying there is a specific positive number e with

$$\left(1 + \frac{1}{n}\right)^n \approx e$$

for n large.



By definition, Euler's constant e satisfies (A.3.7). To obtain a second formula for e, insert $n = \infty$ in (A.3.1). Using $1/\infty = 0$, since the k-th term approaches $1/k!$, and since the number of terms increases with n, we obtain the second formula

$$e = 1 + 1 + \sum_{k=2}^{\infty} \frac{1}{k!}\left(1 - \frac{1}{\infty}\right)\left(1 - \frac{2}{\infty}\right)\cdots\left(1 - \frac{k-1}{\infty}\right) = \sum_{k=0}^{\infty} \frac{1}{k!}.$$

For a guarantee that plugging $n = \infty$ into (A.3.1) really makes sense, see Exercise **A.3.5**. To summarize,

Euler's Constant

Euler's constant satisfies

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} = 1 + 1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} + \frac{1}{720} + \ldots \tag{A.3.8}$$



Depositing one dollar in a bank offering 100% interest returns two dollars after one year. Depositing one dollar in a bank offering the same annual interest compounded at mid-year returns

$$\left(1+\frac{1}{2}\right)^2 = 2.25$$

dollars after one year.

Depositing one dollar in a bank offering the same annual interest compounded at n intermediate time points returns $(1+1/n)^n$ dollars after one year.

Passing to the limit, depositing one dollar in a bank and continuously compounding at an annual interest rate of 100% returns e dollars after one year. Because of this, (A.3.7) is often called the *compound-interest formula*.

Now we derive the result of continuously compounding at any specified annual interest rate x. Note here x is a proportion, not a percent. An interest rate of 30% corresponds to $x = .3$ in the exponential function.

Exponential Function

For any real number x, the limit

$$\exp x = \lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n \tag{A.3.9}$$

exists. In particular, $\exp 0 = 1$ and $\exp 1 = e$.

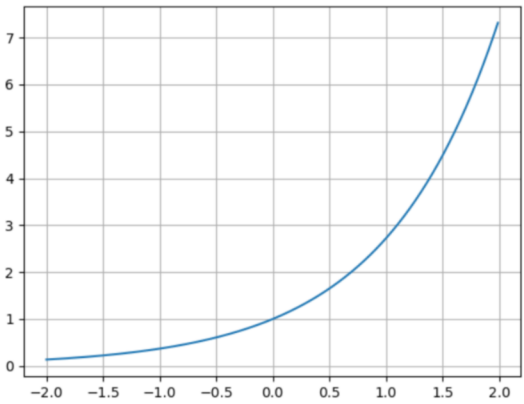


Fig. A.2 The exponential function $\exp x$.

Note, in the compound-interest interpretation, when $x > 0$, the bank is giving you interest, while, if $x < 0$, the bank is taking away interest, leading to a continual loss.

To derive this, assume first $x > 0$ is a positive real number. Then, exactly as before, using the binomial theorem,

$$\left(1 + \frac{x}{n}\right)^n, \qquad n \geq 1,$$

is increasing with n, so the limit in (A.3.9) is well-defined.

To establish the existence of the limit when $x < 0$, we first show

$$(1 - x)^n \geq 1 - nx, \qquad 0 < x < 1, n \geq 1. \tag{A.3.10}$$

This follows inductively: Each of the following inequalities is implied by the preceding one,

$$\begin{aligned}
&(1 - x) = 1 - x \\
&(1 - x)^2 = 1 - 2x + x^2 \geq 1 - 2x \\
&(1 - x)^3 = (1 - x)(1 - x)^2 \geq (1 - x)(1 - 2x) = 1 - 3x + 2x^2 \geq 1 - 3x \\
&(1 - x)^4 = (1 - x)(1 - x)^3 \geq (1 - x)(1 - 3x) = 1 - 4x + 3x^3 \geq 1 - 4x \\
&\quad \ldots \quad \ldots
\end{aligned}$$

This establishes (A.3.10) for all $n \geq 1$.

Now let x be any real number. Replacing x by x^2/n^2 in (A.3.10), we obtain

$$1 \geq \left(1 - \frac{x^2}{n^2}\right)^n \geq 1 - \frac{x^2}{n}.$$

As $n \to \infty$, both sides of this last equation approach 1, so

$$\lim_{n\to\infty} \left(1 - \frac{x^2}{n^2}\right)^n = 1. \tag{A.3.11}$$

Now let n grow without bound in

$$\left(1 + \frac{x}{n}\right)^n \left(1 - \frac{x}{n}\right)^n = \left(1 - \frac{x^2}{n^2}\right)^n.$$

Since the limit $\exp x$ is well-defined when $x > 0$, by (A.3.11), we obtain

$$\exp x \cdot \lim_{n\to\infty} \left(1 - \frac{x}{n}\right)^n = 1, \qquad x > 0.$$

This shows the limit $\exp x$ in (A.3.9) is well-defined when $x < 0$, and

$$\exp(-x) = \frac{1}{\exp x}, \qquad \text{for all } x.$$

The code

```
from numpy import *
from matplotlib.pyplot import *

x = arange(-2,2,.01)
plot(x,exp(x))

grid()
show()
```

returns Figure A.2.

Repeating the logic yielding (A.3.1), we have

$$\left(1+\frac{x}{n}\right)^n = 1 + x + \sum_{k=2}^{n} \frac{x^k}{k!}\left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\dots\left(1-\frac{k-1}{n}\right). \quad \text{(A.3.12)}$$

Letting $n \to \infty$ in (A.3.12) as before, results in the following.

Exponential Series

The exponential function is always positive and satisfies, for every real number x,

$$\exp x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720} + \dots \quad \text{(A.3.13)}$$

The graph of $\exp x$ is in Figure A.2.

We use the binomial theorem one more time to show

Law of Exponents

For real numbers x and y,

$$\exp(x+y) = \exp x \cdot \exp y.$$

To see this, multiply out the sums

$$(a_0 + a_1 + a_2 + a_3 + \dots)(b_0 + b_1 + b_2 + b_3 + \dots)$$

in a "symmetric" manner, obtaining

$$a_0b_0 + (a_0b_1 + a_1b_0) + (a_0b_2 + a_1b_1 + a_2b_0) + (a_0b_3 + a_1b_2 + a_2b_1 + a_3b_0) + \dots$$

Using summation notation, the n-th term in this last sum is

$$\sum_{k=0}^{n} a_k b_{n-k} = a_0b_n + a_1b_{n-1} + \cdots + a_{n-1}b_1 + a_nb_0.$$

Thus

$$\left(\sum_{k=0}^{\infty} a_k\right)\left(\sum_{m=0}^{\infty} b_m\right) = \sum_{n=0}^{\infty}\left(\sum_{k=0}^{n} a_k b_{n-k}\right).$$

Now insert

$$a_k = \frac{x^k}{k!}, \qquad b_{n-k} = \frac{y^{n-k}}{(n-k)!}.$$

Then the n-th term in the resulting sum equals, by the binomial theorem,

$$\sum_{k=0}^{n} a_k b_{n-k} = \sum_{k=0}^{n} \frac{x^k}{k!}\frac{y^{n-k}}{(n-k)!} = \frac{1}{n!}\sum_{k=0}^{n}\binom{n}{k}x^k y^{n-k} = \frac{1}{n!}(x+y)^n.$$

Thus

$$\exp x \cdot \exp y = \left(\sum_{k=0}^{\infty}\frac{x^k}{k!}\right)\left(\sum_{m=0}^{\infty}\frac{y^m}{m!}\right) = \sum_{n=0}^{\infty}\frac{(x+y)^n}{n!} = \exp(x+y).$$

This derives the law of exponents.

Repeating the law of exponents n times implies

$$\exp(nx) = \exp(x + x + \cdots + x) = \exp x \cdot \exp x \cdot \cdots \cdot \exp x = (\exp x)^n.$$

If we write $\sqrt[n]{x} = x^{1/n}$, replacing x by x/n yields

$$\exp(x/n) = (\exp x)^{1/n}.$$

Combining the last two equations yields

$$\exp(nx/m) = ((\exp x)^n)^{1/m} = (\exp x)^{n/m}.$$

Inserting $x = 1$ in this last equation, it follows, for any rational number $x = n/m$,

$$\exp x = \exp(1 \cdot x) = (\exp 1)^x = e^x.$$

Because of this, as a matter of convenience, we write the exponential function either way, $\exp x$ or e^x, even when x is not rational.

Exponential Notation

For any real number x,

$$e^x = \exp x.$$

Suppose $0 < r < 1$. Then $r^2 < r$, $r^3 < r$, and so on. Replacing x by rx in the exponential series (A.3.13),

$$\begin{aligned} e^{rx} &= 1 + rx + \frac{1}{2!}r^2x^2 + \frac{1}{3!}r^3x^3 + \dots \\ &< 1 + rx + \frac{1}{2!}rx^2 + \frac{1}{3!}rx^3 + \dots \\ &= 1 - r + re^x. \end{aligned} \tag{A.3.14}$$

From this we can show

Convexity of the Exponential Function

For $0 < r < 1$,

$$\exp((1-r)x + ry) < (1-r)\exp x + r\exp y. \tag{A.3.15}$$

To derive (A.3.15), replace x by $y - x$ in (A.3.14), obtaining

$$e^{r(y-x)} < 1 - r + re^{y-x}.$$

Now multiply both sides by e^x, obtaining (A.3.15).

Graphically, the convexity of the exponential functions is the fact that the line segment joining two points on the graph lies above the graph (Figure A.3).

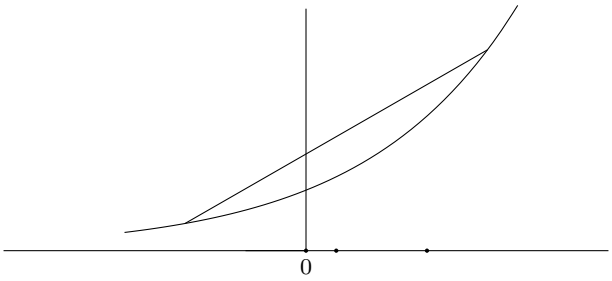


Fig. A.3 Convexity of the exponential function.

Convexity is discussed further in §4.5.

Exercises

Exercise A.3.1 Assume a bank gives 50% annual interest on deposits. After one year, what does \$1 become? Do this when the money is compounded once, twice, and at every instant during the year.

Exercise A.3.2 Assume a bank gives -50% annual interest on deposits. After one year, what does \$1 become? Do this when the money is compounded once, twice, and at every instant during the year.

Exercise A.3.3 Which of the following is correct? For n large,

$$n+1 \approx n, \qquad e^{n+1} \approx e^n, \qquad e^{e^{n+1}} \approx e^{e^n}.$$

$\approx$ is asymptotic equality (see §A.7).

Exercise A.3.4 Extend (A.3.10) by showing

$$(1-a)(1-b)(1-c) \geq 1-(a+b+c),$$

valid for a, b, c in $(0,1)$. Use this to show

$$1-\left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\left(1-\frac{3}{n}\right) \leq \frac{1}{n}+\frac{2}{n}+\frac{3}{n}.$$

This remains true for any number of factors.

Exercise A.3.5 Use the previous exercise, (A.1.9), (A.3.1), and (A.3.4), to derive the error estimate

$$0 \leq \sum_{k=0}^{n} \frac{1}{k!} - \left(1+\frac{1}{n}\right)^n \leq \frac{3}{2n}, \qquad n \geq 2.$$

This is a complete derivation of (A.3.8).

Exercise A.3.6 Replace x by x/n in (A.3.10) to show

$$1-x < e^{-x}, \qquad x > 0. \tag{A.3.16}$$

Exercise A.3.7 Use (A.3.5) to derive the *geometric series*

$$\frac{1}{1-a} = \sum_{n=0}^{\infty} a^n = 1+a+a^2+a^3+\ldots, \qquad -1 < a < 1. \tag{A.3.17}$$

Exercise A.3.8 Take the derivative of (A.3.17) to obtain

$$\frac{1}{(1-a)^2} = \sum_{n=1}^{\infty} n a^{n-1} = 1 + 2a + 3a^2 + 4a^3 + \dots, \qquad -1 < a < 1. \quad \text{(A.3.18)}$$

A.4 Two Dimensions

This section is a review of the geometry of vectors and matrices in two dimensions. This is the *cartesian plane* $\mathbf{R}^2$, also called 2-dimensional real space. The plane $\mathbf{R}^2$ is a vector space, in the sense described in §1.3.

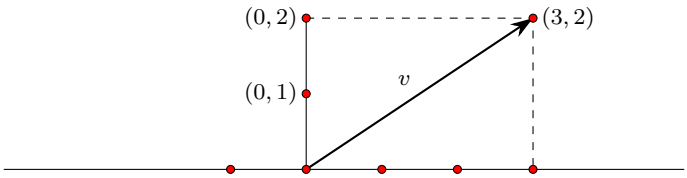


Fig. A.4 A vector v.

In the cartesian plane, a *vector* is an arrow v joining the origin to a point (Figure A.4). In this way, points and vectors are almost interchangeable, as a point x in $\mathbf{R}^d$ corresponds to the vector v starting at the origin 0 and ending at x.

In the cartesian plane, each vector v has a *shadow*. This is the triangle constructed by dropping the perpendicular from the tip of v to the x-axis, as in Figure A.5.

This cannot be done unless one first draws a horizontal line (the x-axis), then a vertical line (the y-axis). In this manner, each vector v has *cartesian* coordinates $v = (x, y)$. In Figure A.4, the coordinates of v are $(3, 2)$. In particular, the vector $0 = (0, 0)$, the *zero vector*, corresponds to the origin.

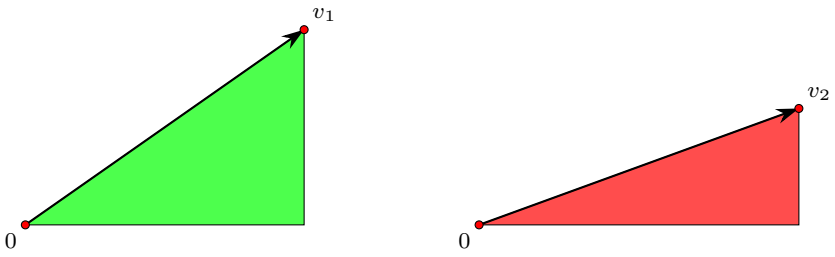


Fig. A.5 Vectors v_1 and v_2 and their shadows in the plane.

In the cartesian plane, vectors $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$ are added by adding their coordinates,

Addition of vectors

If $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$, then

$$v_1 + v_2 = (x_1 + x_2, y_1 + y_2). \tag{A.4.1}$$

Because points and vectors are interchangeable, the same formula is used for addition $P + P'$ of points P and P'.

This addition is the same as combining their shadows as in Figure A.6. In Python, lists and tuples do not add this way. Lists and tuples have to first be converted into `numpy arrays`.

```
v1 = (1,2)
v2 = (3,4)
v1 + v2 == (1+3,2+4) # returns False

v1 = [1,2]
v2 = [3,4]
v1 + v2 == [1+3,2+4] # returns False

from numpy import *
set_printoptions(legacy = "1.25")

v1 = array([1,2])
v2 = array([3,4])
v1 + v2 == array([1+3,2+4]) # returns True
```

For example, $v_1 = (-3, 1)$ and $v_2 = (2, -2)$ returns

$$v_1 + v_2 = (-3, 1) + (2, -2) = (-3 + 2, 1 - 2) = (-1, -1).$$

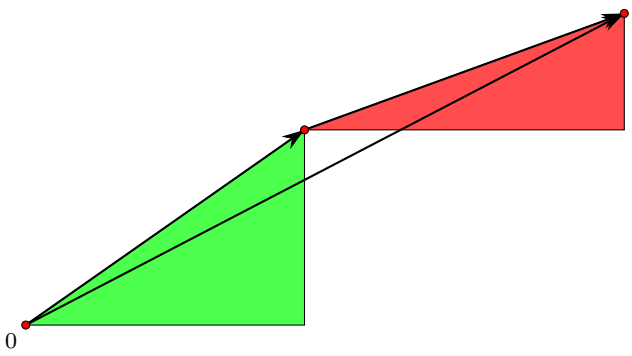


Fig. A.6 Adding v_1 and v_2

In Python, `u == v` is exact equality of values. When the entries of `u` and `v` are `int`s, this is not a problem. However, when the entries of `u` and `v` are `float`s, `u == v` may return `False` even though the floats agree to within the underlying precision of the Python code.

To remedy this, it's best to use `isclose(u,v)` for scalars or `allclose(u,v)` for arrays. This returns `True` when `u` and `v` agree to within the underlying precision. In this chapter, we ignore this point, but we are more careful starting in Chapter 2.

Scaling of vectors

If $v = (x, y)$, then

$$tv = (tx, ty).$$

A vector $v = (x, y)$ in the plane may be scaled by scaling the shadow as in Figure A.7. This is *vector scaling* by t. Note when t is negative, the shadow is also flipped. In Python, we write

```
v = array([1,2])
3*v == array([3,6]) # returns True
```

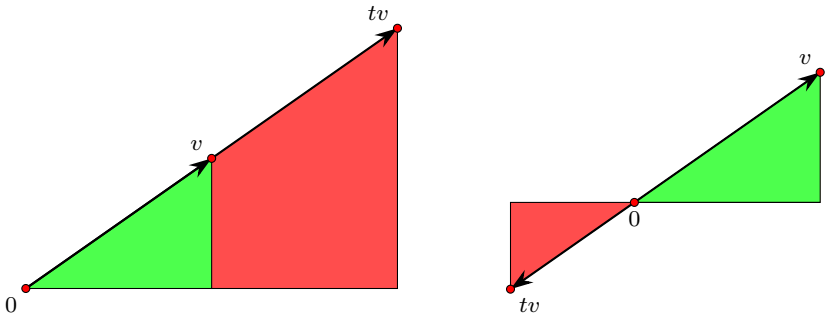


Fig. A.7 Scaling with $t = 2$ and $t = -2/3$

Given a vector v, the scalings tv of v form a line passing through the origin 0 (Figure A.9). This line is the *span* of v (more on this in §2.4). Scalings tv of v are also called *multiples* of v.

If t, s are scalars and u, v are vectors, it is easy to check *linearity*

$$t(u + v) = tu + tv \qquad \text{and} \qquad (s + t)u = su + tu,$$

and

$$t(sv) = (ts)v.$$

Thus scaling v by s, and then scaling the result by t, has the same effect as scaling v by ts, in a single step. Because points and vectors are interchangeable, the same formula tP is used for scaling points P by t.

