



Data Analysis

with Open Source Tools

O'REILLY®

Philipp K. Janert

Table of Contents

Copyright.....	1
Preface.....	11
Chapter 1: Introduction: Data and Data Analysis.....	13
Chapter 2: A Single Variable: Univariate Data.....	35
Chapter 3: Two Variables: Establishing Relationships.....	73
Chapter 4: Time as a Variable: Time Series Analysis.....	107
Chapter 5: More Than Two Variables: Graphical Multivariate Analysis.....	129
Chapter 6: Intermezzo: A Data Analysis Session.....	151
Chapter 7: Guesstimation and the Back of the Envelope.....	165
Chapter 8: Arguments from Scale.....	187
Chapter 9: Arguments from Probability Models.....	213
Chapter 10: What You Really Need to Know About Classical Statistics.....	243
Chapter 11: Intermezzo: Mythbusting--Bigfoot, Least Squares, and All That.....	275
Chapter 12: Simulation.....	289
Chapter 13: Clustering.....	291
Chapter 14: Seeing the Forest for the Trees: Finding Important Attributes.....	325
Chapter 15: Intermezzo: When More is Different.....	349
Chapter 16: Reporting, Business Intelligence, and Dashboards.....	359
Chapter 17: Financial Calculations and Modeling.....	379
Chapter 18: Predictive Analytics.....	397
Chapter 19: Epilogue: Facts Are Not Reality.....	421
Appendix A: Programming Environments for Science and Data Analysis.....	423
Appendix B: Results from Calculus.....	435

Data Analysis

with Open Source Tools

Philipp K. Janert

Licensed by
Shahriyar Matloub
1969616

Contents

Preface	ix
1 Introduction: Data and Data Analysis	1
1.1 Data	1
1.1.1 Types of Data	1
1.1.2 Types of Data Sets	3
1.1.3 Data and “Big Data”	4
1.2 From Data to Analysis	5
1.2.1 Sources for Data	5
1.2.2 Cleaning and Conditioning	6
1.2.3 Sampling	9
1.2.4 Data File Formats	10
1.2.5 Care and Feeding of Your Data Zoo	11
1.2.6 Skills	12
1.3 Analysis	14
1.4 My Philosophy	16
1.5 Workshop: Four Unix Tools	17
1.5.1 awk	17
1.5.2 sort	19
1.5.3 paste	19
1.5.4 gzip	19
1.6 Further Reading	20
I Graphics: Looking at Data	21
2 A Single Variable: Univariate Data	23
2.1 Dot and Jitter Plot	24
2.2 Histograms and Kernel Density Estimates	27
2.2.1 Histograms	27
2.2.2 Kernel Density Estimates	31
2.2.3 Optimal Bandwidth Selection	34
2.3 Cumulative Distribution Function	35
2.3.1 Comparing Distributions with Probability- or QQ-plots .	38

2.4	Rank-Order Plots and Lift Charts	42
2.5	Only when Appropriate: Summary Statistics and Box Plots	45
2.5.1	Summary Statistics and Box-and-Whisker Plots	46
2.5.2	Box-and-Whisker Plots	48
2.6	Workshop: NumPy	51
2.6.1	NumPy in Action	51
2.6.2	NumPy in Detail	54
2.7	Summary	56
2.8	Further Reading	57
3	Two Variables: Establishing Relationships	61
3.1	Scatter Plots	61
3.2	Conquering Noise: Smoothing	62
3.2.1	Splines	64
3.2.2	LOESS	64
3.2.3	Examples	65
3.2.4	Residuals	68
3.2.5	Additional Ideas and Warnings	69
3.3	Logarithmic Plots	71
3.4	Banking	74
3.5	Linear Regression and all that	76
3.6	Showing What's Important	80
3.7	Graphical Analysis and Presentation Graphics	82
3.8	Workshop: matplotlib	83
3.8.1	Using Matplotlib Interactively	84
3.8.2	Managing Properties	87
3.8.3	Case Study: LOESS with Matplotlib	88
3.8.4	The Matplotlib Object Model and Architecture	89
3.8.5	Odds and Ends	90
3.9	Further Reading	91
4	Time as a Variable: Time Series Analysis	95
4.1	Examples	95
4.2	The Task	99
4.2.1	Requirements and the Real World	100
4.3	Smoothing	101
4.3.1	Running Averages	101
4.3.2	Exponential Smoothing	102
4.4	Don't Overlook the Obvious!	106
4.5	Correlation Function	108
4.5.1	Examples	109
4.5.2	Implementation Issues	109
4.6	Filters and Convolutions	112
4.7	Workshop: scipy.signal	113
4.8	Summary	114
4.9	Further Reading	114

5 More Than Two Variables	117
5.1 False-Color Plots	118
5.2 A lot at a Glance: Multiplots	123
5.2.1 The Scatter-Plot Matrix	123
5.2.2 The Co-Plot	125
5.2.3 Variations	127
5.3 Composition Problems	128
5.3.1 Changes in Composition	128
5.3.2 Multidimensional Composition: Tree- and Mosaic-Plots	131
5.4 Novel Plot Types	131
5.4.1 Glyphs	131
5.4.2 Parallel Coordinate Plots	132
5.5 Interactive Explorations	135
5.6 Toolbox	135
5.6.1 R	136
5.6.2 ggobi and Mondrian	136
5.6.3 Python Chaco Library	137
5.7 Further Reading	137
6 Intermezzo: A Data Analysis Session	139
6.1 A Data Analysis Session	139
6.2 Workshop: gnuplot	148
6.3 Further Reading	150
II Analytics: Modeling Data	151
7 Guesstimation and the Back of the Envelope	153
7.1 Principles of Guesstimation	154
7.1.1 Estimating Sizes	155
7.1.2 Establishing Relationships	157
7.1.3 Working With Numbers	157
7.1.4 More Examples	160
7.1.5 Things I Know	163
7.2 How good are those numbers?	163
7.2.1 Before You Get Started: Feasibility and Cost	164
7.2.2 After You Finish: Quoting and Displaying Numbers	164
7.3 A Closer Look at Perturbation Theory and Error Propagation	165
7.3.1 Error Propagation	167
7.4 Workshop: The Gnu Scientific Library (GSL)	168
7.5 Further Reading	170