We set $-v = (-1)v$, and define subtraction of vectors by

$$v_1 - v_2 = v_1 + (-v_2).$$

```
v1 = array([1,2])
v2 = array([3,4])
v1 - v2 == array([1-3,2-4]) # returns True
```

Subtraction of vectors

If $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$, then

$$v_1 - v_2 = (x_1 - x_2, y_1 - y_2) \tag{A.4.2}$$



Distance Formula

If $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$, then the *distance* between v_1 and v_2 is

$$|v_1 - v_2| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

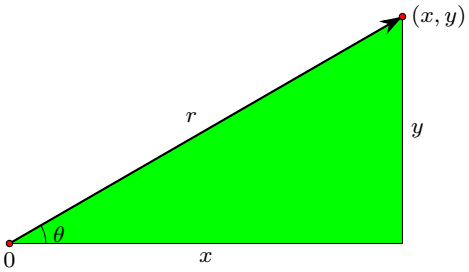


Fig. A.8 The polar representation of $v = (x, y)$.

The distance of $v = (x, y)$ to the origin $0 = (0, 0)$ is its *magnitude* or *norm* or *length*

$$r = |v| = |v - 0| = \sqrt{x^2 + y^2}.$$

In Python,

```
from scipy.linalg import norm

v = array([1,2])
norm(v) ==  sqrt(5) # returns True
```

For future use, we recall the trigonometric function $\sin\theta$ and $\cos\theta$. These are defined in terms of r, x, and y (Figure A.8)

$$x = r\cos\theta, \qquad y = r\sin\theta. \tag{A.4.3}$$

This is the *polar* representation of (x, y).

The *unit circle* consists of the vectors which are distance 1 from the origin 0. When v is on the unit circle, the magnitude of v is 1, and we say v is a *unit vector*. In this case, the line formed by the scalings of v intersects the unit circle at $\pm v$ (Figure A.9).

When v is a unit vector, $r = 1$, and (Figure A.8),

$$v = (x, y) = (\cos\theta, \sin\theta). \tag{A.4.4}$$

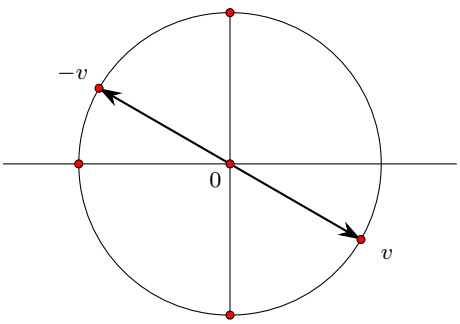


Fig. A.9 v and its antipode $-v$.

The unit circle intersects the horizontal axis at $(1, 0)$, and $(-1, 0)$, and intersects the vertical axis at $(0, 1)$, and $(0, -1)$. These four points are equally spaced on the unit circle (Figure A.9).

By the distance formula, a vector $v = (x, y)$ is a unit vector when

$$x^2 + y^2 = 1.$$

More generally, any circle with *center* $Q = (a, b)$ and *radius* r consists of points (x, y) satisfying

$$(x - a)^2 + (y - b)^2 = r^2.$$

Let R be a point on the unit circle, and let $t > 0$. From this, we see the scaled point tR is on the circle with center $(0, 0)$ and radius t. Moreover, it follows *a point P is on the circle of center Q and radius r iff $P = Q + rR$ for some R on the unit circle.*

Given this, it is easy to check

$$|tv| = |t|\,|v|$$

for any scalar t and vector v.

From this, if a vector v is unit and $r > 0$, then rv has magnitude r. If v is any vector not equal to the zero vector, then $r = |v|$ is positive, and

$$\left|\frac{1}{r}v\right| = \frac{1}{r}\,|v| = \frac{1}{r}r = 1,$$

so v/r is a unit vector.

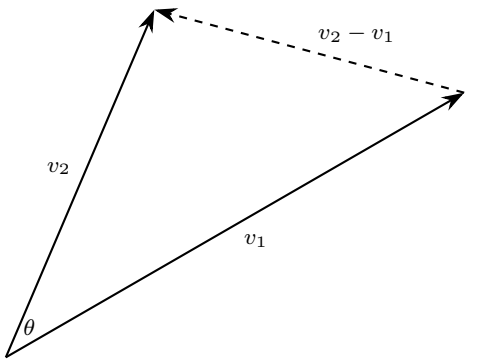


Fig. A.10 Two vectors v_1 and v_2.

Now we discuss the dot product in two dimensions. Let $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$ be two vectors in the plane. The *dot product* of v_1 and v_2 is given algebraically as

$$v_1 \cdot v_2 = x_1x_2 + y_1y_2,$$

or geometrically as

$$v_1 \cdot v_2 = |v_1|\,|v_2| \cos\theta,$$

where θ is the angle between v_1 and v_2. To show that these are the same, below we derive the

Dot Product Identity

$$x_1x_2 + y_1y_2 = v_1 \cdot v_2 = |v_1|\,|v_2|\,\cos\theta. \tag{A.4.5}$$

From the algebraic definition of dot product, we have $v \cdot v = x^2 + y^2 = |v|^2$. Since $v_2 - v_1 = (x_2 - x_1, y_2 - y_1)$,

$$|v_2 - v_1|^2 = (v_2 - v_1) \cdot (v_2 - v_1) = (x_2 - x_1)^2 + (y_2 - y_1)^2.$$

Expanding the squares, we obtain

$$|v_2 - v_1|^2 = |v_2|^2 - 2v_1 \cdot v_2 + |v_1|^2. \tag{A.4.6}$$

As an example of the usefulness of the dot product identity, the *linearity* of the dot product

$$(su + tv) \cdot w = su \cdot w + tv \cdot w$$

follows immediately from the algebraic definition, but is not at all obvious from the viewpoint of the geometric definition.



In Python, the dot product is given by `numpy.dot`,

```
v1 = array([1,2])
v2 = array([3,4])
dot(v1,v2) == 1*3 + 2*4 # returns True
```

Using the geometric definition of the dot product, we can write

$$\cos\theta = \frac{u \cdot v}{|u|\,|v|}.$$

As a consequence, we have code for the angle between two vectors (we write `Angle` to distinguish from the built-in `numpy.angle`).

```
def Angle(u,v):
   a = dot(u,v)
   b = dot(u,u)
   c = dot(v,v)
   theta = arccos(a / sqrt(b*c))
   return degrees(theta)
```

Recall that $-1 \leq \cos\theta \leq 1$. Using the dot product identity (A.4.5), we obtain the important

Cauchy-Schwarz Inequality

If u and v are any two vectors, then

$$-|u|\,|v| \leq u \cdot v \leq |u|\,|v|. \tag{A.4.7}$$

Vectors u and v are *orthogonal* or *perpendicular* if the angle between them is a right angle (90 degrees or $\pi/2$ radians). Since $\cos(\pi/2) = 0$, we see *vectors are orthogonal when their dot product is zero.*

When orthogonal vectors u and v are also unit vectors, we say u and v are *orthonormal.* As a consequence of the Cauchy-Schwarz inequality,

Parallel, Anti-Parallel, and Orthogonal Vectors

Let u and v be two vectors. Then $u \cdot v$ achieves its maximum $|u|\,|v|$ when u and v point in the same direction, achieves its minimum $|u|\,|v|$ when u and v point in opposite directions, and equals zero when u and v are orthogonal.

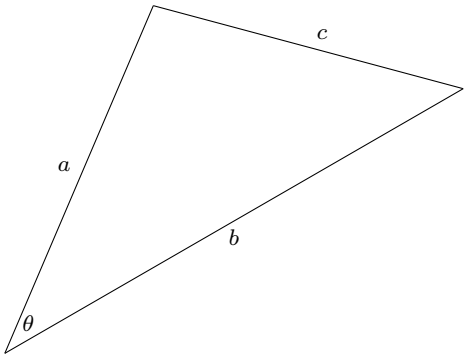


Fig. A.11 Pythagoras for general triangles.

To derive the dot product identity, we first derive Pythagoras' theorem for general triangles (Figure A.11)

$$c^2 = a^2 + b^2 - 2ab\cos\theta. \tag{A.4.8}$$

To derive (A.4.8), we drop a perpendicular to the base b, obtaining two right triangles, as in Figure A.12.

By Pythagoras applied to each right triangle,

$$a^2 = d^2 + f^2 \qquad \text{and} \qquad c^2 = e^2 + f^2.$$

Also $b = e + d$, so $e = b - d$, so

$$e^2 = (b-d)^2 = b^2 - 2bd + d^2.$$

By the definition of $\cos\theta$, $d = a\cos\theta$. Putting this all together,

$$\begin{aligned} c^2 = e^2 + f^2 &= (b-d)^2 + f^2 \\ &= f^2 + d^2 + b^2 - 2db \\ &= a^2 + b^2 - 2ab\cos\theta, \end{aligned}$$

which is (A.4.8).

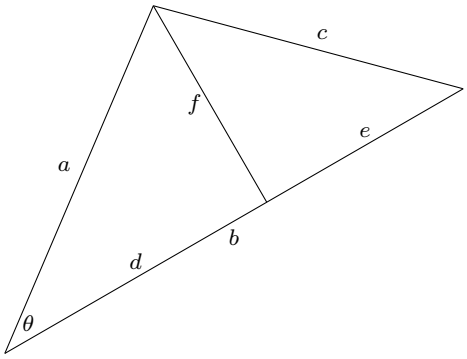


Fig. A.12 Proof of Pythagoras for general triangles.

Next, connect Figures A.10 and A.11 by noting $a = |v_2|$ and $b = |v_1|$ and $c = |v_2 - v_1|$. By (A.4.6),

$$c^2 = |v_2 - v_1|^2 = |v_2|^2 - 2v_1 \cdot v_2 + |v_1|^2 = a^2 + b^2 - 2(x_1x_2 + y_1y_2),$$

thus

$$c^2 = a^2 + b^2 - 2(x_1x_2 + y_1y_2). \tag{A.4.9}$$

Comparing the terms in (A.4.8) and (A.4.9), we arrive at the dot product identity (A.4.5).

If $P = (x, y)$, let $P^\perp = (-y, x)$ (pronounced "P-perp"), and let $v = OP$ and $v^\perp = OP'$ be the vectors emanating from the origin, and ending at P and $P^\perp$. Then

$$v \cdot v^\perp = (x, y) \cdot (-y, x) = 0.$$

This shows v and $v^\perp$ are perpendicular (Figure A.13).

From Figure A.13, we see points P and P' on the unit circle satisfy $P \cdot P' = 0$ iff $P' = \pm P^\perp$.

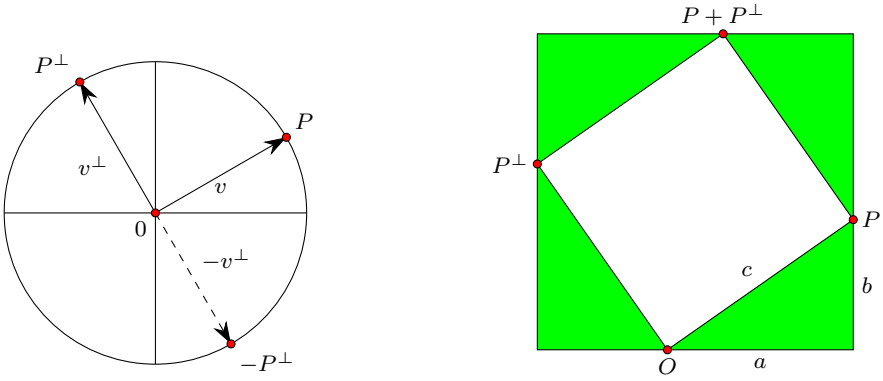


Fig. A.13 P and $P^\perp$ and v and $v^\perp$.

We now solve two linear equations in two unknowns x, y. We start with the homogeneous linear system

$$ax + by = 0, \qquad cx + dy = 0. \tag{A.4.10}$$

Let A be the 2×2 matrix

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{A.4.11}$$

Assume $(a, b) \neq (0, 0)$. Then it is easy to exhibit a nonzero solution of the first equation in (A.4.10): choose $(x, y) = (-b, a) = (a, b)^\perp$. If we want this to be a solution of the second equation as well, we must have $cx + dy = ad - bc = 0$.

On the other hand, if $(c, d) \neq (0, 0)$, $(x, y) = (-d, c) = (c, d)^\perp$ is a nonzero solution of the second equation in (A.4.10). If we want this to be a solution of the first equation as well, we must have $ax + by = bc - ad = 0$.

Based on this, we make the following definition. The *determinant* of A is

$$\det(A) = \det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc. \tag{A.4.12}$$

Above we found solutions of (A.4.10) when $\det(A) = 0$. We now show the only solution is $(x, y) = (0, 0)$ when $\det(A) \neq 0$.

Multiply the first equation in (A.4.10) by d and the second by b and subtract, obtaining

$$(ad - bc)x = d(ax + by) - b(cx + dy) = 0.$$

When $ad - bc \neq 0$, this leads to $x = 0$. Similarly, in (A.4.10), multiply the first equation by c and the second by a and subtract, obtaining

$$(bc - ad)y = c(ax + by) - a(cx + dy) = 0.$$

When $ad - bc \neq 0$, this leads to $y = 0$.

Summarizing, we conclude

Homogeneous Linear System

Let A be the matrix (A.4.11). There are three cases.

- If $\det(A) \neq 0$, the only solution of (A.4.10) is $(x, y) = (0, 0)$.
- If $\det(A) = 0$ but $A \neq 0$, every solution of (A.4.10) is a scalar multiple of $(x, y) = (-b, a)$, or of $(x, y) = (-d, c)$, depending on whether $(a, b) \neq (0, 0)$ or $(c, d) \neq (0, 0)$.
- If $A = 0$, any (x, y) is a solution of (A.4.10).

This covers the homogeneous case. For the inhomogeneous linear system

$$ax + by = e, \qquad cx + dy = f, \tag{A.4.13}$$

again there are the same three cases.

- If $A = 0$, the system (A.4.13) has a solution only if $(e, f) = (0, 0)$, in which case, any (x, y) is a solution.
- If $A \neq 0$ but $\det(A) = 0$, multiplying and subtracting as above, we obtain

$$\begin{aligned} (ad - bc)x &= d(ax + by) - b(cx + dy) = de - bf, \\ (ad - bc)y &= a(cx + dy) - c(ax + by) = af - ce. \end{aligned} \tag{A.4.14}$$

  This implies $ce = af$ and $de = bf$. Conversely, when $ce = af$ and $de = bf$,

$$(x, y) = (e/a, 0), \quad (x, y) = (0, e/b), \quad (x, y) = (f/c, 0), \quad (x, y) = (0, f/d)$$

  are solutions, when $a \neq 0$, $b \neq 0$, $c \neq 0$, or $d \neq 0$ respectively.
- If $\det(A) \neq 0$, dividing (A.4.14) by $ad - bc$ leads to

$$x = \frac{de - bf}{ad - bc}, \qquad y = \frac{af - ce}{ad - bc}. \tag{A.4.15}$$

Putting all this together, we conclude

Inhomogeneous Linear System

Let A be the matrix (A.4.11). There are three cases.

- If $\det(A) \neq 0$, (A.4.13) has the unique solution (A.4.15).
- If $\det(A) = 0$ but $A \neq 0$, (A.4.13) has a solution iff $ce = af$ and $de = bf$, in which case, there are four possible solutions, listed above, depending on which of a, b, c, d is nonzero.
- If $A = 0$, (A.4.13) has a solution if $(e, f) = (0, 0)$, in which case, any (x, y) is a solution.

All other solutions differ from these solutions by a solution of (A.4.10).

In §2.9, we will understand the three cases in terms of the rank of A equal to 2, 1, or 0.



The *trace* of A is the sum of the diagonal entries,

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \qquad \Longrightarrow \qquad \operatorname{trace}(A) = a + d. \tag{A.4.16}$$

The determinant and trace are the basic functions of 2×2 matrices.

We now go over the properties of 2×2 matrices. These we use in §1.4, and these properties are a prelude to Chapter 2.

We go over matrix-vector, matrix-matrix, and tensor products, and we connect them to the dot product. For the remainder of the section, matrices are 2×2 and vectors are in $\mathbf{R}^2$.

The matrix (A.4.11) can be written in terms of the two vectors $u = (a, b)$ and $v = (c, d)$, as follows

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix}, \qquad u = (a, b), v = (c, d).$$

In this case, we call u and v the *rows* of A. On the other hand, A may be written as

$$A = \begin{pmatrix} a & c \\ b & d \end{pmatrix} = \begin{pmatrix} u & v \end{pmatrix}, \qquad u = (a, b), \quad v = (c, d).$$

In this case, we call u and v the *columns* of A. Many texts then write u and v as

$$u = \begin{pmatrix} a \\ b \end{pmatrix}, \qquad v = \begin{pmatrix} c \\ d \end{pmatrix}. \tag{A.4.17}$$

We do not usually do this when u and v are on their own, because then they are just vectors. We only do this when u and v being multiplied with matrices or are rows or columns of matrices.

In fact, when we do write (A.4.17), we are thinking of u and v as 2×1 matrices, not as vectors. This shows there are at least three ways to think about a matrix: as rows, or as columns, or as a single block.

The simplest operations on matrices are addition and scaling. Addition is

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad A' = \begin{pmatrix} a' & b' \\ c' & d' \end{pmatrix} \qquad \Longrightarrow \qquad A + A' = \begin{pmatrix} a + a' & b + b' \\ c + c' & d + d' \end{pmatrix},$$

and scaling is

$$tA = \begin{pmatrix} ta & tb \\ tc & td \end{pmatrix}.$$

As with vectors, addition and scaling are *linear*

$$s(A + B) = sA + sB, \qquad (s + t)A = sA + tA.$$

The *transpose* A^t of the matrix A is

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \qquad \Longrightarrow \qquad A^t = \begin{pmatrix} a & c \\ b & d \end{pmatrix}.$$

Then the rows of A^t are the columns of A, and vice-versa.

If Q satisfies $Q^t = Q$, then Q is a *symmetric matrix*. In this case, the rows of Q equal the columns of Q, and Q looks like

$$Q = \begin{pmatrix} a & b \\ b & c \end{pmatrix}.$$

A general matrix A consists of four numbers a, b, c, d, and a symmetric matrix Q consists of three numbers a, b, c.



Let $x = (s, t)$ be a vector. We now explain how to multiply the matrix A by the vector x. The result is then another vector Ax. This is called *matrix-vector multiplication*.

To do this, we write A as rows $A = \begin{pmatrix} u \\ v \end{pmatrix}$, then use the dot product to define

$$Ax = (u \cdot x, v \cdot x) = (as + bt, cs + dt).$$

When multiplying this way, we often write

$$Ax = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} as + bt \\ cs + dt \end{pmatrix} = \begin{pmatrix} u \cdot x \\ v \cdot x \end{pmatrix},$$

and we think of x and Ax as column vectors.

This terminology is introduced to keep things consistent: *It's always row-times-column with row on the left and column on the right.* Nevertheless, a vector, a row vector, and a column vector are all the same thing, just a vector.