8 Arguments from Scale	175
8.1 Models	175
8.1.1 Modeling	176
8.1.2 Using and Misusing Models	176
8.2 Arguments from Scale	177
8.2.1 Scaling Arguments	177
8.2.2 Case Study: A Simple Cost Model	180
8.2.3 Typical Scales and Collapsing Curves	182
8.2.4 Finding the Right Scales	182
8.2.5 Scaling Arguments versus Dimensional Analysis	182
8.2.6 Other Arguments	183
8.3 Mean-Field Approximations	184
8.3.1 Background and Further Examples	185
8.4 Common Time-Evolution Scenarios	187
8.4.1 Unconstrained Growth and Decay Phenomena	187
8.4.2 Constrained Growth: The Logistic Equation	189
8.4.3 Oscillations	190
8.5 Case Study: How Many Servers Are Best?	191
8.6 Why Modeling?	193
8.7 Workshop: SAGE	194
8.8 Further Reading	198
9 Arguments from Probability Models	201
9.1 The Binomial Distribution and Bernoulli Trials	201
9.1.1 Exact Results	202
9.1.2 Using Bernoulli Trials To Develop Mean-Field Models .	204
9.2 The Gaussian Distribution and the Central Limit Theorem .	205
9.2.1 The Central Limit Theorem	205
9.2.2 The Central Term and The Tails	207
9.2.3 Why is the Gaussian so Useful?	208
9.2.4 Beware: The World is Not Normal!	210
9.3 Power Law Distributions and non-Normal Statistics	211
9.3.1 Working with Power-Law Distributions	213
9.3.2 Theory: Distributions with Infinite Expectation Values .	215
9.3.3 Where To Go From Here	216
9.4 Other Distributions	216
9.4.1 Geometric Distribution	216
9.4.2 Poisson Distribution	217
9.4.3 Log-Normal Distribution	219
9.4.4 Special Purpose Distributions	220
9.5 Case Study: Unique Visitors over Time	221
9.6 Workshop: Power Law Distributions	224
9.7 Further Reading	227

10 What You Really Need to Know About Classical Statistics	231
10.1 Genesis	231
10.2 Statistics, defined	233
10.3 Statistics, explained	236
10.3.1 Example: Formal Tests versus Graphical Methods	238
10.4 Controlled Experiments vs Observational Studies	240
10.4.1 Design of Experiments	242
10.4.2 Perspective	244
10.5 The Other Point of View: Bayesian Statistics	245
10.5.1 The Frequentist Interpretation of Probability	245
10.5.2 The Bayesian Interpretation of Probability	246
10.5.3 Bayesian Data Analysis — A Worked Example	247
10.5.4 Bayesian Inference: Summary and Discussion	249
10.6 Workshop: R	252
10.7 Further Reading	257
11 Intermezzo: Mythbusting — Bigfoot, Least Squares, and All That	263
11.1 How to Average Averages	263
11.1.1 Simpson's Paradox	265
11.2 The Standard Deviation	266
11.2.1 How to calculate	268
11.2.2 One over What?	269
11.2.3 The Standard Error	269
11.3 Least Squares	270
11.3.1 Statistical Parameter Estimation	272
11.3.2 Function Approximation	273
III Computation: Mining Data	275
12 Simulation	277
13 Clustering	279
13.1 What constitutes a Cluster?	279
13.1.1 A different point of view	283
13.2 Distance and Similarity Measures	285
13.2.1 Common Distance and Similarity Measures	287
13.3 Clustering Methods	291
13.3.1 Center Seekers	292
13.3.2 Tree Builders	294
13.3.3 Neighborhood Growers	296
13.4 Pre- and Post-Processing	298
13.4.1 Scale Normalization	298
13.4.2 Cluster Properties and Evaluation	299
13.5 Other Thoughts	302
13.6 A Special Case: Market-Basket Analysis	304

13.7 A Word of Warning	306
13.8 Toolbox: Pycluster and the C Clustering Library	307
13.9 Further Reading	309
14 Seeing the Forest for the Trees	313
14.1 Introduction	313
14.2 Principal Component Analysis	314
14.2.1 Introduction	314
14.2.2 Theory	317
14.2.3 Computation	319
14.2.4 Interpretation	320
14.2.5 Practical Points	321
14.2.6 Biplots	322
14.3 Visual Techniques	323
14.3.1 Multidimensional Scaling	323
14.3.2 Network Graphs	324
14.3.3 Grand Tours and Projection Pursuits	325
14.4 Kohonen Maps	326
14.5 Perspective	329
14.6 Toolbox: PCA with R	329
14.7 Further Reading	335
14.7.1 Linear Algebra	336
15 Intermezzo: When More is Different	337
15.1 A Horror Story	339
15.2 Some Suggestions	340
15.3 What about map/reduce?	342
15.4 Workshop: Permutations	343
15.5 Further Reading	344
IV Applications: Using Data	345
16 Reporting, Business Intelligence, and Dashboards	347
16.1 Business Intelligence	348
16.1.1 Reporting	350
16.2 Corporate Metrics and Dashboards	354
16.2.1 Recommendations for a Metrics Program	355
16.3 Data Quality Issues	358
16.3.1 Data Availability	359
16.3.2 Data Consistency	360
16.4 Workshop: Berkeley DB and SQLite	362
16.4.1 Berkeley DB	363
16.4.2 SQLite	364