Matrix-vector multiplication and scaling are related by

$$A(sx) = s(Ax) = (sA)x$$

for any scalar s, and is *linear*

$$A(u + v) = Au + Av, \qquad (A + B)u = Au + Bu.$$

Let $A = (u, v)$ be a matrix with columns u and v, and let $x = (s, t)$ be a vector. To express Ax in terms of u, v, write

$$Ax = \begin{pmatrix} a & c \\ b & d \end{pmatrix} \begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} as + ct \\ bs + dt \end{pmatrix} = \begin{pmatrix} as \\ bs \end{pmatrix} + \begin{pmatrix} ct \\ dt \end{pmatrix} = su + tv.$$

Summarizing,

Matrix-Vector Product in Terms of Rows and Columns

Let A be a matrix and $x = (s, t)$ a vector. If A has rows u, v, then $Ax = (u \cdot x, v \cdot x)$. If A has columns u, v, then $Ax = su + tv$.

As a consequence, a matrix A may be identified by its matrix-vector products Ax over all vectors x,

Matrix-Vector Product Uniquely Identifies the Matrix

Let A and A' be matrices. If $Ax = A'x$ for every vector x, then $A = A'$.

To see this, let u, v and u', v' be the columns of A, A'. Inserting $x = (s, t) = (1, 0)$ into

$$su + tv = Ax = A'x = su' + tv'.$$

yields $u = u'$, and repeating with $x = (0, 1)$ yields $v = v'$.



Let x and y be vectors, with $x = (s, t)$, and let A be a matrix with rows u, v. Then A^t has columns u, v, so we have

$$Ay = (u \cdot y, v \cdot y), \qquad A^t x = su + tv.$$

Taking the dot product of Ay with x leads to

$$x \cdot Ay = su \cdot y + tv \cdot y.$$

Taking the dot product of $A^t x$ with y leads to

$$(A^t x) \cdot y = (su + tv) \cdot y = su \cdot y + tv \cdot y.$$

Since the last two displays are equal, we arrive at the

Dot Product Transpose Identity

Let A be a matrix and x, y vectors. Then

$$(A^t x) \cdot y = x \cdot Ay \qquad \text{and} \qquad (Ax) \cdot y = x \cdot (A^t y). \tag{A.4.18}$$

We established the first equation in (A.4.18). The second equation follows by switching x and y and using $a \cdot b = b \cdot a$.



Just like we can multiply matrices and vectors, we can also multiply two matrices A and A' and obtain a product AA'. This is *matrix-matrix multiplication*. Following the row-column rule above, we write $A = \begin{pmatrix} u \\ v \end{pmatrix}$ as rows and $A' = (u', v')$ as columns and define

$$AA' = (Au', Av') = \begin{pmatrix} u \cdot u' & u \cdot v' \\ v \cdot u' & v \cdot v' \end{pmatrix}.$$

If we do this the other way, we obtain

$$A'A = \begin{pmatrix} u' \cdot u & u' \cdot v \\ v' \cdot u & v' \cdot v \end{pmatrix},$$

so in general

$$AA' \neq A'A.$$

The matrix-vector product and the matrix-matrix product are related by the identity

$$(AA')x = A(A'x). \tag{A.4.19}$$

To check this, let A' have columns u', v', and let $x = (s, t)$. Then $A'x = su' + tv'$ so

$$A(A'x) = A(su' + tv') = sAu' + tAv' = (Au', Av')x = (AA')x.$$

We conclude *multiplying a vector by two matrices, one after the other, is the same as multiplying the vector by their matrix product.*

As a consequence, the matrix-matrix product is also *linear*

$$A(B+C) = AB + AC, \qquad (A+B)C = AC + BC.$$

We can use (A.4.19) to derive

Transpose of Matrix-Matrix Product

If A and B are matrices, then

$$(AB)^t = B^t A^t.$$

This follows from

$$(AB)x \cdot y = (A(Bx)) \cdot y = Bx \cdot A^t y = x \cdot B^t(A^t y) = x \cdot (B^t A^t)y,$$

valid for any x and y.



If $u = (a, b)$ and $v = (c, d)$ are vectors, their *tensor product* is the matrix

$$u \otimes v = \begin{pmatrix} ac & ad \\ bc & bd \end{pmatrix} = \begin{pmatrix} cu & du \end{pmatrix} = \begin{pmatrix} av \\ bv \end{pmatrix}.$$

Here we wrote $u \otimes v$ as a single block, and also in terms of rows and columns.

If we do this the other way, we get

$$v \otimes u = \begin{pmatrix} ca & cb \\ da & db \end{pmatrix},$$

so

$$(u \otimes v)^t = v \otimes u.$$

When $u = v$, $u \otimes v = v \otimes v$ is a symmetric matrix.

Just like for the other products, the tensor product is *linear*

$$(su + tv) \otimes w = su \otimes w + tv \otimes w.$$

Here is code for `tensor`.

```
def tensor(u,v): return array([ [ a*b for b in v] for a in u ])
```

There is no need to use this, since `numpy.outer` does the same job,

```
u = array([1,2])
v = array([3,4])

tensor(u,v), outer(u,v)
```

From the definition of $u \otimes v$, the following equations are immediate,

$$\det(u \otimes v) = 0, \qquad \operatorname{trace}(u \otimes v) = u \cdot v.$$

Here are the basic identities for the tensor product.

Tensor Product Identities

The following identities hold.

1. If u, v, w are vectors,
$$(u \otimes v)w = (v \cdot w)u. \tag{A.4.20}$$
2. If a, b, c, d are vectors,
$$a \cdot (b \otimes c)d = (a \cdot b)(c \cdot d). \tag{A.4.21}$$
3. If $Q = v \otimes v$,
$$u \cdot Qu = u \cdot (v \otimes v)u = (u \cdot v)^2. \tag{A.4.22}$$
4. If A has columns u, v,
$$AA^t = u \otimes u + v \otimes v. \tag{A.4.23}$$
5. For matrices A, B, and vectors u, v,
$$A(u \otimes v)B^t = (Au) \otimes (Bv). \tag{A.4.24}$$

For (A.4.20), suppose $u = (s, t)$. Then

$$(u \otimes v)w = \begin{pmatrix} sv \\ tv \end{pmatrix} w = (sv \cdot w, tv \cdot w) = (v \cdot w)u.$$

For (A.4.21), replace u, v, w in (A.4.20) by b, c, d, then take the dot product with a.

For (A.4.22), replace a, b, c, d in (A.4.21) by v, u, u, v.

For (A.4.23), let x be any vector. Since A^t has rows u, v, $A^t x = (u \cdot x, v \cdot x)$. Since A has columns u, v,

$$AA^t x = A(A^t x) = (u \cdot x)u + (v \cdot x)v = (u \otimes u + v \otimes v)x.$$

Since a matrix-vector product against all vectors uniquely identifies the matrix, this derives (A.4.23).

For (A.4.24), let x be any vector. Then

$$((u \otimes v)B^t)x = (u \otimes v)(B^t x) = (v \cdot B^t x)u = (Bv \cdot x)u,$$

so

$$(A(u \otimes v)B^t)x = A((u \otimes v)B^t)x) = A((Bv \cdot x)u) = (Bv \cdot x)Au = (Au \otimes Bv)x.$$



A *rotation* in the plane is the matrix

$$U = U(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}.$$

Here θ is the *angle of rotation*. By the trigonometric addition formulas (A.5.6),

$$\begin{aligned} U(\theta)U(\theta') &= \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\theta' & -\sin\theta' \\ \sin\theta' & \cos\theta' \end{pmatrix} \\ &= \begin{pmatrix} \cos(\theta+\theta') & -\sin(\theta+\theta') \\ \sin(\theta+\theta') & \cos(\theta+\theta') \end{pmatrix} = U(\theta+\theta'). \end{aligned}$$

This says rotating by θ' followed by rotating by θ is the same as rotating by $\theta + \theta'$.



There is a special matrix I, the identity matrix,

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

The matrix I satisfies

$$AI = IA = A$$

for any matrix A.

Also, for each matrix A with $\det(A) \neq 0$, the matrix

$$A^{-1} = \frac{1}{\det(A)} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} = \frac{1}{ad-bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

is the *inverse* of A. The inverse matrix satisfies

$$AA^{-1} = I = A^{-1}A.$$

Since

$$(B^{-1}A^{-1})(AB) = B^{-1}(A^{-1}A)B = B^{-1}IB = B^{-1}B = I,$$

we have

Inverse of Matrix-Matrix Product

If A and B are matrices, then

$$(AB)^{-1} = B^{-1}A^{-1}.$$



Using the matrix inverse property, the solution of

$$Ax = b$$

is

$$x = A^{-1}b,$$

since

$$Ax = AA^{-1}b = Ib = b.$$

With this, we can rewrite (A.4.13) as

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} e \\ f \end{pmatrix},$$

and the solution (A.4.15) can be rewritten

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} \begin{pmatrix} e \\ f \end{pmatrix} = \frac{1}{ad-bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \begin{pmatrix} e \\ f \end{pmatrix} = \frac{1}{ad-bc} \begin{pmatrix} de-bf \\ af-ce \end{pmatrix}.$$

We study inverse matrices in depth in §2.3.



Let A have columns u, v. Then A^t has rows u, v. Since matrix multiplication is *row* × *column*,

$$A^tA = \begin{pmatrix} u \\ v \end{pmatrix} (u\ v) = \begin{pmatrix} u \cdot u & u \cdot v \\ v \cdot u & v \cdot v \end{pmatrix}.$$

Now suppose $A^tA = I$. Then $u \cdot u = 1 = v \cdot v$ and $u \cdot v = 0$, so u and v are orthogonal unit vectors, or orthonormal vectors. We have shown

Orthogonal Matrices

Let A be a matrix. Then $A^tA = I$ iff the columns of A are orthonormal, and $AA^t = I$ iff the rows of A are orthonormal.

The second statement follows by applying the first to A^t instead of A. A matrix U is *orthogonal* if

$$U^tU = I = UU^t.$$

Thus a matrix is orthogonal iff its rows are orthonormal, and its columns are orthonormal. Later we see that, for a square matrix, either equation, $UU^t = I$ or $U^tU = I$, implies the other.

Exercises

Exercise A.4.1 Solve the linear system

$$ax + by = c, \qquad -bx + ay = d.$$

Exercise A.4.2 Let $u = (1, a)$, $v = (b, 2)$, and $w = (3, 4)$. Solve

$$u + 2v + 3w = 0$$

for a and b.

Exercise A.4.3 Let $u = (1, 2)$, $v = (3, 4)$, and $w = (5, 6)$. Find a and b such that

$$au + bv = w.$$

Exercise A.4.4 Let P be a nonzero point in the plane. What is $\left(P^{\perp}\right)^{\perp}$?

Exercise A.4.5 Let $A = \begin{pmatrix} 8 & -8 \\ -7 & -3 \end{pmatrix}$ and $B = \begin{pmatrix} 3 & -2 \\ 2 & -2 \end{pmatrix}$. Compute AB and BA.

Exercise A.4.6 Let $A = \begin{pmatrix} 9 & 2 \\ -36 & -8 \end{pmatrix}$. Find a nonzero 2×2 matrix B satisfying $AB = 0$.

Exercise A.4.7 Solve for X

$$\begin{pmatrix} -7 & 4 \\ 4 & -3 \end{pmatrix} - 4X = \begin{pmatrix} -9 & 5 \\ 6 & -9 \end{pmatrix}.$$

Exercise A.4.8 Use (A.4.20) to compute the matrix-matrix product $(u \otimes v)^2$.

Exercise A.4.9 Find a nonzero 2×2 matrix A satisfying $A^2 = 0$.

Exercise A.4.10 If V is a symmetric 2×2 matrix, show

$$(\text{trace}(V))^2 = 2\det(V) + \text{trace}(V^2).$$

Exercise A.4.11 With V a 2×2 matrix and t scalar, show

$$\det(I + tV) = 1 + t \cdot \text{trace}(V) + t^2 \det(V).$$

Exercise A.4.12 With Q and V 2×2 matrices, Q invertible, and t scalar, show

$$\frac{\det(Q + tV)}{\det(Q)} = 1 + t \cdot \text{trace}(Q^{-1}V) + t^2 \det(Q^{-1}V).$$

Exercise A.4.13 What is the trace of $A = \begin{pmatrix} 9 & 2 \\ -36 & -8 \end{pmatrix}$?

Exercise A.4.14 Let A be a 2×2 matrix with rows u, v. The *wedge product* of u and v is the matrix

$$u \wedge v = u \otimes v - v \otimes u.$$

Show

$$u \wedge v = \begin{pmatrix} 0 & \delta \\ -\delta & 0 \end{pmatrix},$$

where $\delta = \det(A)$.

Exercise A.4.15 Let u be a unit vector, and let $A = u \otimes u$. Compute A^{100}.

Exercise A.4.16 Calculate the areas of the triangles and the squares in Figure A.13. From that, deduce Pythagoras's theorem $c^2 = a^2 + b^2$.

Exercise A.4.17 Let u and v be unit vectors, and let $A = u \otimes v$. If $A^2 = 0$, what is the angle between u and v?

Exercise A.4.18 Let $W = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$. What is W^{211}?

A.5 Complex Numbers

In §A.4, we study points in two dimensions, and we saw how points can be added and subtracted. In §2.1, we study points in any number of dimensions, and there we also add and subtract points.

In two dimensions, each point has a shadow (Figure A.5). By stacking shadows, points in the plane can be multiplied and divided (Figure A.14). In

this sense, points in the plane behave like numbers, because they follow the usual rules of arithmetic.

This ability of points in the plane to follow the usual rules of arithmetic is, apart from the real line, unique to two dimensions, and not present in any other dimension. When thought of in this manner, points in the plane are called *complex numbers*, and the plane is the *complex plane*.



To define multiplication of points, let $P = (x, y)$ and $P' = (x', y')$ be points on the unit circle. Stack the shadow of P' on top of the shadow of P, as in Figure A.14. Because angle stacking is at the basis of angle measurement [13], we must do this without knowledge of angle measure.

Here is how one does this without any angle measurement: Mark $Q = x'P$ at distance x' along the vector OP joining O and P, and draw the circle with radius y' and center Q. Then this circle intersects the unit circle at two points, both called P''.

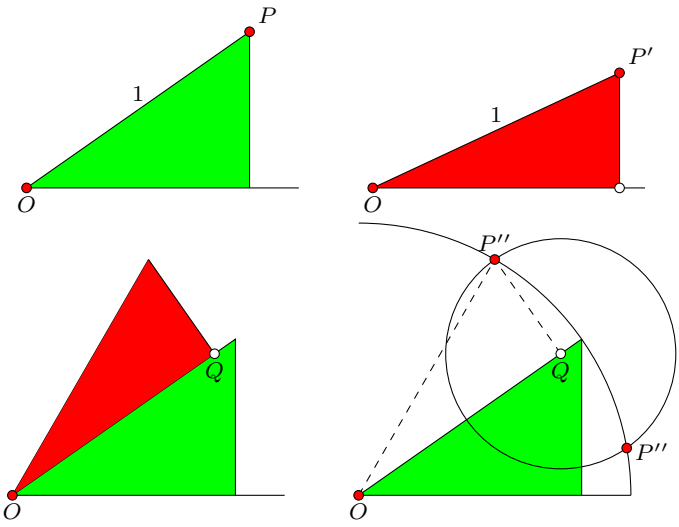


Fig. A.14 Multiplying and dividing points on the unit circle.

We think of the first point P'' as the result of multiplying P and P', and we write $P'' = PP'$, and we think of the second point P'' as the result of dividing P by P', and we write $P'' = P/P'$. Then we have

Multiplication and Division of Points

For $P = (x, y)$ and $P' = (x', y')$ on the unit circle, when $x'y' \neq 0$,

$$\begin{aligned} P'' &= PP' = (xx' - yy', x'y + xy'), \\ P'' &= P/P' = (xx' + yy', x'y - xy'). \end{aligned} \tag{A.5.1}$$

To derive (A.5.1), let $P^\perp = (-y, x)$ ("P-perp"). Then

$$\begin{aligned} x'P + y'P^\perp &= (x'x, x'y) + (-y'y, y'x) = (xx' - yy', x'y + xy'), \\ x'P - y'P^\perp &= (x'x, x'y) - (-y'y, y'x) = (xx' + yy', x'y - xy'), \end{aligned}$$

so (A.5.1) is equivalent to

$$P'' = x'P \pm y'P^\perp. \tag{A.5.2}$$

To establish (A.5.2), since P'' is on the circle of center Q and radius y', we may write $P'' = Q + y'R$, for some point R on the unit circle (see §A.4).

Interpreting points as vectors, and using (A.4.6), $P'' = x'P + y'R$ is on the unit circle iff

$$\begin{aligned} 1 = |x'P + y'R|^2 &= |x'P|^2 + 2x'P \cdot y'R + |y'R|^2 \\ &= x'^2|P|^2 + 2x'y'P \cdot R + y'^2|R|^2 \\ &= x'^2 + y'^2 + 2x'y'P \cdot R \\ &= 1 + 2x'y'P \cdot R. \end{aligned}$$

But this happens iff $P \cdot R = 0$, which happens iff $R = \pm P^\perp$ (Figure A.13). This establishes (A.5.2).

More generally, if P and P' are not on the unit circle, the formula for the product PP' remains unchanged, while the formula for the quotient P/P' must be modified.

To see how, let $r = |P|$ and $r' = |P'|$, and let R be any point satisfying $|R| = r$. Then

$$P'' = Q + y'R = x'P + y'R$$

satisfies $|P''| = rr'$ exactly when $R = \pm P^\perp$, leading to the two points in (A.5.1).

Let $\bar{P}$ be the *conjugate* $(x, -y)$ of $P = (x, y)$. The first P'' is the product

$$PP' = (xx' - yy', x'y + xy'), \tag{A.5.3}$$

and the second P'' is the *hermitian product* $P\bar{P}'$ of P and $\bar{P}'$.

The correct formula for division is given by

$$P/P' = \frac{1}{r'^2} P\bar{P}' = \frac{1}{x'^2 + y'^2}(xx' + yy', x'y - xy'), \qquad r' \neq 0. \tag{A.5.4}$$

When $r' = 1$, (A.5.4) reduces to the formula in (A.5.1).

With this understood, it is easily checked that division undoes multiplication,

$$(P/P')P' = P.$$

In fact, one can check that multiplication and division as defined by (A.5.3) and (A.5.4) follow the usual rules of arithmetic, for any two points P and P' in the plane, not necessarily on the unit circle.



It is natural to identify points on the horizontal axis with real numbers, because, using (A.5.1), $P = (x, 0)$ and $P' = (x', 0)$ implies

$$P+P' = (x,0)+(x',0) = (x+x',0), \qquad PP' = (xx'-00, x0+x'0) = (xx',0).$$

Because of this, we can write $P = x$ instead of $P = (x, 0)$, and we call the horizontal axis the *real axis*.

Similarly, let $i = (0, 1)$. Then the point i is on the vertical axis, and, using (A.5.1), one can check

$$iy = (0,1)(y,0) = (-0, y) = y^{\perp}$$

($y^{\perp}$ was defined in §A.4). Thus the vertical axis consists of all points of the form iy. These are called *imaginary numbers*, and the vertical axis is the *imaginary axis*.

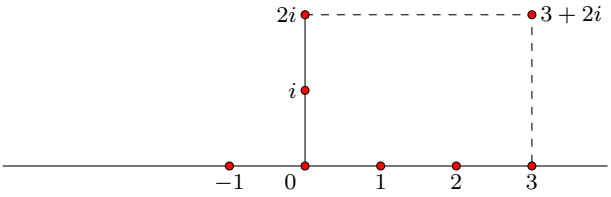


Fig. A.15 Complex numbers

Using i, any point $P = (x, y)$ may be written

$$P = x + iy,$$

since

$$x + iy = (x, 0) + (0, 1)(y, 0) = (x, 0) + (0, y) = (x, y).$$

This leads to Figure A.15.

When writing a complex number as $x + iy$, the *real part* is x, and the *imaginary part* is y (not iy). In this way, real numbers x are considered complex numbers with zero imaginary part, $x = x + i0$.

By (A.5.1), $i^2 = (0, 1)^2 = (-1, 0) = -1$. This shows

Square Root of -1

The complex number i satisfies $i^2 = -1$.

When thinking of points in the plane as complex numbers, it is traditional to denote them by z instead of P. By (A.5.1), we have

$$z = x + iy, \quad z' = x' + iy' \quad \Longrightarrow \quad zz' = (xx' - yy') + i(x'y + xy'),$$

and

$$\frac{z}{z'} = \frac{x + iy}{x' + iy'} = \frac{(xx' + yy') + i(x'y - xy')}{x'^2 + y'^2}.$$

In particular, one can always "move" the i from the denominator to the numerator by the formula

$$\frac{1}{z} = \frac{1}{x + iy} = \frac{x - iy}{x^2 + y^2} = \frac{\bar{z}}{|z|^2}.$$

Here $x^2 + y^2 = r^2 = |z|^2$ is the absolute value squared of z, and $\bar{z}$ is the conjugate of z.

Let r, θ be the polar coordinates of $P = (x, y)$ (Figure A.8). When we write $z = x + iy$, r is the *absolute value* of z, $r = |z| = \sqrt{x^2 + y^2}$, and

$$z = x + iy = r\cos\theta + ir\sin\theta = r(\cos\theta + i\sin\theta).$$

Let r, r', r'' and θ, θ', θ'' be the polar coordinates of z, z', $z'' = zz'$. Then Figure A.14 says $\theta'' = \theta + \theta'$. Using angle stacking together with his bisection method, Archimedes [14] defined angle measure θ, providing the theoretical basis for the addition formula $\theta'' = \theta + \theta'$.

By elementary algebra,

$$(x^2 + y^2)(x'^2 + y'^2) = (xx' - yy')^2 + (x'y + xy')^2. \tag{A.5.5}$$

Since this says $r^2 r'^2 = r''^2$, we conclude

Polar Coordinates of Complex Numbers

If (r, θ) and (r', θ') are the polar coordinates of complex numbers z and z', and (r'', θ'') are the polar coordinates of the product $z'' = zz'$, then

$$r'' = rr' \qquad \text{and} \qquad \theta'' = \theta + \theta'.$$

From this and (A.5.1), using $(x, y) = (\cos\theta, \sin\theta)$, $(x', y') = (\cos\theta', \sin\theta')$, we have the addition formulas

$$\begin{aligned} \sin(\theta + \theta') &= \sin\theta\cos\theta' + \cos\theta\sin\theta', \\ \cos(\theta + \theta') &= \cos\theta\cos\theta' - \sin\theta\sin\theta'. \end{aligned} \tag{A.5.6}$$

For example, if $\omega = \cos\theta + i\sin\theta$, then the polar coordinates of ω are $r = 1$ and θ. It follows the polar coordinates of ω^2 are $r = 1$ and 2θ, so $\omega^2 = \cos(2\theta) + i\sin(2\theta)$.

By the same logic, for any power k, the polar coordinates of ω^k are $r = 1$ and $k\theta$, so $\omega^k = \cos(k\theta) + i\sin(k\theta)$.

We can reverse the logic in the previous paragraph to compute square roots. We define the *square root* of a complex number z to be a complex number w satisfying $w^2 = z$. In this case, we write $w = \sqrt{z}$. If w is a square root, so is $-w$, so there are two square roots $w = \pm\sqrt{z}$.

The formula for $w = \sqrt{z}$ is

$$z = x + yi \qquad \Longrightarrow \qquad \sqrt{z} = \frac{r + x}{\sqrt{2r + 2x}} + \frac{yi}{\sqrt{2r + 2x}}. \tag{A.5.7}$$

Here $r = \sqrt{x^2 + y^2}$ and this formula is valid as long as z is not a negative number or zero. When z is a negative number or zero, $z = -x$ with $x \geq 0$. in this case, we have $\sqrt{z} = i\sqrt{x}$. Therefore *every complex number, other than zero, has two square roots.*

When z is on the unit circle, $r = 1$, (A.5.7) reduces to

$$\sqrt{z} = \frac{1 + x}{\sqrt{2 + 2x}} + \frac{yi}{\sqrt{2 + 2x}}.$$

We will need the roots of unity in §3.2. This generalizes square roots, cube roots, etc.

A complex number ω is a *root of unity* if $\omega^d = 1$ for some power d. If d is the power, we say ω is a d-th root of unity.

For example, the square roots of unity are ± 1, since $(\pm 1)^2 = 1$. Here we have

$$1 = \cos 0 + i \sin 0, \qquad -1 = \cos \pi + i \sin \pi.$$

The fourth roots of unity are ± 1 and $\pm i$, since $(\pm 1)^4 = 1$ and $(\pm i)^4 = 1$. Here we have

$$\begin{aligned} 1 &= \cos 0 + i \sin 0, \\ i &= \cos(\pi/2) + i \sin(\pi/2), \\ -1 &= \cos \pi + i \sin \pi, \\ -i &= \cos(3\pi/2) + i \sin(3\pi/2). \end{aligned}$$

In general, the roots of unity are denoted by powers of ω, so the square roots of unity are 1 and $\omega = -1$, and the fourth roots of unity are 1, $\omega = i$, $\omega^2 = -1$, $\omega^3 = -i$.

Let $\omega = \cos\theta + i \sin\theta$. Since $1 = \cos(2\pi) + i \sin(2\pi)$ and $\omega^k = \cos(k\theta) + i \sin(k\theta)$, a d-th root of unity ω satisfies

$$\omega = \cos(2\pi/d) + i \sin(2\pi/d). \tag{A.5.8}$$

If $\omega^d = 1$, then

$$\left(\omega^k\right)^d = \left(\omega^d\right)^k = 1^k = 1.$$

With ω given by (A.5.8), this implies

$$1, \omega, \omega^2, \ldots, \omega^{d-1}$$

are the d-th roots of unity.

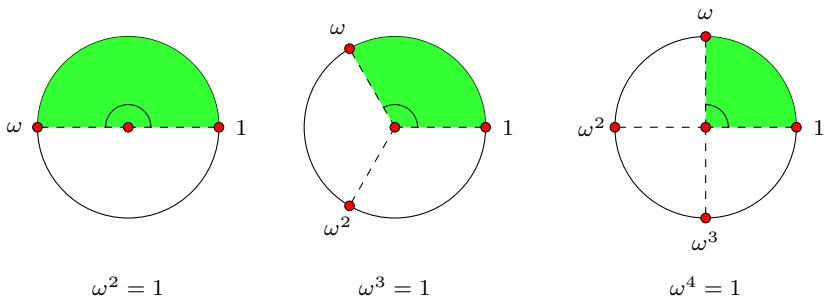


Fig. A.16 The second, third, and fourth roots of unity

If we set

$$\omega = -\frac{1}{2} + i\frac{\sqrt{3}}{2} = \cos(2\pi/3) + i \sin(2\pi/3),$$

then a calculation shows

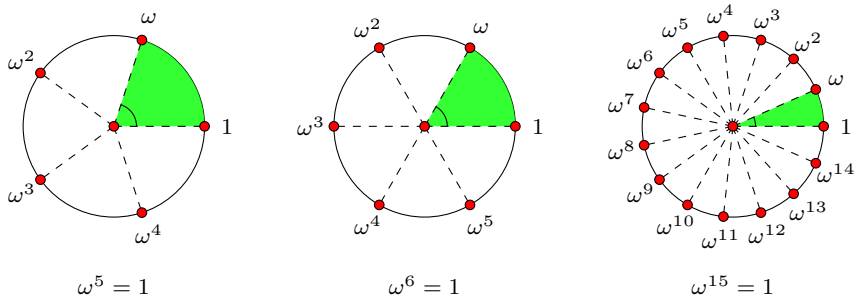
$$1, \qquad \omega, \qquad \omega^2 = -\frac{1}{2} - i\frac{\sqrt{3}}{2}$$

are the cube roots of unity,

$$1^3 = 1, \qquad \omega^3 = 1, \qquad (\omega^2)^3 = 1.$$

Similarly, the fifth roots of unity are 1, ω, ω^2, ω^3, ω^4, where

$$\omega = -\frac{1}{4} + \frac{\sqrt{5}}{4} + i\sqrt{\frac{\sqrt{5}}{8} + \frac{5}{8}} = \cos(2\pi/5) + i\sin(2\pi/5).$$



$\omega^5 = 1$ $\omega^6 = 1$ $\omega^{15} = 1$

Fig. A.17 The fifth, sixth, and fifteenth roots of unity

Summarizing,

Roots of Unity

Let $d \geq 1$ and let

$$\omega = \cos(2\pi/d) + i\sin(2\pi/d),$$

Then the d-th roots of unity are

$$1, \omega, \omega^2, \ldots, \omega^{d-1}.$$

The roots satisfy

$$\omega^k = \cos(2\pi k/d) + i\sin(2\pi k/d), \qquad k = 0, 1, 2, \ldots, d-1.$$

Here we write ω instead of ω_d, we do not indicate the dependence of ω on d.

Since $\omega^d = 1$, one has, from Figures A.16 and A.17,

$$\omega^k + \omega^{-k} = \omega^k + \omega^{d-k} = 2\cos(2\pi k/d), \qquad k = 0, 1, 2, \ldots, d-1. \quad \text{(A.5.9)}$$

This we need in §3.2.



A *polynomial* is an expression of the form

$$p(z) = z^d + c_1 z^{d-1} + c_2 z^{d-2} + \cdots + c_d.$$

For example, $p(z) = z^3 - 5z + 2$ or $p(z) = z^2 - 2z + 2$. Here z is the *variable*, and the constants c_1, c_2, …, c_d are the *coefficients*.

A *root* of a polynomial $p(z)$ is a complex number a satisfying $p(a) = 0$. For example, the roots of $z^2 - 2z + 2$ are $1 \pm i$, and the roots of $z^5 - 1$ are the fifth roots of unity 1, ω, ω^2, ω^3, ω^4. In general, the roots of $z^d - 1$ are the d-th roots of unity 1, ω, ω^2, …, ω^{d-1}.

The *fundamental theorem of algebra* states that every polynomial has as many roots as its degree: If the degree of $p(z)$ is d, there are d (not necessarily distinct) roots a_1, a_2, …, a_d of $p(z)$, and $p(z)$ may be factored into a product

$$p(z) = \prod_{k=1}^{d} (z - a_k) = (z - a_1)(z - a_2)\ldots(z - a_d). \quad \text{(A.5.10)}$$

Conversely, given a_1, a_2, …, a_d, (A.5.10) exhibits a polynomial with these as roots.

In particular, when $p(z) = z^d - 1$, we have

$$\frac{z^d - 1}{z - 1} = \prod_{k=1}^{d-1} (z - \omega^k). \quad \text{(A.5.11)}$$



Here is `sympy` code for the roots of unity.

```
from sympy import solve, symbols

z = symbols('z')

d = 5
solve(z**d - 1)
```

In `numpy`, the roots of $p(z) = az^2 + bz + c$ are returned by

```
from numpy import roots

# roots([a,b,c])
```

Since the cube roots of unity are the roots of $p(z) = z^3 - 1$, the code

```
from numpy import *

# cube roots of unity

roots([1,0,0,-1])
```

returns the cube roots

```
array([-0.5+0.8660254j, -0.5-0.8660254j, 1. +0.j ])
```

Exercises

Exercise A.5.1 Let $P = (1, 2)$ and $Q = (3, 4)$ and $R = (5, 6)$. Calculate PQ, P/Q, PR, P/R, QR, Q/R.

Exercise A.5.2 Let $a = 1 + 2i$ and $b = 3 + 4i$ and $c = 5 + 6i$. Calculate ab, a/b, ac, a/c, bc, b/c.

Exercise A.5.3 We say z' is the *reciprocal* of z if $zz' = 1$. Show the reciprocal of $z = x + yi$ is

$$z' = \frac{x - yi}{x^2 + y^2}.$$

Exercise A.5.4 Show $\sqrt{z}$ given by (A.5.7) satisfies $(\sqrt{z})^2 = z$.

Exercise A.5.5 Check (A.5.5) is correct.

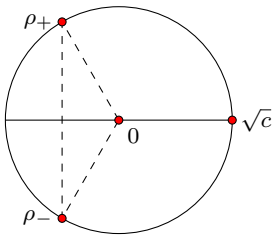


Fig. A.18 Complex conjugate roots $\rho_\pm$.

Exercise A.5.6 Let a, b, c be complex numbers, with $a \neq 0$. Show the roots of $p(z) = az^2 + bz + c$ are given by the Babylonian quadratic formula

$$z = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Exercise A.5.7 Let b, c be real numbers, with $b^2 < c$. If $\rho = x + yi$ satisfies $\rho(\rho + 2b) + c = 0$, show $|\rho| = \sqrt{c}$ (Figure A.18).

Exercise A.5.8 Let 1, ω, $\ldots$, ω^{d-1} be the d-th roots of unity. Using the code below, compute the *product*

$$(1 - \omega)(1 - \omega^2)(1 - \omega^3)\ldots(1 - \omega^{d-1}).$$

What is the answer? Try different degrees d.

```
from sympy import prod, solve, symbols, simplify

z = symbols('z')

d = 5
roots = solve(z**d - 1)

prod([ 1-a if a != 1 else 1 for a in roots ]).simplify()
```

The answer can be derived algebraically by using (A.5.11).

Exercise A.5.9 Given three distinct (non-equal) numbers a, b, c, there is a quadratic $p(z) = r+sz+tz^2$ satisfying $p(a) = 0$, $p(b) = 0$, and $p(c) = 1$. (With $d = 2$ and roots a and b, divide (A.5.10) by a constant to make $p(c) = 1$.)

Exercise A.5.10 Given three distinct (non-equal) numbers a, b, c, and three numbers α, β, γ, there is a quadratic $p(z) = r + sz + tz^2$ satisfying $p(a) = \alpha$, $p(b) = \beta$, and $p(c) = \gamma$. Repeat the previous exercise three times, once each for roots a and b, then b and c, then c and a, and take a linear combination of the results.

A.6 Integration

This section is a review of integration, using the fundamental theorem of calculus and Python.

Let $y = f(x)$ be a function, and let its graph be as in Figure A.19. The *integral*

$$I = \int_a^b f(x)\,dx \tag{A.6.1}$$

is the area under the graph between the vertical lines at a and b.

To repeat, the integral is a *number*, the area of a specific region under the graph $y = f(x)$. In Figure A.19, the integral (A.6.1) is the sum of three areas: red, green, blue.

We use Figure A.19 to derive the

Fundamental Theorem of Calculus (FTC)

If $F'(x) = f(x)$, then

$$\int_a^b f(x)\,dx = F(b) - F(a). \tag{A.6.2}$$

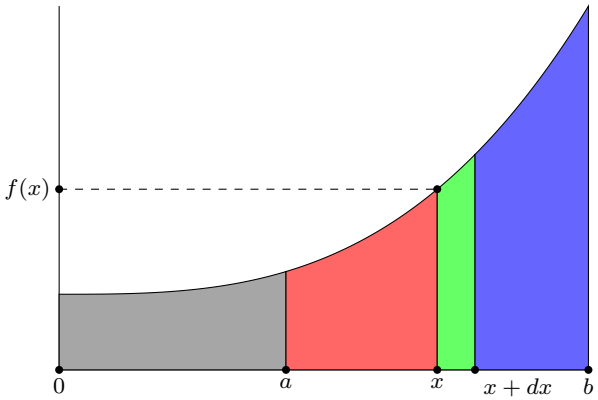


Fig. A.19 Areas under the graph.

To derive this, let $A(x)$ denote the area under the graph between the y-axis and the vertical line at x. Then $A(x)$ is the sum of the gray area and the red area, $A(a)$ is the gray area, and $A(b)$ is the sum of four areas: gray, red, green, and blue. It follows the integral (A.6.1) equals $A(b) - A(a)$.

Since $A(x + dx)$ is the sum of three areas, gray, red, green, it follows $A(x + dx) - A(x)$ is the green area. But the green area is approximately a rectangle of width dx and height $f(x)$. Hence the green area is approximately $f(x) \times dx$, or

$$A(x + dx) - A(x) \approx f(x)\,dx.$$

As a consequence of this analysis,

$$\frac{A(x + dx) - A(x)}{dx} \approx f(x).$$

The smaller dx is, the closer the green area is to a rectangle. Taking the limit $dx \to 0$, the green rectangle becomes infinitely thin, and we obtain

$$A'(x) = \lim_{dx \to 0} \frac{A(x+dx) - A(x)}{dx} = f(x).$$

Now let $F(x)$ be any function satisfying $F'(x) = f(x)$. Then $A(x)$ and $F(x)$ have the same derivative, so $A(x) - F(x)$ has derivative zero. By (4.1.2), $A(x) - F(x)$ is a constant C, or $A(x) = F(x) + C$. This implies

$$\int_a^b f(x)\,dx = A(b) - A(a) = (F(b) + C) - (F(a) + C) = F(b) - F(a).$$

This completes the derivation of the fundamental theorem of calculus.

Often one writes $F(x)|_a^b$ for $F(b) - F(a)$. The FTC then reads

$$\int_a^b f(x)\,dx = F(x)\Big|_{x=a}^{x=b}.$$

When $F'(x) = f(x)$, $F(x)$ is called an *anti-derivative* or *indefinite integral* of $f(x)$. This should not be confused with the integral (A.6.1), which is a number, an area.

Since the total area between a and b may be *sliced* into many thin green rectangles, interpreting the symbol $\int$ as "sum" explains the notation (A.6.1).

Important consequences of the FTC are integral *additivity*,

$$\int_a^b (f(x) + g(x))\,dx = \int_a^b f(x)\,dx + \int_a^b g(x)\,dx,$$

and integral *scaling*,

$$\int_a^b cf(x)\,dx = c\int_a^b f(x)\,dx.$$

For example, if $f(x) = x^d$, then, by (4.1.4), $F(x) = x^{d+1}/(d+1)$ satisfies $F'(x) = f(x)$, so, by the FTC,

$$\int_a^b x^n\,dx = F(b) - F(a) = \frac{b^{d+1}}{d+1} - \frac{a^{d+1}}{d+1}.$$

When $d = 2$, $a = -1$, $b = 1$, this is $2/3$, which is the area under the parabola in Figure A.20.