17 Financial Calculations and Modeling	367
17.1 The Time-Value of Money	368
17.1.1 A single payment: Future and Present Value	368
17.1.2 Multiple payments: compounding	370
17.1.3 Calculational Tricks with Compounding	371
17.1.4 The whole picture: cashflow analysis and net present value	373
17.2 Uncertainty in Planning and Opportunity Costs	375
17.2.1 Using Expectation Values to Account for Uncertainty . .	376
17.2.2 Opportunity Costs	377
17.3 Costs Concepts and Depreciation	378
17.3.1 Direct and Indirect Cost	379
17.3.2 Fixed and Variable Cost	380
17.3.3 Capital Expenditure and Operating Cost	381
17.4 Should You Care?	383
17.5 Further Reading	384
18 Predictive Analytics	385
18.1 Introduction	385
18.2 Some Classification Terminology	387
18.3 Algorithms for Classification	388
18.3.1 Instance-Based Classifiers and Nearest-Neighbor Methods	388
18.3.2 Bayesian Classifiers	389
18.3.3 Regression	391
18.3.4 Support Vector Machines	392
18.3.5 Decision Trees and Rule-Based Classifiers	394
18.3.6 Other Classifiers	396
18.4 The Process	397
18.4.1 Ensemble Methods: Bagging and Boosting	397
18.4.2 Estimating Prediction Error	398
18.4.3 Class Imbalance	398
18.5 The Secret Sauce	400
18.6 The Nature of Statistical Learning	401
18.7 Toolbox	403
18.8 Further Reading	407
19 Epilogue: Facts Are Not Reality	409
A Programming Environments for Science and Data Analysis	411
A.1 Overview	411
A.1.1 Scientific Software is Different	412
A.2 A Catalog of Scientific Software	413
A.2.1 Matlab	413
A.2.2 R	414
A.2.3 Python	414
A.2.4 What about Java?	417
A.2.5 Other Players	417

A.2.6	Recommendations	418
A.3	Writing your own	419
A.4	Further Reading	420
A.4.1	Matlab	420
A.4.2	R	420
A.4.3	NumPy/SciPy	421
B	Results from Calculus	423
B.1	Common Functions	424
B.1.1	Powers	424
B.1.2	Polynomials and Rational Functions	426
B.1.3	Exponential Function and Logarithm	428
B.1.4	Trigonometric Functions	430
B.1.5	Gaussian Function and Normal Distribution	431
B.1.6	Other Functions	432
B.1.7	The Inverse of a Function	435
B.2	Calculus	436
B.2.1	Derivatives	436
B.2.2	Finding Minima and Maxima	439
B.2.3	Integrals	440
B.2.4	Limits, Sequences, Series	441
B.2.5	Power Series and Taylor Expansion	442
B.3	Useful Tricks	444
B.3.1	The Binomial Theorem	444
B.3.2	The Linear Transformation	445
B.3.3	Dividing by Zero	446
B.4	Notation and Basic Math	448
B.4.1	On Reading Formulas	448
B.4.2	Elementary Algebra	449
B.4.3	Working with Fractions	450
B.4.4	Sets, Sequences, and Series	451
B.4.5	Special Symbols	452
B.4.6	The Greek Alphabet	453
B.5	On Learning Math	453
B.6	Further Reading	453
B.6.1	Calculus	453
B.6.2	Linear Algebra	454
B.6.3	For Further Study	454

Preface

Coming soon

Chapter 1

Introduction: Data and Data Analysis

This is a book on *data analysis*. To get started, let's introduce all *three* parts of that expression: what constitutes data? What do we mean by analysis? And: how do we get from one to the other?

1.1 Data

Individual bits of data usually come grouped in data sets. Both the atomic pieces, as well as the entire data sets can be of different type.

1.1.1 Types of Data

Individual bits of data can be of different type: numerical, string, whatever. There is a little bit of terminology associated with these distinctions, but also some more abstract concepts. Let's take a quick look.

In most books on data analysis, you will find data being classified as either *numerical* or *categorical*. Numerical data can be further subdivided into the *discrete* and *continuous* cases. Categorical data, which comprises all non-numeric data, such as {"red", "green", "blue"}, or {"Like", "Dislike", "Don't Care"}, is split into *ordinal* (sortable) and *nominal* (non-sortable) data. For instance "red", "green", "blue" would be nominal, since no meaningful sort order exists. On the other hand, {"Like", "Dislike", "Don't Care"} would be considered nominal, since there is a clear progression from "Like", over "Don't Care", to "Dislike".

Sortability is an important property, because if categorical data is sortable ("ordinal"), then it can be mapped to a set of numbers and therefore is in many ways *equivalent* to numerical data. To be sure, we can argue about the

specifics of the mapping (whether the difference between “Red” and “Yellow” be the same as between “Yellow” and “Green”, and so on), but there is no doubt that a mapping *can* be found. Also watch out that the mapping is correct: {“Like”, “Dislike”, “Don’t Care”} obeys the sort order “Dislike” < “Don’t Care” < “Like” (with the “Don’t Care” in the middle).

Another distinction for numerical data that you will find in the literature is the distinction between *interval* and *ratio* data. Interval data is data that does not have a proper origin, whereas ratio data does. Examples for interval data (without proper origin) are calendar dates and temperatures in Fahrenheit and Celsius units. You can subtract such data to form *intervals* (there are 7 days between 01Apr09 and 07Apr09) but you cannot form ratios: it does not make sense to say that 60 Celsius is “twice as hot” as 30 Celsius. In contrast, things like length or weight measurements are ratio data: 0 kilograms truly means “no mass”. We can define a unit (1 kilogram) and then form the *ratio* between some arbitrary mass (of for instance 4 kilograms) and the unit (1 kilogram) to find that the mass is four times as heavy as the unit.

The distinction between ratio and interval data is not very important in practice, because interval data occurs rarely (I can not think of other examples as the two given above), and can always be avoided through better encoding: the data is numeric by construction, hence a zero must exist, and an encoding can be found that measures magnitudes from this origin (the Kelvin scale for temperatures does exactly that).

Another property I look for determines whether data is “mixable”. Is it possible to form arbitrary “mixtures” of data points, such as (for example) the average: $(x + y)/2$ (where x and y are arbitrary data points)? Is the result of this operation a valid data point and does it, in some way, lie “between” x and y ? It is not enough to be able to just *combine* data points (such as concatenating two strings), but I must be able to combine *arbitrary* multiples of all data points. If I can do this, then the data is similar to points in space, and a lot of geometric intuition can be brought to bear. For example, I can define directions (all possible multiples like x , $2x$, $3x$, ...) and neighborhoods. (Technically, the data forms a vector space over the real numbers.)

It is very important to realize that these properties are *not* inherent in the data, but only arise out of its *context* or *semantics* (its meaning). The data points “Red”, “Yellow”, “Green” may be sortable (if they are status indicators), or may be entirely categorical (if they refer to the color of a sold item). Even numbers may be categorical — think postal (ZIP) codes! Furthermore, if you define a sort order, make sure it captures the semantics of the problem domain. We already mentioned the “Like”, “Dislike”, “Don’t Care” problem. On the other hand, it makes no sense to sort ZIP codes by their numeric value — but it may make sense to sort them according to the number of customers in each. Again: properties of the data are not inherent in the data, and therefore can not be deduced through the blind application of any “rules”. Instead, the properties arise out of the semantics of the data, which can only be obtained through insight into the problem domain in which the data originated.

Data by itself does not provide information. It is only if we have the data *together* with the context that defines its semantics that data becomes meaning. (This point is occasionally overlooked by people with an overly formalistic disposition.)

1.1.2 Types of Data Sets

The first question when approaching a new data set is often a variation of one of the following:

- What does the data look like?
- What does knowing one thing tell me about another?
- What the heck is going on?

As it turns out, these three questions allow us to differentiate between different types of data sets.

If our primary question is the first (“What does the data look like?”), chances are we are talking about a *univariate* data set. A univariate data set consists of measurements describing the same quantity for a number of comparable items. The weight of all people in a group — that’s a univariate data set. If the data is from a file or a database table, it will come from a single column. Chapter 2 will deal with univariate data.

If we are mostly concerned with the second question, we are almost certainly talking about a *bivariate* data set (two columns). Postage rates as a function of weight of the package are a good example of a bivariate data set: given the weight of the package, the post office or parcel service can tell us the exact postage required to ship the package. This is typical of such data sets: we would like to know how the values in one column depend on the values in the other column. We’ll study some general methods for dealing with bivariate data set in chapters 3 and in chapter 4 we will look at methods for the special case that we are tracking the change of some quantity over time.

The last question is typical for situations where there are more than two columns. In such cases, it is often not even clear what to look for. Imagine you have been given a file with last months sales for (say) a clothing retailer. There are almost a dozen columns: sales date, garment size, color, quantity sold, price, type of garment (shirt, pants, socks, whatever), store ID, name of sales assistant, and so on, and you have been asked to use the sales data to come up with a way to “improve our business”. Where do you even start? This is typical for many *multivariate* data analysis problems, and often the first step is to try to find some form of structure in the data set — for example by finding pairs of columns which depend strongly on each other (for example we may find that the store ID tells us a lot about the number of units sold, regardless of item type or price: that would be an interesting start). Multivariate problems are hard — we’ll introduce some graphical approaches in chapter 5, and more computationally intensive methods in chapter 13 on clustering and in particular in chapter 14, which is specifically about some pretty advanced ways to identify the most important relationships in multivariate data sets.

1.1.3 Data and “Big Data”

A distinction, completely different from the one made just a moment ago, is the one between small and large data sets. I consider a data set as “large” when its size starts to become comparable to main memory — say, anything in excess of 200MB, or equivalently 20 million records or more (at the time of this writing). Data sets that are “large” in this sense are cumbersome to work with — disk operations are no longer instantaneous, processing a file may take minutes, and one has to be mindful about storage and memory requirements.

It can be tempting to work with the largest data sets possible, but this does not necessarily bring about any tangible benefit. Instead, taking a sample (see below) to bring the data set to a more manageable size may be a better choice.

There is nothing shameful about working with “small” data sets — in fact, many (if not most) interesting data sets are “small”. Very large data sets are frequently either messy, or contain a lot of basically redundant information. It is often true that a smaller, but carefully collected data set contains more information of value than a huge data set with lots of noise. (And if you really want to get into small data sets, there is even an entire book filled with examples of them!¹)

In addition to regular small and large data sets, a new concept has started to emerge: *Big Data*. While a large data set as defined previously is one that no longer fits comfortably into main memory, Big Data deals with data sets that no longer fit comfortably on a *disk*, requiring compute clusters and the respective software and algorithms (in practice: map/reduce running on Hadoop).

It is a little early to say something definitive about Big Data, but the current trend strikes me as being something quite *different*: it is not just classical data analysis on a larger scale. The entire approach of classical data analysis and statistics was (and is) *inductive*. Given a part, make statements about the whole: from a sample, estimate parameters of the population; given an observation, develop a theory for the underlying system. By contrast, Big Data (at least as it currently is being used) is *incidental*: primarily concerned with individual data points. Given that *this specific* user liked *this specific* movie, which other *specific* movie might he like? This is a very different question than asking what movies are most liked by people in general!

I may be wrong, but most (if not all) applications of Big Data that I am aware of seem to be concerned with a form of generalized exhaustive search: given some starting parameters, find the those specific instances among all data points that form the closest match (for sufficiently general definitions of “parameter”, “instance”, and “match”).