When $a = 0$, $b = 1$,

$$\int_0^1 t^d \, dt = \frac{1}{d+1}. \tag{A.6.3}$$

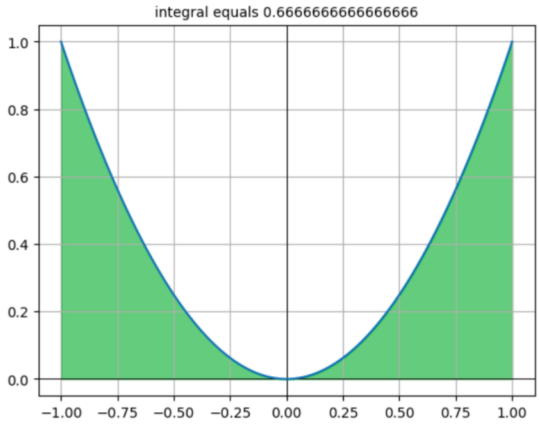


Fig. A.20 Area under the parabola.

When $F(x)$ can't be found, we can't use the FTC. Instead we use Python to evaluate the integral (A.6.1) as follows.

```
from scipy.integrate import quad

d = 2

def f(x): return x**d

a,b = -1, 1

# integral of f(x) over the interval [a,b]
# plus error
quad(f,a,b)
```

This not only returns the computed integral I but also an estimate of the error between the computed integral and the theoretical value,

```
(0.6666666666666666, 7.401486830834376e-15).
```

`quad` refers to *quadrature*, which is another term for integration.

Another example is the area under one hump of the sine curve in Figure A.21,

$$\int_0^{\pi} \sin x \, dx = -\cos \pi - (-\cos 0) = -(-1) + 1 = 2.$$

Here $f(x) = \sin x$, $F(x) = -\cos x$, $F'(x) = f(x)$. The Python code `quad` returns `(2.0, 2.220446049250313e-14)`.

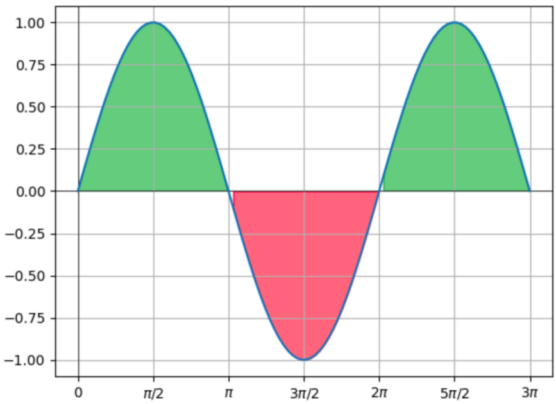


Fig. A.21 The graph and area under $\sin x$.

It is important to realize the integral (A.6.1) is the *signed* area under the graph: Portions of areas that are below the x-axis are counted negatively. For example,

$$\int_0^{2\pi} \sin x \, dx = -\cos(2\pi) - (-\cos 0) = -1 + 1 = 0.$$

Explicitly,

$$\int_0^{2\pi} \sin x \, dx = \int_0^{\pi} \sin x \, dx + \int_{\pi}^{2\pi} \sin x \, dx = 2 - 2 = 0,$$

so the areas under the first two humps in Figure A.21 cancel.

First we import the code from §4.1 (see Appendix B)

```
%%capture

import import_ipynb
from single_variable_calculus import *
```

then we run code for Figures A.20, A.21, A.22,

```
from numpy import *
set_printoptions(legacy = "1.25")

def plot_and_integrate(f,a,b,pi_ticks = False):
   # draw x-axis and y-axis
   axhline(0, color = 'black', lw = 1)
   axvline(0, color = 'black', lw = 1)
   # set x-axis ticks as multiples of pi/2
   if pi_ticks: set_pi_ticks(a,b)
   x = linspace(a,b,100)
   plot(x,f(x))
   positive = f(x) >= 0
   negative = f(x) < 0
   fill_between(x, f(x), 0, color = 'g', where = positive, alpha
    ↪ = .5)
   fill_between(x, f(x), 0, color = 'r', where = negative, alpha
    ↪ = .5)
   # just the integral, no error estimate
   I = quad(f, a, b, limit=1000)[0]
   title("integral equals " + str(I),fontsize = 10)
   grid()
   show()

def f(x): return sin(x)/x
a, b = 0.001, 3*pi

plot_and_integrate(f,a,b,pi_ticks = True)
```

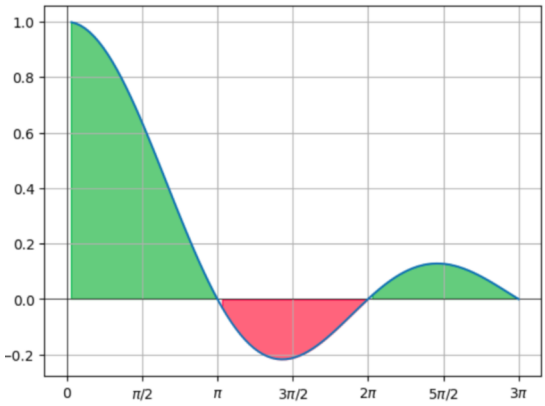


Fig. A.22 Integral of $\sin x/x$.

Above, the Python function `set_pi_ticks(a,b)` sets the x-axis tick mark labels at the multiples of $\pi/2$ The code for `set_pi_ticks` is in §4.1.

The exercises are meant to be done using the code in this section. For the infinite limits below, use `numpy.inf`.

Exercises

Exercise A.6.1 Plot and integrate $f(x) = x^2 + A\sin(5x)$ over the interval $[-10, 10]$, for amplitudes $A = 0, 1, 2, 4, 15$. Note the integral doesn't depend on A. Why?

Exercise A.6.2 Plot and integrate (Figure A.22)

$$\int_0^{3\pi} \frac{\sin x}{x}\, dx.$$

Exercise A.6.3 Plot and integrate $f(x) = \exp(-x)$ over $[a, b]$ with $a = 0$, $b = 1, 10, 100, 1000, 10000$.

Exercise A.6.4 Plot and integrate $f(x) = \sqrt{1 - x^2}$ over $[-1, 1]$.

Exercise A.6.5 Plot and integrate $f(x) = 1/\sqrt{1 - x^2}$ over $[-1, 1]$.

Exercise A.6.6 Plot and integrate $f(x) = (-\log x)^n$ over $[0, 1]$ for $n = 2, 3, 4$. What is the answer for general n?

Exercise A.6.7 With $k = 7$, $n = 10$, plot and integrate using Python

$$\int_0^1 x^k (1 - x)^{n-k}\, dx.$$

From (5.2.8), what is the exact integral?

Exercise A.6.8 Plot and integrate $f(x) = \sin(nx)/x$ over $[0, \pi]$ with $n = 1, 2, 3, 4, \ldots$. What's the limit of the integral as $n \to \infty$?

Exercise A.6.9 Use `numpy.inf` to compute

$$\frac{2}{\pi} \int_0^{\infty} \frac{\sin x}{x}\, dx.$$

Exercise A.6.10 Use `numpy.inf` to plot the normal `pdf` and compute its integral

$$\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-x^2/2}\, dx.$$

Exercise A.6.11 Let $\sigma(x) = 1/(1 + e^{-x})$. Plot and integrate $f(x) = \sigma(x)(1 - \sigma(x))$ over $[-10, 10]$. What is the answer for $(-\infty, \infty)$?

Exercise A.6.12 Let $P_n(x)$ be the Legendre polynomial of degree n (§4.1). Use `num_legendre` (§4.1) to compute the integral

$$\int_{-1}^{1} P_n(x)^2 \, dx$$

for $n = 1, 2, 3, 4$. What is the integral for general n? Hint – take the reciprocal of the answers.

A.7 Asymptotics and Convergence

Let x_1, x_2, ... be a sequence of scalars. What does it mean to say x_n is asymptotically zero? It means for n sufficiently large, x_n is arbitrarily small. To make this precise, we introduce some terminology.

We say x_n is *bounded* if all terms lie in some bounded interval, $a \le x_n \le b$. If $b > 0$, we say x_n is *bounded by* b if $|x_n| \le b$. For example $x_n = \sin(n)$ is bounded by $b = 1$. The constant b is a *bound*.

We say x_n is *eventually bounded* by a positive constant b, if, after ignoring finitely many terms, the remaining terms are bounded by b. For example, $x_n = 1/n$ is eventually bounded by $b = .01$, since, after ignoring the first ninety-nine terms, the sequence is bounded by b. The sequence $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, ..., is eventually bounded by 1, but not eventually bounded by 0.5.

If x_n is bounded by 5, then x_n is eventually bounded by 5. On the other hand, if x_n is eventually bounded by 5, then x_n is bounded, but not necessarily by 5, since we are ignoring finitely many terms, when writing the bound.

Typically we use the greek letter epsilon ϵ to denote small positive numbers.

Asymptotic Vanishing

If for any positive constant ϵ, no matter how small, a sequence x_n is eventually bounded by ϵ, we say x_n is *asymptotically zero* or *asymptotically vanishing*, and we write $x_n \approx 0$.

For example, $x_n = 1, 1, 1, \ldots$ is not asymptotically zero, since x_n is not eventually bounded by $\epsilon = 0.5$.

On the other hand, $x_n = 1/n$ is asymptotically zero: To bound the sequence by $\epsilon = 1/10$, we ignore the first nine terms. To bound the sequence by $\epsilon = 1/100$, we ignore the first ninety-nine terms. To bound the sequence by $\epsilon = 1/10000$, we ignore the first 9999 terms. Notice the smaller the desired bound, the more terms need to be ignored.

Immediate consequences of the asymptotic vanishing definition are the following properties.

1. $|x_n| \le e_n$ and $e_n \approx 0$ imply $x_n \approx 0$.
2. $x_n \approx 0$ and $y_n \approx 0$ implies $x_n + y_n \approx 0$,
3. $x_n \approx 0$ and y_n eventually bounded implies $x_n y_n \approx 0$.

These are intuitively clear. As a special case, for any constant c,

$$x_n \approx 0 \qquad \Longrightarrow \qquad cx_n \approx 0.$$

A sequence x_n is *asymptotically positive* if, apart from finitely many terms, x_n is positive. More generally, x_n is *asymptotically nonzero* if, apart from finitely many terms, x_n is not zero.

We say x_n is *asymptotically one*, if the *difference* $x_n - 1$ is asymptotically zero. In this case, we write

$$x_n \approx 1.$$

As a consequence of the above properties, we show

Convergence of Reciprocals

If $x_n \approx 1$, then $1/x_n \approx 1$.

To derive this, assume $x_n \approx 1$. Then $x_n - 1 \approx 0$, so $x_n - 1$ is eventually bounded by any positive constant. In particular, $x_n - 1$ is eventually bounded by $\epsilon = 1/2$, which means x_n is eventually in the interval $(1/2, 3/2)$, so eventually $x_n \ge 1/2$, or $1/x_n$ is eventually bounded. By property 3,

$$\frac{1}{x_n} - 1 = \frac{1}{x_n}(1 - x_n) \approx 0,$$

yielding the result.

The exercises exhibit many other consequences of the above properties. The point of the exercises is that they depend only on these properties, or their consequences.

Let $y_1, y_2, \ldots$ be another sequence. We say x_n is *asymptotically equal* to y_n, and we write

$$x_n \approx y_n, \tag{A.7.1}$$

if y_n is asymptotically nonzero, and the *ratio* x_n/y_n is asymptotically one, or $x_n/y_n \approx 1$. As part of this, we are assuming y_n is asymptotically nonzero, to ensure we aren't dividing by zero.

From the definition, it is not clear that $x_n \approx y_n$ is equivalent to $y_n \approx x_n$. Nevertheless, this is correct (Exercise **A.7.10**). Summarizing,

Asymptotic Equality

$$a_n \approx b_n \iff \frac{a_n}{b_n} \approx 1 \iff \frac{a_n}{b_n} - 1 \approx 0.$$

For example, let $a_n = n$, $b_n = n + 1$, $c_n = n^2$. When $n = 1000$,

$$a_n = 1000, \qquad b_n = 1001, \qquad \frac{a_n}{b_n} = .9990000,$$

so here we do have $a_n \approx b_n$ for n large. On the hand, here c_n is a million, and a_n is a thousand, so we don't have $a_n \approx c_n$.

This is exactly what is meant in (A.1.7). While both sides in (A.1.7) increase without bound, their ratio is close to one, for large n.

In general, $a_n \approx b_n$ is not the same as $a_n - b_n \approx 0$: ratios and differences behave differently. For example, based on (A.1.7), the following code

```
from numpy import *
set_printoptions(legacy = "1.25")


def factorial(n):
   if n == 1: return 1
   else: return n * factorial(n-1)

def stirling(n): return sqrt(2*pi*n) * (n/e)**n

a = factorial(100)
b = stirling(100)

a/b, a-b
```

returns

$$(1.000833677872004, 7.773919124995513 \times 10^{154}).$$

The first entry is close to one, but the second entry is far from zero.

If, however, $b_n \approx b$ for some nonzero constant b, then (Exercise **A.7.15**) ratios and differences are the same,

$$a_n \approx b_n \iff a_n - b_n \approx 0. \tag{A.7.2}$$

In particular, for $a \neq 0$, $a_n \approx a$ is the same as $a_n - a \approx 0$.



When we have $a_n - a \approx 0$, we say a is the *limit* of a_n, and we write

$$a = \lim_{n \to \infty} a_n. \tag{A.7.3}$$

As we saw above, limits and asymptotic equality are the same, as long as the limit is not zero. When a is the limit of a_n, we also say a_n *converges to* a, or a_n *approaches* a and we write $a_n \to a$.

With this notation, asymptotic vanishing is $a_n \to 0$, asymptotically one is $a_n \to 1$, and asymptotic equality is $a_n/b_n \to 1$.

Limits can be taken for sequences of points in $\mathbf{R}^d$ as well. Let a_n be a sequence of points in $\mathbf{R}^d$. We say a_n *converges to* a if $a_n \cdot v$ converges to $a \cdot v$ for every vector v. Here we also write $a_n \to a$ and we write (A.7.3).



In Chapter 6, $\approx$ is used for random variables. We say random variables X_n are asymptotically equal to random variables Y_n, and we write $X_n \approx Y_n$, if their corresponding probabilities are asymptotically equal,

$$Prob(a < X_n < b) \approx Prob(a < Y_n < b),$$

for any interval (a, b).

In particular, when Y does not depend on n, the asymptotic equality $X_n \approx Y$ is short for

$$Prob(a < X_n < b) \approx Prob(a < Y < b). \tag{A.7.4}$$

When Y is normal or standard normal or chi-squared, we also write $X_n \approx$ normal or $X_n \approx N(0,1)$ or $X_n \approx \chi^2_d$.

Since probabilities are positive, here both interpretations in (A.7.2) hold, hence we also have

$$Prob(a < X_n < b) \to Prob(a < Y < b).$$

Also, $X_n \approx Y$ in the sense of (A.7.4) is equivalent to approximations of the means

$$E(f(X_n)) \to E(f(Y)),$$

and equivalent to approximations of the moment-generating functions

$$M_{X_n}(t) \to M_Y(t).$$

Exercises

Exercise A.7.1 Let k be fixed. Show

$$\frac{\binom{n}{k}}{\binom{n}{k-1} + \binom{n}{k+1}} \approx 0 \qquad \text{and} \qquad 2^{-n}\binom{n}{k} \approx 0, \qquad n \to \infty.$$

(Use (A.2.9) for the second part.)

Exercise A.7.2 Show the infinite sum

$$\sum_{n=1}^{\infty} \frac{1}{n(n+1)} = \frac{1}{1\cdot 2} + \frac{1}{2\cdot 3} + \frac{1}{3\cdot 4} + \frac{1}{4\cdot 5} + \dots$$

equals 1. (If $t_n = 1/n$, the n-th term is $t_n - t_{n+1}$. Write out the sum term-by-term.)

Exercise A.7.3 Show the infinite sum

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \dots$$

is less than 2. In fact, as was first shown by Euler, this infinite sum equals $\pi^2/6$ [15]. (Compare with the infinite sum in the previous exercise.)

Exercise A.7.4 Let α and β be constants, with $\alpha > 0$ and β nonnegative. Show the infinite sum

$$\sum_{n=1}^{\infty} \frac{\alpha}{(n+\beta)^2} = \frac{\alpha}{(1+\beta)^2} + \frac{\alpha}{(2+\beta)^2} + \frac{\alpha}{(3+\beta)^2} + \frac{\alpha}{(4+\beta)^2} + \dots$$

is less than 2α, by repeating the argument in the previous exercise.

Exercise A.7.5 Show the infinite sum

$$\sum_{n=1}^{\infty} \frac{1}{n} = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \dots$$

is infinite, by grouping the terms into mini-batches, where the n-th mini-batch b_n is the sum of the 2^{n-1} terms ending with the term $1/2^n$. For example, for $n = 1, 2, 3$,

$$b_1 = \frac{1}{2}, \qquad b_2 = \frac{1}{3} + \frac{1}{4}, \qquad b_3 = \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8}.$$

Verify each mini-batch b_n is not less than $1/2$, for every $n = 1, 2, 3, 4, 5, \dots$. Use this to conclude this infinite sum equals ∞.

Exercise A.7.6 Let α and β be constants, with $\alpha > 0$ and β nonnegative. Show the infinite sum

$$\sum_{n=1}^{\infty} \frac{\alpha}{n+\beta} = \frac{\alpha}{1+\beta} + \frac{\alpha}{2+\beta} + \frac{\alpha}{3+\beta} + \frac{\alpha}{4+\beta} + \dots$$

is infinite, by repeating the argument in the previous exercise.

Exercise A.7.7 Let $m > 0$ and $L \geq 0$. Given scalars t_n converging to zero, let p_n be the product

$$p_n = \prod_{k=1}^{n} (1 - 2mt_k + L^2 t_k^2), \qquad n = 1, 2, \ldots$$

For example, if $L = 0$,

$$p_n = (1 - 2mt_1)(1 - 2mt_2) \ldots (1 - 2mt_n), \qquad n = 1, 2, \ldots$$

Suppose

$$t_1 + t_2 + t_3 + \cdots = \infty, \qquad t_1^2 + t_2^2 + t_3^2 + \cdots < \infty.$$

Then $t_n \to 0$, so the factors in p_n are eventually all positive, which implies p_n/p_N is positive for N large enough and all $n \geq N$. Use (A.3.16) to show p_n converges to zero as $n \to \infty$.

Exercise A.7.8 Let α and β be constants, with $\alpha > 0$ and β nonnegative. Show

$$t_n = \frac{\alpha}{n + \beta}, \qquad n = 1, 2, \ldots$$

satisfies

$$t_n t_{n+1} = \alpha(t_n - t_{n+1}), \qquad n = 1, 2, \ldots$$

Exercise A.7.9 Let α and β be constants, with $\alpha > 0$ and β nonnegative, and let

$$t_n = \frac{\alpha}{n + \beta}, \qquad n = 1, 2, \ldots$$

Use the previous exercise to show recursively

$$\prod_{k=1}^{n} \left(1 - \frac{t_k}{\alpha}\right) = \frac{\beta}{n + \beta}, \qquad n = 1, 2, \ldots$$

Exercise A.7.10 If $a_n \approx b_n$, then $b_n \approx a_n$.

Exercise A.7.11 If $a_n \approx 1$ and $b_n \approx 1$, then $a_n b_n \approx 1$.

Exercise A.7.12 If $a_n \approx b_n$ and $b_n \approx c_n$, then $a_n \approx c_n$.

Exercise A.7.13 If $a_n \approx a'_n$ and $b_n \approx b'_n$, then $a_n b_n \approx a'_n b'_n$.

Exercise A.7.14 Let $a \neq 0$. If $a_n \approx a$, then $a_n - a \approx 0$, and conversely.

Exercise A.7.15 If $b_n \approx b$ and $b \neq 0$, then (A.7.2) holds.

Exercise A.7.16 If $a_n - b_n \approx 0$ and $b_n \to b$, then $a_n \to b$.

Exercise A.7.17 Let μ be a constant and let $\bar{X}_1$, $\bar{X}_2$, ... be a sequence of random variables. For example, in the LLN, $\bar{X}_n$ is the sample mean. Show $\bar{X}_n \approx \mu$ is equivalent to

$$Prob(a < \bar{X}_n < b) \approx 0,$$

for any interval (a, b) not containing μ.

Exercise A.7.18 If $a_n \to a$ and $b_n \to b$, then

$$a_n + b_n \to a + b, \qquad a_n b_n \to ab.$$

Exercise A.7.19 If $a_n \leq b_n \leq c_n$ and $a_n \to L$ and $c_n \to L$, then $b_n \to L$ (squeezing lemma).

A.8 Existence of Minimizers

Several times in the text, we deal with minimizing functions, most notably for the pseudo-inverse of a matrix (§2.3), for proper continuous functions (§4.3), and for gradient descent (§7.3).

Previously, the technical foundations underlying the *existence* of minimizers were ignored. In this section, we review the foundational material supporting the existence of minimizers.

For example, since $y = e^x$ is an increasing function, the minimum

$$\min_{0 \leq x \leq 1} e^x = \min\{e^x \mid 0 \leq x \leq 1\}$$

is $y^* = e^0 = 1$, and the minimizer, the location at which the minimum occurs, is $x^* = 0$. Here we have one minimizer.

For the function $y = x^4 - 2x^2$ in Figure 4.4, the minimum over $-2 \leq x \leq 2$ is $y^* = -1$, which occurs at the minimizers $x^* = \pm 1$. Here we have two minimizers.

On the other hand, there is no minimizer for $y = e^x$ on the entire real line $-\infty < x < \infty$, because as x approaches $-\infty$, e^x approaches zero, but never reaches it. Our goal in this section is to establish conditions which guarantee the existence of minimizers.



A sequence x_n is *increasing* if $x_1 \leq x_2 \leq x_3 \leq \ldots$. A sequence x_n is *decreasing* if $x_1 \geq x_2 \geq x_3 \geq \ldots$. If x_n is increasing, then $-x_n$ is decreasing, and vice-versa.

In §A.7, we had bounded sequences and limits. A foundational axiom for real numbers, the *completeness property*, is the following.

Completeness Property

Let x_n be a bounded decreasing sequence of scalars. Then x_n has a limit

$$\lim_{n\to\infty} x_n.$$

By multiplying a sequence by a minus, we also see every bounded increasing sequence has a limit. In general, a bounded sequence need not converge. However below we see it subconverges.



Let $x_1, x_2, \ldots$ be a sequence. A *subsequence* is a selection of terms

$$x_{n_1}, x_{n_2}, x_{n_3}, \ldots, \qquad n_1 < n_2 < n_3 < \ldots.$$

Here it is important that the indices $n_1 < n_2 < n_3 < \ldots$ be strictly increasing.

If a sequence $x_1, x_2, \ldots$ has a subsequence $x'_1, x'_2, \ldots$ converging to x^*, then we say the sequence $x_1, x_2, \ldots$ *subconverges* to x^*. For example, the sequence $1, -1, 1, -1, 1, -1, \ldots$ subconverges to 1 and also subconverges to -1, as can be seen by considering the odd-indexed terms and the even-indexed terms separately.



Bounded Sequences Must Subconverge

Let $x_1, x_2, \ldots$ be a bounded sequence of points in $\mathbf{R}^d$. Then there is a subsequence $x'_1, x'_2, \ldots$ converging to some x^*.

To see this, assume first $x_1, x_2, \ldots$ are scalars, and let $x_1, x_2, \ldots$ be a bounded sequence of numbers, say $a \le x_n \le b$ for $n \ge 1$. Bisect the interval $I_0 = [a, b]$ into two equal subintervals. Then at least one of the subintervals, call it I_1, has infinitely many terms of the sequence. Select x'_1 in I_1 and let x^*_1 be the right endpoint of I_1.

Now bisect I_1 into two equal subintervals. Then at least one of the subintervals, call it I_2, has infinitely many terms of the sequence. Select x'_2 in I_2 and let x^*_2 be the right endpoint of I_2. Continuing in this manner, we obtain a subsubsequence $x'_1, x'_2, \ldots$ with x'_n in I_n, and a sequence $x^*_1, x^*_2, \ldots$.

Since the intervals are nested

$$I_0 \supset I_1 \supset I_2 \supset \ldots,$$

the sequence $x_1^*, x_2^*, \ldots$ is decreasing. By the completeness property,

$$x^* = \lim_{n\to\infty} x_n^*$$

exists. By definition of limit, this says $e_n = x_n^* - x^* \approx 0$.

Since the length of I_n equals $(b-a)/2^n$, and $2^{-n} \to 0$,

$$0 \le x_n^* - x_n' \le (b-a)2^{-n},$$

hence $x_n^* - x_n' \approx 0$. By Exercise **A.7.16**, we conclude $x_n' \to x^*$.

Now let $x_1, x_2, \ldots$ be a sequence of points in $\mathbf{R}^d$, and let v be a vector; then $x_1 \cdot v, x_2 \cdot v, \ldots$ are scalars, so, from the previous paragraph, there is a subsequence $x_n' \cdot v$ (depending on v) converging to some x_v^*.

Let $e_1, e_2, \ldots, e_d$ be the standard basis in $\mathbf{R}^d$. By choosing $v = e_1$, there is a subsequence $x_1', x_2', \ldots$ such that the first features of x_n' converge. By choosing $v = e_2$, and focusing on the subsequence $x_1', x_2', \ldots$, there is a sub-subsequence $x_1'', x_2'', \ldots$ such that the first and second features of x_n'' converge. Continuing in this manner, we obtain a subsequence $x_1^*, x_2^*, \ldots$ such that the k-th feature of the subsequence converges to the k-th feature of a single x^*, for every $1 \le k \le d$. From this, it follows that x_n^* converges to x^*.



Let K be a set of points and let $y = f(x)$ be a scalar-valued function bounded below on K, $f(x) \ge b$ for some number b, for all x in S. Then b is a *lower bound* for $f(x)$ over K.

A *minimizer for* $f(x)$ *over* K is a point x^* *in* K satisfying

$$f(x^*) \le f(x), \qquad \text{for every } x \text{ in } K.$$

As we saw above, a minimizer may or may not exist, and, when the minimizer does exist, there may be several minimizers.

A function $y = f(x)$ is *continuous* if $f(x_n)$ approaches $f(x^*)$ whenever x_n approaches x^*,

$$x_n \to x^* \qquad \Longrightarrow \qquad f(x_n) \to f(x^*),$$

for every x^* and every $x_n \to x^*$. Here $x_n \to x$ means $x_n - x \approx 0$, see §A.7.

A set K is *closed* if x_n in K and $x_n \to x$ implies K. Now we establish

Existence of Minimizers

If $f(x)$ is continuous and K is a bounded and closed set, then there is a minimizer x^* in K,

$$f(x^*) \le f(x), \qquad \text{for every } x \text{ in } K. \tag{A.8.1}$$

To establish the result, let m_1 be a lower bound for $f(x)$ over K, and let x_1 be any point in K. Then $f(x_1) \ge m_1$. Let

$$c = \frac{f(x_1) + m_1}{2}$$

be the midpoint between m_1 and $f(x_1)$.

There are two possibilities. Either c is a lower bound for $f(x)$ over K or not. In the first case, define $m_2 = c$ and $x_2 = x_1$. In the second case, there is a point x_2 in K satisfying $f(x_2) < c$, and we define $m_2 = m_1$. As a consequence, in either case, we have $f(x_2) \ge m_2$, $m_1 \le m_2$, and

$$f(x_2) - m_2 \le \frac{1}{2}(f(x_1) - m_1).$$

Let

$$c = \frac{f(x_2) + m_2}{2}$$

be the midpoint between m_2 and $f(x_2)$.

There are two possibilities. Either c is a lower bound or not. In the first case, define $m_3 = c$ and $x_3 = x_2$. In the second case, there is a point x_3 in K satisfying $f(x_3) < c$, and we define $m_3 = m_2$. As a consequence, in either case, we have $f(x_3) \ge m_3$, $m_2 \le m_3$, and

$$f(x_3) - m_3 \le \frac{1}{2^2}(f(x_1) - m_1).$$

Continuing in this manner, we have a sequence $x_1, x_2, \ldots$ in K, and an increasing sequence $m_1 \le m_2 \le \ldots$ of lower bounds, with

$$f(x_n) - m_n \le \frac{2}{2^n}(f(x_1) - m_1).$$

Since K is bounded, x_n subconverges to some x^*. Since $f(x)$ is continuous, $f(x_n)$ subconverges to $f(x^*)$. Since $f(x_n) - m_n \approx 0$ and m_n is a lower bound for all n, $f(x^*)$ is a lower bound. Since K is closed, x^* is in K. Hence x^* is a minimizer in K.



Let a be a point in $\mathbf{R}^d$ and let r be a positive scalar. The *ball* of radius r and center a is the set of points x satisfying $|x - a|^2 \le r^2$.

Let K be a set of points and let a be a point in K. We say the point a is in the *interior* of K if there is some ball B centered at a and wholly contained in K. Since a is the center of such a ball, the interior of K is part of K.

The *complement* of a set K is the set K^c of points that are not in K.

We say a point a is in the *boundary* of K if every ball centered at a, no matter how small, contains points of K and points of K^c.

In general, the boundary of K need not be part of K. For example, if K is the open unit ball $|x| < 1$, then (Exercise **A.8.4**) the boundary of K is the unit sphere $|x| = 1$, which is not part of K.

It turns out K is closed iff the boundary of K is part of K (Exercise **A.8.5**).

If a set K equals its interior, then we say K is an *open set*. It turns out K is open iff K^c is closed (Exercise **A.8.1**).

Summarizing, when K is closed,

$$K = \text{interior}(K) \cup \text{boundary}(K).$$

Exercises

Exercise A.8.1 The complement of an open set is closed.

Exercise A.8.2 A hyperplane in $\mathbf{R}^d$ is a closed set (look at its complement).

Exercise A.8.3 Let B be a ball in $\mathbf{R}^d$ centered at the origin. Then the span of B is $\mathbf{R}^d$.

Exercise A.8.4 The boundaries of the open ball $\{x : |x - a| < r\}$ and the closed ball $\{x : |x - a| \leq r\}$ both are the sphere $\{x : |x - a| = r\}$.

Exercise A.8.5 A set K is closed iff K contains its boundary.

A.9 SQL

Recall matrices (§2.1), datasets, CSV files, spreadsheets, arrays, dataframes are basically the same objects.

Databases are collections of tables, where a *table* is another object similar to the above. Hence

$$matrix = dataset = CSV file = spreadsheet = table = array = dataframe \tag{A.9.1}$$

One difference is that each entry in a table may be a string, or code, or an image, not just a number. Nevertheless, every table has rows and columns; rows are usually called *records*, and columns are columns.

A *database* is a collection of several tables that may or may not be linked by columns with common data. Software that serves databases is a *database server*. Often the computer running this software is also called a *database*

server, or a *server* for short. Databases created by a database server (software) are stored as files on the database server.

There are many varieties of database server software. We use *MariaDB*, a widely-used open-source database server. By using open-source software, one is assured to be using the "purest" form of the software, in the sense that proprietary extensions are avoided, and the software is compatible with the widest range of commercial variations.

Because database tables can contain millions of records, it is best to access a database server programmatically, using an *application programming interface*, rather than a *graphical user interface*. The basic API for interacting with database servers is SQL (*structured query language*). SQL is a programming language for creating and modifying databases.

Any application on your laptop that is used to access a database is called an SQL *client*. The database server being accessed may be *local*, running on the same computer you are logged into, or *remote*, running on another computer on the internet. In our examples, the code assumes a local or remote database server is being accessed.

Because SQL commands are case-insensitive, by default we write them in lowercase. Depending on the SQL client, commands may terminate with semicolons or not. As mentioned above, data may be numbers or strings.

The basic SQL commands are

```
select from
limit
select distinct
where/where not <column>
where <column> = <data> and/or <column> = <data>
order by <column1>,<column2>
insert into table (<column1>,<column2>,...) \
    values (<data1>, <data2>, ...)
is null
update <table> set <column> = <data> where ...
like <regex> (%, _, [abc], [a-f], [!abc])
delete from <table> where ...
select min(<column>) from <table> (also max, count, avg)
where <column> in/not in (<data array>)
between/not between <data1> and <data2>
as
join (left, right, inner, full)
create database <database>
drop database <database>
create table <table>
truncate <table>
```

```
alter table <table> add <column> <datatype>
alter table <table> drop column <column>
insert into <table> select
```

All the objects in (A.9.1) are also equivalent to a Python list-of-dicts. In this section we explain how to convert between the objects

$$\textit{list-of-dicts} \iff \textit{JSON string} \iff \textit{dataframe} \iff \textit{CSV file} \iff \textit{SQL table} \tag{A.9.2}$$

For all conversions, we use `pandas`. We begin describing a Python *list-of-dicts*, because this does not require any additional Python packages.

A Python *dictionary* or *dict* is a Python object of the form (prices are in cents)

```
item1 = {"dish": "Hummus", "price": 800, "quantity": 5}
```

This is an *unordered* listing of key-value pairs. Here the keys are the strings `dish`, `price`, and `quantity`. Keys need not be strings; they may be integers or any *unmutable* Python objects. Since a Python list is mutable, a key cannot be a list. Values may be any Python objects, so a value may be a list. In a dict, values are accessed through their keys. For example, `item1["dish"]` returns `'Hummus'`.

A *list-of-dicts* is simply a Python list whose elements are Python dicts, for example,

```
item2 = {"dish": "Avocado", "price": 900, "quantity": 2}
L = [item1,item2]
```

Here `L` is a list and

```
len(L), L[0]["dish"]
```

returns `(2,'Hummus')`. In other words, `L` is a *list-of-dicts*,

```
L == [{"dish": "Hummus", "price": 800, "quantity": 5},
   {"dish": "Avocado", "price": 900, "quantity": 2}]
```

returns `True`.

A list-of-dicts `L` can be converted into a string using the `json` module,

```
from json import *
```

```
s  = dumps(L)
L, s
```

Then `L` is a list, and `s` is a string. To emphasize this point, note

- `len(L) == 2` and `len(s) == 99`,
- `L[0:2] == L` and `s[0:2] == '[{'`
- `L[8]` returns an error and `s[8] == ':'`

To convert back the other way, use

```
from  json import *

L1 = loads(s)
L == L1
```

This returns `True`. Strings having this form are called *JSON strings*, and are easy to store in a database as `VARCHAR`s (see Figure A.26).

The basic object in the Python package `pandas` is the *dataframe* (Figures A.23, A.24, A.26, A.27). `pandas` can convert a dataframe `df` to many other formats

```
df.to_dict(), df.to_csv(), df.to_excel(), df.to_sql(), df.to_json(), ...
```

To convert a list-of-dicts to a dataframe is easy. The code

```
from pandas import *

df = DataFrame(L)
df
```

returns the dataframe in Figure A.23 (prices are in cents).

	dish	price	quantity
0	Hummus	800	5
1	Avocado	900	2

Fig. A.23 Dataframe from list-of-dicts.

To go the other way is equally easy. The code

```
L1 =  df.to_dict('records')
L == L1
```

returns `True`. Here the option `'records'` returns a list-of-dicts; other options returns a dict-of-dicts or other combinations.

To convert a CSV file into a dataframe, use the code

```
menu_df = read_csv("menu.csv")
menu_df
```

This returns Figure A.24 (prices are in cents).

	dish	price
0	Hummus	800
1	Baba Ghanouj	800
2	Avocado	900
3	Muhammara	800
4	Mujaddara	800
...	...	...
73	Chicken Kebab, one dip, baklava	2300
74	Lahmbajeen Pizza, two dips	2400
75	Merguez Kebab and Moussaka	2600
76	Margherita Pizza, Chicken Wings, Baklava	2500
77	Any three pizzas	5000

78 rows × 2 columns

Fig. A.24 Menu dataframe and SQL table.

To go the other way, to convert the dataframe `df` to the CSV file `menu1.csv`, use the code

```
df.to_csv("menu1.csv")
df.to_csv("menu2.csv",index=False)
```

The option `index=False` suppresses the index column, so `menu2.csv` has two columns, while `menu1.csv` has three columns. Also useful is the method `.to_excel`, which returns an excel file.

Now we explain how to convert between a dataframe and an SQL table. What we have seen so far uses only `pandas`. To convert to SQL, we need two more packages, `sqlalchemy` and `pymysql`.

The package `sqlalchemy` allows us to connect to a database server from within Python, and the package `pymysql` is the code necessary to complete the connection to the *MariaDB* database server. For example, if we are con-

necting to an *Oracle* database server, we would use the package `cx-Oracle` instead of `pymysql`.

To connect using `sqlalchemy`, we first collect the connection data into one URI string,

```
protocol = "mysql+pymysql://"
credentials = "username:password"
server = "@servername"
port = ":3306"
uri = protocol + credentials + server + port
```

This string contains your database username, your database password, the database server name, the server port, and the protocol. If the database is "rawa", the URI is

```
database = "/rawa"
uri = protocol + credentials + server + port + database
```

Using this uri, the connection is made as follows

```
from sqlalchemy import create_engine

engine = create_engine(uri)
```

(In `sqlalchemy`, a connection is called an "engine".) After this, to store the dataframe `df` into a table `Menu`, use the code

```
df.to_sql('Menu',engine,if_exists = 'replace')
```

The `if_exists = 'replace'` option replaces the table `Menu` if it existed prior to this command. Other options are `if_exists = 'fail'` or `'append'`. The default is `if_exists = 'fail'`, so

```
df.to_sql('Menu',engine)
```

returns an error if `Menu` exists.