Being mostly interested in the inductive aspects of data analysis, I won’t have much to say about Big Data in this book. It really is quite a different topic.

¹A *Handbook of Small Data Sets*. David J. Hand, Fergus Daly, K. McConway, D. Lunn, E. Ostrowski. Chapman & Hall/CRC. 1993.

1.2 From Data to Analysis

One of the uncomfortable (and easily overlooked) truths of working with data is that usually only a small fraction of the time is spent on the actual “analysis”. Often a far greater amount of time and effort is expended on a variety of tasks that may appear “menial” by comparison, but are absolutely critical nevertheless: obtaining the data; verifying, cleaning and possibly reformatting it; dealing with updates, storage, and archiving. For somebody new to working with data (and even, periodically, for somebody not-as-new), it typically comes as a surprise that this kind of preparatory work is not only necessary, but also takes up as much time as it does.

By their nature, these housekeeping and auxiliary tasks tend to be very specific: specific to the data, specific to the environment, and even specific to the particular question that is being investigated. Consequentially, there is little that can be said about them in generality — it pretty much all comes down to ad-hoc hackery. Of course, this absence of non-trivial, recognizable techniques is one of the main reasons these activities receive as little awareness as they do.

That being said, we can try and draw attention to some of the issues that typically arise in practical situations.

1.2.1 Sources for Data

The two most common sources for data in an enterprise environment are *databases* and *logfiles*. As data sources, the two of them tend to address different needs. Databases will contain data related to the “business”, whereas logfiles will be a source for “operational” data: databases answer the question “what did we sell to whom?”, logfiles answer the questions “what did we do, and when?”.

Databases can be either “online transaction processing” (OLTP) or “production” databases, or “data warehouses” for long-term storage. Production databases tend to be normalized, fast, and busy. You may or may not be able to get read access to them for ad-hoc queries, depending on company policy. Data warehouses tend to be denormalized, slow, and often accessed through a batch processing facility (submit your query tonight to find out that you left out a field you needed tomorrow). Production databases tend to be owned (at least in spirit) by the application development teams. Data warehouses are invariably owned by the IT department. (We’ll have a bit more to say about this topic in chapter 16.) In either case, databases tend to form a stable foundation for data needs — provided you are interested in something the company already considers part of its “business”.

By contrast, logfiles are often an important source of data for new initiatives: if you try to evaluate a new business idea, chances are that the data required for your analysis will not be available in the database — not *yet*: there has never been a reason to store it before. In such situations you may find that you can nevertheless find the information you need in logfiles that are regularly produced.

One *very* important distinction is that databases and logfiles have different lifecycles: making changes to the design of a database is always a slow (often excruciatingly slow) process, but the data itself lives in the database forever (if it is properly designed). By contrast, logfiles often contain much more information than the databases, but they are usually removed very quickly — if an organization keeps logfiles for two weeks, consider yourself lucky!

Consequentially, if you want to do a project with the data contained in logfiles you need to move *fast*: start saving all files to your desktop or another safe location immediately, *then* figure out what you want to do with them! Frequently, you will need several weeks (or months) worth of data for a conclusive analysis, and any day that you wait can never be made. Also keep in mind that logfiles are usually generated on production servers to which access may be heavily restricted. It is not uncommon to spend *weeks* in negotiations with network administrators if you need to move significant amounts of data off of production systems.

The same consideration applies if information is not available in the logfiles, so that existing code needs to be instrumented to support collection of the required data. In this situation, you will likely find yourself captive to pre-existing release schedules and other constraints. Again: start to think about *collecting* data early.

Because databases and logfiles are so common and so directly useful sources of data in an enterprise environment, it is easy to forget that they are not the only ones there are!

A separate source of data, which at times can be extremely useful, comes from the company's finance department: companies are required to report on various financial metrics. This means that this kind of information *must* be available, although possibly only in a highly aggregated form (quarterly), and possibly quite late. On the other hand, this information is normative and therefore reliable: after all, that's what the company is paying taxes on! (I am ignoring the possibility that the data provided by the finance department might be *wrong*, but don't get me wrong: forensic data analysis is also a very interesting field of study.)

What works internally may also work with competitors: the quarterly filings that publicly listed companies are required to make can contain highly relevant information, also!

1.2.2 Cleaning and Conditioning

Raw data, whether it was obtained from a database query or by parsing a logfile, typically needs to be cleaned or conditioned. Here are some areas that often need attention:

Missing Values If individual attributes or entire data points are missing, we need to decide how to handle them: discard the whole record, mark the information in question as missing, or backfill it in some way?

Outliers In general, you should be very careful when removing outliers — you may be removing the effect that you are looking for, and *never* should data points be removed silently. (There is a story that the discovery of the hole in the Ozone layer over Antarctica was delayed by several years, because the automated data gathering system kept discarding readings that it considered to be “impossibly low”.)

Junk Data that comes over a network may contain non-printable characters or similar junk. They are not only useless, but can also seriously confuse downstream applications trying to process the data (for instance if the non-printable characters are interpreted as control characters — many programming environments will not issue helpful diagnostics if this happens!). This kind of problem frequently goes unnoticed, because such junk is often rare and not easily noticed when just scanning the beginning of a data set.

Formatting and Normalizing Individual values may not be formatted in the most useful way for subsequent analysis. Examples of frequently used transformations of this kind include: forcing upper- or lower-case, removing blanks within strings or replacing them with dashes, replacing timestamps with Unix Epoch seconds, the Julian Day number, or a similar numerical value, replacing numeric codes with string labels or vice versa, and so on.

Dedupe Recognize and remove duplicate records. Depending on what you consider “duplicate”, this may be a non-trivial effort. (I once worked on a project that tried to recognize misspelled postal addresses and assign them to the “correctly” spelled one. That also is a form of deduping.)