To read a table into a dataframe, use for example the code

```
from sqlalchemy import text

query1 = text("select * from rawa.OrdersIn")
query2 = text("select * from rawa.OrdersIn where items
    ↪ like '%Hummus%';")
```

```
connection = engine.connect()
df1 = read_sql(query1, connection)
df2 = read_sql(query2, connection)
```

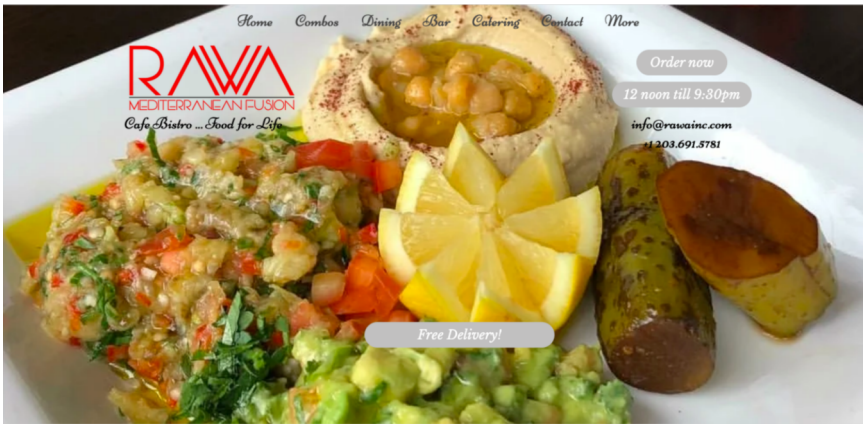


Fig. A.25 Rawa restaurant.

Better Python coding technique is to place `read_sql` and `to_sql` in a `with` block, as follows

```
with engine.connect() as connection:
    df = pd.read_sql(query, connection)
    df.to_sql('Menu',engine)
```

One benefit of this syntax is the automatic closure of the connection upon completion. This completes the conversion between dataframes and SQL tables, and completes the discussion of conversions between any of the objects in (A.9.2).

As an example how all this goes together, here is a task:

> Given two CSV files `menu.csv` and `orders.csv` downloaded from a restaurant website (Figure A.25), create three SQL tables `Menu`, `OrdersIn`, `OrdersOut`.

The three SQL table columns are as follows (price, tip, tax, subtotal, total are in cents)

```
/* Menu */
dish    varchar
price   integer
```

```
/* ordersin */
orderid    integer
created    datetime
customerid integer
items    json

/* ordersout */
orderid integer
subtotal integer
tip      integer
tax      integer
total    integer
```

	orderId	created	customerId	items
0	1	6/29/19 2:37	1497	[{"dish": "Citrus Chicken", "price": 1000, "quantity": 8}]
1	5	7/1/19 14:30	1600	[{"dish": "Hummus", "price": 800, "quantity": 10}]
2	11	7/9/19 17:07	1704	[{"dish": "Hummus", "price": 800, "quantity": 1}, {"dish": "Baba G...
3	12	7/10/19 12:14	1431	[{"dish": "Hummus", "price": 800, "quantity": 9}]
4	13	7/10/19 18:50	1458	[{"dish": "Labne", "price": 800, "quantity": 1}, {"dish": "Garden"...
...	...	...	...	...
3965	3985	1/20/23 17:23	1787	[{"dish": "Hummus", "price": 800, "quantity": 1}, {"dish": "Chicke...
3966	3986	1/20/23 17:37	10	[{"dish": "Falafel", "price": 1300, "quantity": 1}, {"dish": "Chic...
3967	3987	1/20/23 19:16	1354	[{"dish": "Lentil", "price": 800, "quantity": 1}, {"dish": "Marghe...
3968	3988	1/21/23 12:06	152	[{"dish": "Mamyaldi Vegan", "price": 1700, "quantity": 2}]
3969	3989	1/21/23 17:00	1579	[{"dish": "Sampler Plate", "price": 1800, "quantity": 1}, {"dish":...

3970 rows × 4 columns

Fig. A.26 OrdersIn dataframe and SQL table.

To achieve this task, we download the CSV files `menu.csv` and `orders.csv`, then we carry out these steps. (price and tip in `menu.csv` and `orders.csv` are in cents so they are `INT`s.)

1. Read the CSV files into dataframes `menu_df` and `orders_df`.
2. Convert the dataframes into list-of-dicts `menu` and `orders`.
3. Create a list-of-dicts `OrdersIn` with keys `orderId`, `created`, `customerId` whose values are obtained from list-of-dicts `orders`.
4. Create a list-of-dicts `OrdersOut` with keys `orderId`, `tip` whose values are obtained from list-of-dicts `orders` (tips are in cents so they are `INT`s).

5. Add a key `items` to `OrdersIn` whose values are JSON strings specifying the items ordered in `orders`, using the prices in `menu` (these are in cents so they are `INT`s). The JSON string is of a list-of-dicts in the form discussed above `L = [item1, item2]` (see row `0` in Figure A.26).
   Do this by looping over each `order` in the list-of-dicts `orders`, then looping over each `item` in the list-of-dicts `menu`, and extracting the quantity ordered of the item `item` in the order `order`.
6. Add a key `subtotal` to `OrdersOut` whose values (in cents) are computed from the above values.
   Add a key `tax` to `OrdersOut` whose values (in cents) are computed using the Connecticut tax rate `7.35%`. Tax is applied to the sum of subtotal and tip.
   Add a key `total` to `OrdersOut` whose values (in cents) are computed from the above values (subtotal, tax, tip).
7. Convert `OrdersIn`, `OrdersOut` to `OrdersIn_df`, `OrdersOut_df` (list-of-dicts to dataframes).
8. Upload `menu_df`, `OrdersIn_df`, `OrdersOut_df` to tables `Menu`, `OrdersIn`, `OrdersOut`.

The resulting dataframes `ordersin_df` and `ordersout_df`, and SQL tables `OrdersIn` and `OrdersOut`, are in Figures A.26 and A.27.

	orderId	tip	subtotal	tax	total
0	0	0	4000	294	4294
1	1	0	8000	588	8588
2	2	0	2200	161	2361
3	3	0	8800	646	9446
4	4	0	10400	764	11164
...	...	...	...	...	...
3985	3985	434	7800	605	8839
3986	3986	0	3000	220	3220
3987	3987	1275	8200	696	10171
3988	3988	180	3400	263	3843
3989	3989	885	5800	491	7176

3990 rows × 5 columns

Fig. A.27 OrdersOut dataframe and SQL table.

Complete Code for the Task

```
# step 1
from pandas import *

menu_df = read_csv("menu.csv")
orders_df = read_csv("orders.csv")

# step 2
menu = menu_df.to_dict('records')
orders = orders_df.to_dict('records')

# step 3
OrdersIn = h
for r in orders:
    d = {}
    d["orderId"] = r["orderId"]
    d["created"] = r["created"]
    d["customerId"] = r["customerId"]
    OrdersIn.append(d)

# step 4
OrdersOut = h
for r in orders:
    d = {}
    d["orderId"] = r["orderId"]
    d["tip"] = r["tip"]
    OrdersOut.append(d)

# step 5
from json import *

for i,r in enumerate(OrdersIn):
    itemsOrdered = h
    for item in menu:
        dish = item["dish"]
        price = item["price"]
        if dish in orders[i]:
            quantity = orders[i][dish]
            if quantity > 0:
                d = {"dish": dish, "price": price, "quantity":
    ↪ quantity}
                itemsOrdered.append(d)
    r["items"] = dumps(itemsOrdered)

# steps 6
for i,r in enumerate(OrdersOut):
    items = loads(OrdersIn[i]["items"])
    subtotal = sum([ item["price"]*item["quantity"] for item in
    ↪ items ])
    r["subtotal"] = subtotal
```

```
    tip = OrdersOut[i]["tip"]
    tax = int(.0735*(tip + subtotal))
    total = subtotal + tip + tax
    r["tax"] = tax
    r["total"] = total

# step 7
ordersin_df = DataFrame(OrdersIn)
ordersout_df = DataFrame(OrdersOut)

# step 8
from sqlalchemy import create_engine, text

# connect to the database
protocol = "mysql+pymysql://"
credentials = "username:password@"
server = "servername"
port = ":3306"
database = "/rawa"
uri = protocol + credentials + server + port + database

engine = create_engine(uri)

dtype1 = { "dish":sqlalchemy.String(60),
    ↪ "price":sqlalchemy.Integer }

dtype2 = {
   "orderId":sqlalchemy.Integer,
   "created":sqlalchemy.String(30),
   "customerId":sqlalchemy.Integer,
   "items":sqlalchemy.String(1000)
   }

dtype3 = {
   "orderId":sqlalchemy.Integer,
   "tip":sqlalchemy.Integer,
   "subtotal":sqlalchemy.Integer,
   "tax":sqlalchemy.Integer,
   "total":sqlalchemy.Integer
   }

with engine.connect() as connection:
   menu_df.to_sql('Menu', engine,
         if_exists = 'replace', index = False, dtype = dtype1)
   ordersin_df.to_sql("OrdersIn", engine,
         index = False, if_exists = 'replace', dtype = dtype2)
   ordersout_df.to_sql("OrdersOut", engine,
         index = False, if_exists = 'replace', dtype = dtype3)
```

Moral of this section

In this section, all work was done in Python on a laptop, no SQL was used on the database, other than creating a table or downloading a table. Generally, this is an effective workflow:

- Use SQL to do big manipulations on the database (joining and filtering).
- Use Python to do detailed computations on your laptop (analysis).

Exercises

Exercise A.9.1 What is the total number of orders, and the total number of plates?

Exercise A.9.2 What is the average number of plates per order?

Appendix B
Auxiliary Files

For the code in this text to run, it is assumed the `Jupyter` notebooks and CSV files are collected into a single folder on your laptop. These files are available at

https://mathdatasciencebook.com

If the contents of a notebook are needed in another notebook, we *import* the former notebook into the latter. For example, according to the listing below, §7.2's notebook is named `neural_networks.ipynb`. To import §7.2's notebook into §7.4's notebook, we insert

```
%%capture

import import_ipynb
from neural_networks import *
```

into §7.4's notebook.

`Jupyter` Notebooks

1.1 `introduction.ipynb`
1.2 `the_mnist_dataset.ipynb`
1.3 `averages_and_vector_spaces.ipynb`
1.4 `mean_and_variance.ipynb`
1.5 `high_dimensions.ipynb`
2.1 `vectors_and_matrices.ipynb`
2.2 `products.ipynb`
2.3 `matrix_inverse.ipynb`
2.4 `span_and_linear_independence.ipynb`
2.5 `zero_variance_directions.ipynb`

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

O. Hijab, *Math for Data Science*, https://doi.org/10.1007/978-3-031-89707-8

2.7 projections.ipynb
2.8 basis.ipynb
3.2 eigenvalue_decomposition.ipynb
3.4 singular_value_decomposition.ipynb
3.5 principal_component_analysis.ipynb
4.1 single_variable_calculus.ipynb
4.2 entropy_and_information.ipynb
4.4 back_propagation.ipynb
4.5 convexity.ipynb
5.1 probability.ipynb
5.2 binomial_probability.ipynb
5.3 random_variables.ipynb
5.4 normal_distribution.ipynb
5.5 chi_squared_distribution.ipynb
5.6 multinomial_probability.ipynb
6.1 estimation.ipynb
6.2 z_test.ipynb
6.3 t_test.ipynb
6.4 chi_squared_tests.ipynb
7.2 neural_networks.ipynb
7.3 gradient_descent.ipynb
7.4 network_training.ipynb
7.7 regression_examples.ipynb
A.1 permutations_and_combinations.ipynb
A.2 the_binomial_theorem.ipynb
A.3 the_exponential_function.ipynb
A.4 two_dimensions.ipynb
A.5 complex_numbers.ipynb
A.6 integration.ipynb
A.7 asymptotics_and_convergence.ipynb
A.9 sql.ipynb

CSV Files

1. iris.csv
2. longley.csv
3. menu.csv
4. mnist.csv
5. orders.csv

Virtual Environments

To install `Jupyter` in a `venv`, *activate a* `venv` *shell*, then enter

```
pip install jupyter
```

To activate a `venv` shell and start `Jupyter` all at once, we use a batch file. Assuming

- the `venv` folder is named `Python_Virtual_Environment`,
- the notebook/CSV folder is named `Notebooks`,

and both folders are in your home folder, the batch file is

```
@echo off

:: Windows batch file
:: Name this file jupyter.bat

SET VENV=%USERPROFILE%\Python_Virtual_Environment
SET DIR=%USERPROFILE%\Notebooks

CALL %VENV%\Scripts\activate.bat

cd %DIR%
jupyter lab
```

```
#!/bin/zsh

# macOS batch file
# Name this file jupyter

VENV=~/Python_Virtual_Environment
DIR=~/Notebooks

source $VENV/bin/activate

cd $DIR
jupyter lab
```

```
#!/bin/bash

# Linux batch file
# Name this file jupyter

VENV=~/Python_Virtual_Environment
DIR=~/Notebooks
```

```
source $VENV/bin/activate

cd $DIR
jupyter lab
```

Executing the batch file in an OS shell then activates the `venv` and starts `Jupyter`.

References

1. Akey, J.: Genome 560: Introduction to statistical genomics. https://www.gs.washington.edu/academics/courses/akey/56008/lecture/lecture1.pdf (2008)
2. Bishop, C.M.: Pattern Recognition and Machine Learning. Information Science and Statistics. Springer (2006)
3. Block, H.D., Levin, S.A.: On the boundedness of an iterative procedure for solving a system of linear inequalities. Proceedings of the American Mathematical Society **26**(2), 229 – 235 (1970)
4. Bubeck, S.: Convex Optimization: Algorithms and Complexity, *Foundations and Trends in Machine Learning*, vol. 8. Now Publishers (2015)
5. Cramér, H.: Mathematical Methods of Statistics. Princeton University Press (1946)
6. Deisenroth, M.P., Faisal, A.A., Ong, C.S.: Mathematics for Machine Learning. Cambridge University Press (2020)
7. Doob, J.L.: Probability and statistics. Transactions of the American Mathematical Society **36**, 759–775 (1934)
8. Ferguson, T.S.: A Course in Large Sample Theory. Springer (1996)
9. Fisher, R.A.: The conditions under which χ^2 measures the discrepancy between observation and hypothesis. Journal of the Royal Statistical Society **87**, 442–450 (1924)
10. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
11. Google: Machine learning. https://developers.google.com/machine-learning (2025). Image reprinted under license CC-BY-4.0
12. Gray, R.M.: Toeplitz and circulant matrices: A review. Foundations and Trends in Communications and Information Theory **2**(3), 155–239 (2006). URL http://dx.doi.org/10.1561/0100000006
13. Grinberg, E.L., Hijab, O.: The fundamental theorem of trigonometry. Preprint
14. Heath, T.L.: The Works of Archimedes. Cambridge University Press (1897)
15. Hijab, O.: Introduction to Calculus and Classical Analysis, Fourth Edition. Springer (2016)
16. Hijab, O.: Logistic regression and the perceptron algorithm (2025). Preprint
17. Janakiev, N.: Classifying the Iris data set with keras. https://janakiev.com/blog/keras-iris (2018)
18. Jiang, L.: A visual explanation of gradient descent methods. https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c (2020)
19. Longley, J.W.: An appraisal of least squares programs for the electronic computer from the point of view of the user. Journal of the American Statistical Association **62.319**, 819–841 (1967)

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

O. Hijab, *Math for Data Science*, https://doi.org/10.1007/978-3-031-89707-8

20. Luenberger, D.G., Ye, Y.: Linear and Nonlinear Programming. Springer (2008)
21. Minsky, M.L., Papert, S.A.: Perceptrons, expanded edition. Massachusetts Institute of Technology (1987). Image reprinted by permission of the MIT press
22. Nesterov, Y.: Lectures on Convex Optimization. Springer (2018)
23. Novikov, A.J.: On convergence proofs for perceptrons. Tech. rep., Office of Naval Research (1963)
24. Pearson, K.: On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. Philosophical Magazine Series 5 **50:302**, 157–175 (1900)
25. Penrose, R.: A generalized inverse for matrices. Proceedings of the Cambridge Philosophical Society **51**, 406–413 (1955)
26. Polyak, B.T.: Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics **4(5)**, 1–17 (1964)
27. Raschka, S.: PCA in three simple steps. https://sebastianraschka.com/Articles/2015_pca_in_3_steps.html (2015). Image reprinted by permission of the author
28. Robbins, H., Monro, S.: A stochastic approximation method. The Annals of Mathematical Statistics **22**(3), 400 – 407 (1951)
29. Rosenblatt, F.: Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Spartan Books (1962)
30. Ross, S.M.: Probability and Statistics for Engineers and Scientists, Sixth Edition. Academic Press (2021)
31. Ross, S.M.: Introduction to Probability Models, Thirteenth Edition. Academic Press (2023)
32. Schervish, M.J.: Theory of Statistics. Springer (1995)
33. Stanford University: CS224N: Natural language processing with deep learning. https://web.stanford.edu/class/cs224n
34. Steinwart, I., Christmann, A.: Support Vector Machines. Springer (2008)
35. Strang, G.: Linear Algebra and its Applications. Brooks/Cole (1988)
36. The WeBWorK Project: https://openwebwork.org/
37. Wikipedia: Logistic regression. https://en.wikipedia.org/wiki/Logistic_regression
38. Wikipedia: Seven bridges of Königsberg. https://en.wikipedia.org/wiki/Seven_Bridges_of_Konigsberg
39. Wright, S.J., Recht, B.: Optimization for Data Analysis. Cambridge University Press (2022)

Python Index

*, 9, 16

append, 185

def.Angle, 504
def.assign_clusters, 185
def.backward_prop, 229, 238, 409
def.ball, 37
def.batch_weight_gradient, 430
def.biases, 402
def.cartesian_product, 340
def.chi2_independence, 385
def.cluster, 185
def.comb_tuples, 479
def.confidence_interval, 368, 377
def.derivative, 238
def.dimension_staircase, 111
def.display_image, 179
def.downstream, 409
def.draw_major_minor_axes, 28
def.edges, 402
def.ellipse, 29
def.find_first_defect, 110
def.forward_prop, 229, 236, 403
def.gd, 420
def.goodness_of_fit, 381
def.H, 281
def.hexcolor, 11
def.incoming, 235, 403
def.initial_weights, 402
def.inject_source, 403
def.inject_target, 408
def.inputs, 402
def.J, 405
def.local, 407
def.logreg, 457
def.matrix_text, 28
def.nearest_index, 185
def.neurons, 402
def.newton, 414
def.num_biases, 411
def.num_edges, 411
def.num_inputs, 411
def.num_legendre, 193
def.num_neurons, 411
def.num_outputs, 411
def.outgoing, 235, 403
def.outputs, 402
def.pca, 178
def.pca_with_svd, 179
def.perm_tuples, 477
def.plot_and_integrate, 532
def.plot_cluster, 186
def.plot_descent, 415
def.poly, 454
def.project, 101
def.project_to_ortho, 103