Merge Having to merge data sets from different sources is pretty common, for instance if the data comes from different database instances. Make sure the data is truly compatible, in particular if the database instances are geographically disperse. Differing timezones are a common trouble spot, but also think of monetary units. In addition, you may have to be aware of localization issues, such as font encodings and date formatting.

(On timezones: I used to be a strong proponent of keeping all date/time information in UTC, always. But in the meantime, I learned that this is not always appropriate — for some information, such as customer behavior, it is the *local* time that matters, not the absolute time. Whether we can actually determine the local time at the actual user’s location is a different matter!)

As you read this list, you should have been realizing that the process of *cleaning* data cannot be separated from *analyzing* it. For instance, outlier detection and evaluation requires a pretty deep analysis to be reliable. On the other hand, you may have to remove outliers before you can calculate meaningful values for certain summary statistics. This is an important insight, which we

will see again and again: data analysis is an *iterative* process, with each operation being at the same time the result of a previous, and the preparation for a successive step.

Data files may also be defective in ways that only become apparent when subsequent analysis fails or gives nonsensical results. Some common problems are:

Clerical Errors Basically data entry errors: 0.01 instead of 0.001, values entered into the wrong column, all that. Since most data these days is computer-generated, the classic occasional typo seems to mostly be a thing of the past. But watch out for its industrial counterpart: whole data sets that are corrupted systematically. (On one project I worked on, we did not realize that a certain string field in the database was of fixed width. As we went from entries of the form ID1, ID2, and so on to entries like ID10, the last character was silently truncated by the database. It took a long time before we noticed — after all, the results we got back *looked* alright...)

Numerical “Special” Values Missing values in a data set may be encoded using special numerical values (such as -1 or 9999). Unless these values are filtered out, they will obviously corrupt any statistical analysis. There is less of a need for this sort of thing when you are using text files (because you can indicate missing values using a marker like ???), but be aware that it still is an issue when you are dealing with binary files.

Crazy Business Rules and Overloaded Database Fields Bad schema design can thoroughly wreck your analysis. A particular pernicious problem are overloaded database fields: fields that change their meaning depending on the values of *other* fields in the database. I have witnessed a situation where the *Quantity* field on a table contained the number of items shipped — unless it was zero (in which case it signaled a discount, a promotion, or an out-of-stock situation, depending on whether an entry with the same order ID existed in the *Discounts*, *Promotions*, or *BackOrders* tables), or it contained not the number of items shipped, but the number of multi-item packages that had been shipped (if the *IsMulti* flag was set), or it contained the ID (!) of the return order associated with this line item (if some other flag was set). Confusingly, running a query such as `select avg(Quantity) from ...` would still produce a number that *seemed* sensible. Worse, most people were unaware of this situation — data was usually accessed only through (massive!) stored procedures, which took all these crazy business rules into account.

Blanks Masking for Columns This is a tiny problem by comparison, but it can be really annoying and hard to detect: if you are using whitespace separated text files, then empty or blank strings cannot be recognized, instead the values from the following columns will be substituted. (This is an argument for using comma-separated, rather than whitespace-separated, files.)

1.2.3 Sampling

When dealing with very large data sets we can often simplify our lives significantly by working with a *sample*, rather than with the full data set, provided the sample is *representative* of the whole. And therein lies the problem.

In practice, sampling often means partitioning the data on some property of the data: picking all customers whose name begins with the letter “i” for instance, or whose customer ID ends with “0”. Or using the logfile from one server only (out of 10). Or all transactions that occurred today. The problem is that it can be very difficult to know *a priori* whether these subpopulations are in any way representative for the rest of the population. And to determine whether they are would require us to do an in-depth study on the *whole* population — precisely what we wanted to avoid!

Statistical lore is full of often quite amusing stories about the subtle biases introduced through improper sampling: choosing all customers whose first name ends in “a” will probably introduce a bias towards female customers. Surveying children for the number of siblings will overestimate the number of children per household, because it neglects to include households without children. A long-time study of mutual funds may report overly optimistic returns on investment, because it ignores funds that have been shut down because of poor performance (“survivorship bias”). A trailing zero may indicate a customer record that was created long ago, by the previous version of the software. The server that you picked for your logfile may be the “overflow” server which only comes online during peak hours... And that does not even mention problems involved in collecting data in the first place! (A phone survey is inherently biased against people who don’t have a phone or don’t answer it.) It is also not uncommon that strange biases exist that nobody is aware of (it is not guaranteed that the network administrators will know or understand the algorithm that the loadbalancer uses to assign transactions to servers, in particular if the loadbalancer itself is “smart” and changes its logic based on traffic patterns).

A relatively safe way to create a sample is to take the whole data set (or as large a chunk of it as possible) and randomly pick some of the records. The keyword is *randomly*: don’t take every tenth record; instead evaluate each record and retain it with probability 1/10. Also make sure that the data set does not contain duplicates. (For instance, to sample customers given their purchases, you must first extract the customer IDs and dedupe them, then sample from the deduped IDs. Sampling from the transactions alone will introduce a bias towards repeat customers.)

In any case, you should verify that your sampling process does lead to representative samples. (Take two independent samples, and compare their properties.)

Sampling can be truly useful, even necessary. Just be very careful.

1.2.4 Data File Formats

When it comes to file formats for data, my recommendation is to keep it simple, even dead-simple. The simpler the file format, the greater flexibility you do have in regards to the tools you can use on the data. Avoid formats that require a non-trivial parser!

My personal favorite is that old standby: the delimiter-separated text files, with one record per line, and a single data set per file. (Despite the infamous difficulties with the Unix `make` utility, I nevertheless like tab-delimited files: since numbers don't contain tabs, I never need to quote or escape anything, and the tabs make it easy to visually inspect a file — easier than commas do.) In fairness, delimiter-separated text files do not work well for one-to-many relationships or other situations where each record can have a varying number of attributes. On the other hand, such situations are rare, and tend to require special treatment, anyway.

One disadvantage of this format is that it does not allow you to keep information about the data (“metadata”) within the file itself, except possibly the column names as first row. One possible solution is to use two files: one for the data, and one for the metadata, and to adopt a convenient naming convention (such as using the same basename for both files, and to distinguish them by the extensions `.data` and `.names`).²

In general, I strongly recommend that you stay with text files, and avoid binary files. Text files are portable (despite the annoying newline issue), robust, self-explanatory, and compress nicely. If you nevertheless decide to use binary files, I suggest that you try and use an established format (for which mature libraries exist!), instead of rolling your own.

I also don't find XML very suitable as a file format for data: the ratio of markup to payload is very poor, leading to unnecessarily bloated files. XML is also notoriously expensive to parse, in particular for large files. Finally, the flexibility that is provided by XML is rarely necessary for data sets, which typically have a very regular structure. (It may seem as if XML might be useful for metadata, but even here I disagree: the value of XML is to make data machine-readable, whereas the primary consumers of metadata are humans!)

Everything I have said so far assumes that the data files are primarily for yourself (you don't want to distribute them) and that you are willing to slurp in the entire file sequentially (so that you don't need to perform seeks within the file). There are file formats that allow you to bundle multiple data sets into a single file, and efficiently extract parts of them (for example, check out the Hierarchical Data Format (HDF) and its variants), but I have never encountered them in real life — it should not be lost on you that the Statistics and Machine Learning communities use delimiter-separated text almost exclusively as format for data sets on their public data repositories. (And if you need indexed lookup, you may be better off setting up a minimal standalone database for yourself: see the workshop in chapter 16.)

²This convention is used by many data sets available from the UCI Machine Learning Repository.

Lastly I should point out that some (scientific) disciplines have their own, specialized file formats, and the tools designed to handle them.

1.2.5 Care and Feeding of Your Data Zoo

If you work in the same environment for a while, you are likely to develop a veritable collection of different data sets. Not infrequently, it is this ready access to relevant data sets which makes you valuable to the organization (quite outside your more celebrated skills)! On the down-side, *maintaining* that collection in good order requires a certain amount of effort.

My primary advice is make sure all data sets are *self-explanatory* and *reproducible*.

To ensure that a data set is self-explanatory does not just mean to include some metadata with or in the file itself, but also to include all the information that is necessary to make sense of it. For instance, to represent a time series (that is a data set of measurements taken over time at regular intervals), it is strictly only necessary to store the values, the starting time, and the length of the interval between data points. However, it is safer to store the corresponding timestamp with each measured value — this way, the data set still makes sense, even if the metadata has been lost or garbled! Similar considerations apply more generally: I tend to be pretty generous when it comes to including information that might seem “redundant”.

To keep data reproducible, you should keep track of its source *and* the cleaning and conditioning transformations. At least in my experience, this can be tedious, because so much of the latter consists of ad-hoc, manual operations. These days, I usually keep log with my data sets, noting the URLs (if the data came from the Web), or the database queries. I also note the commands and pipelines issued at the shell-prompt, and keep copies of all transformation scripts. Finally, if I combine data from multiple sources into a single data set, I always retain the original data sets as well.

This kind of housekeeping is very important: not only to produce an audit trail (should it ever be needed), but also because data sets tend to be re-used again and again and for different purposes. Being able to determine *exactly* what is in the data is crucial.

I have not found lots of opportunities to automate many of these processes: the tasks just vary too much. The one exception is the automated, scheduled collection and archiving of volatile data (for instance to copy logfiles to a safe location). Your needs may be different.

Lastly, three pieces of advise on the physical handling of data files. They should be obvious, but aren't necessarily:

- Keep data files readily available. Being able to run a minimal script on a file residing on a local drive to come up with an answer in seconds (compared to the 12- or 24-hour turnaround typical of many data warehouse installations) is a huge enabler.

- Compress your data files. I once worked with a group of statisticians who constantly complained about the lack of disk space and kept requesting more storage. None of them used compression or had even heard of it. And all their data sets were kept in a text-like format that compressed by 90 percent! (Also keep in mind that `gzip` can read from and write to a pipe, so that the uncompressed file never needs to exist on disk.)
- Have a backup strategy, in particular if all of your data resides on your local workstation only. At the very least, get a second drive and mirror files to it. Of course, a remote (and, ideally, managed) storage location is much better. Keep in mind that data sets can easily get large, so you might want to sit down with your network administrators early to allow them to budget your storage needs appropriately.

1.2.6 Skills

I hope I have been able to convince you that obtaining, preparing, and transforming data makes up a large part of the day-to-day activities when working with data. To be *effective* in this role, I recommend you acquire and develop some skills that facilitate these parts of your role.

Mostly, they come down to easy, ad-hoc programming. If you come from software development, you will hardly find anything new here. But if you come from a scientific (or academic) background, you might want to broaden your expertise a little.

A special consideration is due to those who come to “data analysis” from a database-centric, SQL programmer point of view. If this describes your situation, I would *strongly* encourage you to pick up a language besides SQL. SQL is simply too restricted in what it can do, and therefore limits the kinds of problems you will choose to tackle — whether you realize it or not! Additionally, it is a good idea to be able to do the majority of your work “offline”, without taking a toll database (which is, after all, a shared resource).

Learn a scripting language A scripting language such as Perl or Python is required for easy manipulation of data files. Knowledge of a “large-scale” programming language like C/C++/Java/C# is *not* sufficient. Scripting languages eliminate the overhead (“boilerplate code”) typically associated with common tasks such as input/output, file and string handling. This matters, since most data transformation tasks are tiny, and therefore the typical cost of overhead, relative to the overall programming task, is simply not acceptable.