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

O. Hijab, *Math for Data Science*, https://doi.org/10.1007/978-3-031-89707-8

def.pvalue, 330
def.random_batch_mean, 272
def.sample_training
  batch, 430
  minibatch, 432
  single, 427
  stochastic, 432
def.set_pi_ticks, 207
def.single_weight_gradient, 427
def.stirling, 281
def.sym_legendre, 192
def.tensor, 426, 513
def.ttest, 378
def.type2_error, 372, 379
def.uniq, 5
def.update_means, 185
def.update_weights, 427
def.zero_variance, 87
def.ztest, 371
dict, 547

enumerate, 180

import, 9
ipython.display, 44
itertools.product, 37

join, 11
json.dumps, 547
json.loads, 548

lambda, 235
lamda, 130

map, 207
matplotlib.patches
  Circle, 37
  Rectangle, 37
matplotlib.pyplot.axes, 37
  add_patch, 37
  axis, 37
  set_axis_off, 37
matplotlib.pyplot.contour, 29
matplotlib.pyplot.figure, 180
matplotlib.pyplot.grid, 7
matplotlib.pyplot.hist, 269
matplotlib.pyplot.imshow, 8, 9
matplotlib.pyplot.legend, 29
matplotlib.pyplot.meshgrid, 29
matplotlib.pyplot.plot, 18, 20
matplotlib.pyplot.scatter, 7, 18
matplotlib.pyplot.show, 7
matplotlib.pyplot.stairs, 111
matplotlib.pyplot.subplot, 180
matplotlib.pyplot.text, 28
matplotlib.pyplot.title, 281
matplotlib.pyplot.xlabel, 455
matplotlib.pyplot.xticks, 207
matplotlib.pyplot.ylim, 336

numpy.allclose, 130, 500
numpy.amax, 455
numpy.amin, 455
numpy.arange, 18, 29
numpy.arccos, 504
numpy.argmin, 185
numpy.argsort, 178
numpy.array, 8, 41
numpy.ceil, 207
numpy.column_stack, 63
numpy.copy, 427
numpy.corrcoef, 24, 58
numpy.cov, 24, 58
numpy.cumsum, 178
numpy.degrees, 504
numpy.diag, 47, 172
numpy.dot, 48
numpy.dstack, 340
numpy.exp, 281
numpy.eye, 46
numpy.fill_diagonal, 402
numpy.floor, 207
numpy.full, 402
numpy.inf, 534
numpy.invert, 402
numpy.isclose, 140, 500

numpy.linalg.matrix_rank, 109, 110, 115
numpy.linspace, 37
numpy.log, 281
numpy.mean, 14
numpy.meshgrid, 37, 340
numpy.ones, 46
numpy.outer, 385, 513
numpy.pi, 281
numpy.random.default_rng, 16
  binomial, 268, 279, 361
  choice, 11
  normal, 330, 360, 434
  random, 15, 20, 243, 402
  shuffle, 272, 403
numpy.reshape, 7
numpy.roots, 527
numpy.row_stack, 45
numpy.set_printoptions, 9
numpy.shape, 41
numpy.sort, 178
numpy.sqrt, 504
numpy.where, 402
numpy.zeros, 46, 172

pandas.DataFrame, 548
pandas.DataFrame.to_csv, 549
pandas.DataFrame.to_dict, 548
pandas.DataFrame.to_sql, 550
pandas.read_csv, 454, 549
pandas.read_sql, 550

scipy.integrate.quad, 531
scipy.linalg.block_diag, 47
scipy.linalg.eig, 126
scipy.linalg.eigh, 127, 178
scipy.linalg.inv, 62
scipy.linalg.norm, 185, 502
scipy.linalg.null_space, 76
scipy.linalg.orth, 70
scipy.linalg.pinv, 64, 101
scipy.linalg.svd, 172
scipy.optimize.newton, 218
scipy.spatial.ConvexHull, 243
  simplices, 243
scipy.special.comb, 479
scipy.special.expit, 287
scipy.special.factorial, 476
scipy.special.perm, 477
scipy.special.softmax, 348
scipy.stats
  .multivariate_normal, 340
scipy.stats.binom, 279
scipy.stats.chi2, 336
scipy.stats.entropy, 217, 281
scipy.stats.norm, 318
scipy.stats.poisson, 306
scipy.stats.t, 375, 377
sklearn.datasets.load_iris, 2
sklearn.decomposition
  .PCA, 180
sklearn.preprocessing
  .StandardScaler, 58
sqlalchemy.create_engine, 550
sqlalchemy.text, 550
sympy.*, 48
sympy.diag, 47
sympy.diagonalize, 131
sympy.diff, 192
sympy.eigenvects, 131
sympy.lambdify, 193
sympy.Matrix, 41
sympy.Matrix.col, 45
sympy.Matrix.cols, 45
sympy.Matrix.columnspace, 69
sympy.Matrix.eye, 46
sympy.Matrix.hstack, 44, 64, 77
sympy.Matrix.inv, 62
sympy.Matrix.nullspace, 75
sympy.Matrix.ones, 46
sympy.Matrix.rank, 115
sympy.Matrix.row, 45
sympy.Matrix.rows, 45
sympy.Matrix.rowspace, 73
sympy.Matrix.zeros, 46
sympy.prod, 528
sympy.shape, 41
sympy.simplify, 192
sympy.solve, 301, 526
sympy.symbols, 192

tuple, 499

zip, 183

Index

≈, 535
1, 146, 156, 344, 347, 441

angle, 117
Archimedes
  angle measure, 522
  axiom, 208
arcsine law, 148, 209
asymptotically
  equal, 387, 536
  nonzero, 536
  normal, 387
  one, 536
  positive, 536
  zero, 387, 535
average, 11

basis, 107
  of eigenvectors, 129
  of singular vectors, 169
  one-hot encoded, 73
  orthonormal, 108, 118, 129
  standard, 42, 73
Bayes theorem, 284, 286, 288
  perceptron, 290
binary classifiers, 391
  fundamental theorem of, 392, 446
  linear, 391
binomial, 482
  coefficient, 479, 483, 484
  density, 285
  theorem, 482, 484
    chi-squared, 211
    Newton's, 205
bound, 223

cartesian plane, 498
Cauchy-Schwarz inequality, 50, 505
central limit theorem, 270, 321
  and Stirling's approximation, 481
chi-squared, 334
  correlated, 339, 342
circle, 503
  unit, 502
coin-tossing, 276
  bias, 276
  entropy, 280
    relative, 282
column space, 69
columns, 44
  orthonormal, 54
combination, 478
  convex, 242
  linear, 67
complex
  conjugate, 520
  division, 519, 521

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025

O. Hijab, *Math for Data Science*, https://doi.org/10.1007/978-3-031-89707-8

hermitian product, 520
multiplication, 519, 520
numbers, 519
plane, 519
polar representation, 523
roots of unity, 523
concave, 253
strictly, 253
concave function, 198
condition number, 461
confidence, 326
interval, 366
level, 365
contingency table, 384
converges, 538
convex
combination, 242
dual, 203, 255, 350, 354
hull, 243
set, 242
separable, 252
convex function, 198, 253
strictly, 199, 253
strongly, 187, 201, 461
correlation
coefficient, 22, 57
matrix, 22, 57
negative, 23, 32, 57
positive, 23, 32, 57
cumulant-generating function, 214, 295, 346
and variance, 300

dataset, 1
attributes, 1
augmented, 397, 458
centered, 12
dimension, 118
example, 1
features, 1
full-rank, 118, 249
inseparable, 250, 446
strongly, 445
weakly, 445
Iris, 1
label, 1
mean, 19
MNIST, 5
multi-class, 440
observation, 1
projected, 25, 103, 180
reduced, 26, 84, 103, 180
sample, 1
separable, 250
strongly, 445
weakly, 445
soft-class, 440
standard, 21, 22, 57
target, 1
two-class, 250, 391, 440
variance, 21
vectors or points, 13
decision boundary, 250, 289, 445
degree
binomial, 482
chi-squared, 334
graph node, 154
sequence, 154
derivative, 187
directional, 218
downstream, 233, 407
formula, 189
local, 407
logarithm, 198
maximizers, 195
minimizers, 195
partial, 219
second, 191, 225
convexity, 199
strict convexity, 199
upstream, 233, 407
descent
gradient, 416
Newton, 414
sequence, 416
diagonalizable, 130
diagonalization
eigen, 129
singular, 171
dice-rolling

bias, 353
entropy, 351
relative, 353
dimension, 107
staircase, 111
direct sum, 104
distance formula, 501
distribution
arcsine, 316
Bernoulli, 279
binomial, 278
chi-squared, 334, 335
exponential, 316
logistic, 311
normal, 317, 320
Poisson, 306
Student, 374
T-, 374
uniform, 308
Z-, 317, 320
dot product, 48, 504

eigenspace, 140
eigenvalue, 126
bottom, 138
clustering, 148
decomposition, 129
minimum variance, 138
projected variance, 136
top, 136
transpose, 127
eigenvectors, 126
best-aligned vector, 136
is right singular vector, 173
linearly independent, 128
orthogonal, 128
entropy, 211, 350
absolute, 212, 350
cross-, 355
relative, 216, 352
error
information, 354
mean logistic, 404, 442
mean square, 404, 436
Euler, 157, 491, 539
Euler's constant, 491
events, 261
certain, 262
complementary, 262
difference, 262
exclusive, 262
exhaustive, 262
highly significant, 327
impossible, 262
independent, 267
intersection, 262
null, 264
significant, 327
sure, 264
symmetric, 276
union, 262
experiment, 261
exponential
function, 492
matrix, 60
series, 494

factorial, 475
full-rank
dataset, 118
matrix, 115
function
beta, 291
concave, 198
convex, 198, 253
cumulant-generating, 214, 295, 346
independence, 304
relative, 353
cumulative distribution, 294, 309
error, 413, 434
level, 223
logistic, 214, 287
logit, 214
loss, 413, 434
mean error, 393
mean loss, 393
moment-generating
chi-squared, 335

independence, 302, 303
normal, 320
standard normal, 319
probability density, 306, 317
probability mass, 294
proper, 223
and trainability, 435
relu, 315, 397
central limit theorem, 333
Stirling's approximation, 333
sigmoid, 214, 287
softmax, 347
relative, 356
strictly convex, 199, 253
fundamental theorem of
algebra, 526
binary classifiers, 392, 446
calculus, 529

geometric
series, 497
sum, 489
gradient, 219
descent, 414, 416
accelerated, 472
convex case, 462
heavy ball, 471
I, 463
II, 464
learning rates, 416, 418, 423
momentum, 469
short-step, 418, 463
stochastic, 421, 433, 465
with lookahead gradient, 473
weight, 410, 426
graph, 151
bipartite, 162
complement, 159
complete, 153
component, 161
connected, 156
cycle, 154, 156
directed, 151
edge, 151, 393
incoming, 393
outgoing, 393
eulerian, 158
forest, 156
isomorphism, 161
laplacian, 164
node
input, 154
output, 154
nodes, 151, 393
adjacent, 151
connected, 156
degree, 154
dominating, 155
hidden, 393
in-degree, 154
input, 393
isolated, 155
out-degree, 154
output, 393
order, 153
path, 156
regular, 155
simple, 153
size, 153
sub-, 154
tree, 156
undirected, 151
vertex, 151
walk, 155
eulerian, 157
weighed, 151
weight matrix, 393
wheel, 154

hyperplane, 84, 245
level, 246
LR, 446
maximum-margin, 252
separating, 247, 250
separation theorem, 251
suporting, 248
weight, 436
hypothesis
alternate, 369
null, 369

testing, 369

iff, 54, 127
incoming edge, 393
information, 213, 349
absolute, 213, 349
cross-, 355
relative, 215, 352
integral, 528
additivity, 530
scaling, 530
inverse, 61
pseudo-, 62, 90
Iris dataset, 1

Jupyter, 4

Kantorovich's inequality, 465

law of large numbers, 270, 283, 320, 362
learning rates, 416, 418, 423
short-step, 418, 463
Legendre polynomial, 192
limit, 537
line-search, 465
linear
combination, 67
dependence, 75
independence, 75
system, 61, 134
homogeneous, 75, 507
inhomogeneous, 509
transformation, 114, 121
log-odds, 214
logistic function, 214, 287
logistic regression
trainability, 442
logit function, 214
loss, 413
mean logistic, 404, 442
mean square, 404, 436

machine learning, 392
margin of error, 365
mass-spring system, 143
matrix, 42
addition, 46
adjacency, 152
augmented, 72
centered, 442
circulant, 147
eigenvalues, 147
columns, 509
correlation, 22, 57
diagonal, 45
identity, 61
incidence, 164
inverse, 61, 515
network adjacency, 399
network weight, 399
nonnegative, 26, 53
orthogonal, 116
permutation, 48
positive, 26, 53
projection, 98, 100
pseudo-inverse, 93
rank, 115
approximate, 132
rows, 509
scaling, 46
square, 45
symmetric, 53, 510
trace, 52, 509
transpose, 43, 510
variance, 21, 26, 56, 135
weight, 152, 393
maximizer, 194
global, 194
local, 194, 223
mean, 11, 19, 294, 306, 338
sample, 313
minimizer, 541, 543
existence, 224
global, 194, 223
local, 194, 223
properness, 224
residual, 225

network, 235, 393
deep, 410

incoming list, 234, 394
neural, 394
incoming signal, 394
layered, 410
training, 425
neuron, 235, 393
outgoing signal, 234, 393
perceptron, 395
shallow, 410
dense, 410
trainability, 435
training
batch sample, 430
epoch, 393, 430
iteration, 393, 430
minibatch sample, 432
single sample, 427
stochastic sample, 432
Newton's method, 414
norm, 55, 501
nullspace, 75

1, 146, 156, 344, 347, 441
one-hot encoding, 73, 346, 355, 440
orthogonal, 50
complement, 78, 104
orthonormal, 50
outcome, 261
outgoing edge, 393

parabola
lower tangent, 201
upper tangent, 201
Pascal's triangle, 485
perceptron, 290, 395
Bayes theorem, 397
parallel, 410
permutation, 477
perp, 78, 507
point, 41
critical, 194, 223, 419
inflection, 199, 420
point of best-fit, 19
population, 10
power of a test, 373
principal axes, 33
principal components, 130, 135, 176
probability
additivity, 264
binomial, 276
chain rule, 267
coin-tossing, 277
conditional, 266
monotonicity, 264
multiplication of, 278
one-hot encoded, 440
strict, 440
sub-additivity, 264
product
dot, 48, 117, 504
linearity, 504
matrix-matrix, 51, 512
linearity, 513
matrix-vector, 51, 510
linearity, 511
tensor, 54, 513
linearity, 513
projection, 98
matrix, 100
onto column space, 101
onto nullspace, 105
onto row space, 102
propagation
back, 228, 229
chain, 229
network, 238
neural network, 409
forward, 227, 229
chain, 229
network, 236
neural network, 403
proper function, 223
minimizer, 224
Pythagoras theorem, 505
Python, 4

random variables, 291, 292
arcsine, 316

Bernoulli, 279, 292
binomial, 304
chi-squared, 334
continuous, 293
correlation, 300
discrete, 293
expectation, 294, 306
conditional, 316
exponential, 316
gaussian, 317
identically distributed, 312
independence, 301
logistic, 311
normal, 317
Poisson, 306
standard, 312
Student, 374
vector-valued, 338
rank, 115
and eigenvalues, 132
and singular values, 169
approximate, 132
column, 70
full-, 115
nonzero eigenvalues, 132
row, 73
regression
hyperplane, 451
linear, 436, 439, 451, 453
convexity, 436, 437
neural network, 436
properness, 438
trainability, 439
logistic, 391, 442
convexity, 443
neural network, 442
properness, 444
trainability, 391, 446
subspace, 452
regularization, 419
relu function, 315, 397
central limit theorem, 333
Stirling's approximation, 333
residual, 89
vanishing, 90
residual minimizer, 90
and properness, 224
minimum norm, 92
pseudo-inverse, 93
regression equation, 91
row space, 73
rows, 44
orthonormal, 54

scalars, 13
scaling
factor, 122
integral, 530
matrix, 46
principle, 36
vector, 42, 500
sequence, 535
convergent, 538
sub-, 542
subconvergent, 542
series
alternating, 487
exponential, 494
Taylor, 195
set
ball, 544
boundary, 241, 545
complement, 545
convex, 242
interior, 544
level, 240
open, 545
sublevel, 223, 241
sigmoid function, 214, 287
singular
value, 166
decomposition, 169, 171
of pseudo-inverse, 174
versus eigenvalue, 168
vectors, 167
left, 167
right, 167
versus eigenvectors, 173
singular values
transpose, 167

slope, 187
softmax function, 347
space
  column, 69
  eigen-, 140
  feature, 41, 73
  null, 75
  outcome, 261
  row, 73
  sample, 10, 261
  source, 114
  sub-, 80
  target, 114
  vector, 11
span, 69
spherical coordinates, 37
standard
  deviation, 296
  error, 328
statistic, 16
Stirling's approximation, 481
  central limit theorem, 333
  relu function, 333
sum
  direct, 104
  geometric, 489
  harmonic, 539
  of spans, 104
  of vectors, 42
suspensions, 39
system
  linear
    homogeneous, 507
    inhomogeneous, 509

tangent
  line, 188
test
  chi-squared, 381, 385
  goodness of fit, 381
  independence, 385
  T, 378
  Z, 371
trainability, 435
  and properness, 435
  linear regression, 439
transpose, 43
triangle inequality, 50

unit circle, 502

variance, 20, 21, 56, 84
  biased, 24
  ellipse, 27
  explained, 25
  inverse ellipse, 27
  inverse ellipsoid, 139
  matrix, 21, 135
  projected, 26, 84, 123, 128
  reduced, 26, 84
  sample, 343
  total, 25
  unbiased, 24
  zero direction, 84
vector, 12, 41, 498
  addition, 42, 499
  best-aligned, 31
  bias, 411
  cartesian, 498
  centered, 348
  dimension, 41
  dot product, 504
  gradient, 219
    downstream, 407
  incoming, 394
  length, 49, 501
  magnitude, 49
  norm, 49, 501
  one-hot encoded, 73, 346, 355, 440
  orthogonal, 50, 505
  orthonormal, 50, 77, 505
  outgoing, 233, 394
  perp, 78
  perpendicular, 505
  polar, 502
  probability, 346, 381
    strict, 346, 440
  projected, 98, 99, 101, 103
  random, 338, 360

standard, 338
reduced, 98, 99, 101, 103
scaling, 42, 500
shadow, 498
span, 68
subtraction, 501
unit, 49, 502
zero, 42, 498
vectorization, 16, 382
vectors
scaling
linearity, 500
weight, 393
gradient, 410, 426
hyperplane, 436
matrix, 152
centered, 442