R (the statistics package) can do double-duty as a scripting language for these purposes.

Master Regular Expressions If you are dealing with strings (or string-like objects, such as timestamps), regular expressions are the solution (and an amazingly powerful solution) to problems you didn’t even realize you

had! You don't need to develop intimate familiarity with the whole regular expression bestiary, but working knowledge of the basics is required.

Be comfortable browsing a database Pick a graphical database frontend³ and become proficient with it. You should be able to figure out the schema of a database and the semantics of the data simply by browsing the tables and their values, with minimal help.

Develop a good relationship with your system administrator and DBA System administrators and DBAs are in the position to make your life significantly easier (by granting you access, creating accounts, saving files, providing storage, running jobs for you, ...). However, they have not been hired to do that — quite the contrary, they are being paid to "keep the trains on time" and a rogue (and possibly clueless or oblivious) data analyst, running huge batch jobs during the busiest time of the day, does *not* help with that!

I would like to encourage you to take an interest in the situation of your system administrators: try to understand their situation and the constraints that they have to work under. System administrators tend to be paranoid — that's what they are being paid for! Their biggest fear is that *something* will upset the system. If you can convince them that you are not posing a great risk, you will probably find them to be incredible helpful.

(Lastly, I tend to adopt the attitude that any production job by default has higher priority than the research and analysis that I am working on, and therefore better be patient. On the other hand, if I feel actively sandbagged, I can also get quite drastic, as necessary.)

Work on Unix I mean it. Unix was developed for *precisely* this kind of ad-hoc programming with files and data, and continues to provide the most liberating environment for such work.

Unix has some obvious technical advantages, but its most important property in the present context is that it *encourages you to devise solutions*. It does not try (or pretend) to do the job for you, but it goes out of its way to give you tools that you might find handy, without prescribing in which way and for what you are going to use them. By contrast, other operating systems tend to encourage you to stay within the boundaries of certain familiar activity patterns — which does *not* encourage the development of your problem solving abilities (and more importantly your problem solving *attitudes*).

True story: I needed to send a file containing several millions of keys to a coworker. (The company did not work on Unix.) Since the file was too

³The SQuirreL project is a good choice. Free, open-source, and mature, it is also written in Java, meaning that it can run anywhere and connect to any database for which JDBC drivers exist. Home page: <http://squirrel-sql.sourceforge.net>.

large to fit safely into an email message, I posted it to a web server on my desktop and sent my coworker the link. (I dutifully had provided the file with the extension `.txt`, so that he would be able to open it.) Five minutes later, he calls me back: “I can’t open that” — “What do you mean?” — “Well, I click the link, but XXX (the default text editor for small text files on this particular system) dies because the file is too big.” This coworker was not inept (in fact, he was quite good at his primary job), but he displayed the particular acquired non-problem-solving ability that develops in predefined work environments: “link, click”. It did not even occur to him to think of something else to do. That’s a problem!

If you want to be successful working with data, you want to work in an environment that encourages you to devise your own solutions.

You want to work on Unix.

1.3 Analysis

We can distinguish a few different forms or phases of data analysis.

The first step when dealing with a new data set needs to be exploratory in nature: what actually is in the data set? Statisticians refer to this phase as *initial* or *exploratory* data analysis.

To begin with, you want to gather many different, but tiny, pieces of information about the data: start with the number of rows and columns! A visual inspection of the entire data set is usually astoundingly illuminating — every time I skip it, thinking that I know what the data looks like, I regret it.

At this point, one can also calculate some basic summary statistics (such as mean and standard deviation), but I recommend that you don’t spend too much time on it: summary statistics tend to give only a very narrow, “keyhole” view of the data, which is at best limiting and very often misleading.

Instead, you should quickly move to explore the data set *graphically*. One of the most interesting and important questions concerns the distribution of points for all quantities in the data set: what is the shape of the distribution? Is there a central tendency; are there outliers? The answers to these questions are important, because they will strongly influence what methods we will use later for more in-depth studies. In chapter 2 we will learn about graphical methods to study the shape of a distribution.

Depending on the nature of the data set, we may want to proceed to different plots: scatter plots for bivariate data sets, or possibly a scatter plot matrix if the data set is multivariate (see chapters 3 and 5). If the data comes from a time series, you want to do a time plot (chapter 4).

In all of the above I have assumed that you know where the data is coming from, what it means, how it was collected, and how it is encoded. (If not, you really should go find out before doing anything else!) Nevertheless, it is likely that the initial analysis up to this point will show up some surprises, such as inexplicable outliers, mysteriously missing data, or unknown encodings.

That's good: learning where the data is unreliable, messy, or its meaning is unknown is an important part of the exploration phase!

But the main purpose of the initial or exploratory phase of analysis is to be able to *describe* the data set (its structure, its quality, its most basic properties), and based on that description to recommend the most *suitable methods* for further study.

For many practical problems the initial analysis up to this point is all that is really required! A basic description is often all that anybody wants. In particular, some well-conceived graphs frequently provide all the insight that is likely to be useful. Also be aware that you may have found that the data is of such poor quality that further investigation is not really warranted. (Alternatively, your investigation into the way the data was collected may have revealed such fundamental flaws in the process to invalidate the whole approach!)

On the other hand, if there is both a need and an opportunity for further work, now is the time to become clear on the *purpose* of the analysis, and on the *use* that is going to be made of your results. At first sight, it might seem surprising to wait this long to discuss the actual purpose of your efforts, but in fact it is only natural: only now, after you had a chance to see what the data looks like, can you have an informed, intelligent conversation about possible directions.

What the next steps will look like depends on the particular situation, of course. Statisticians will at this point typically settle on a (statistical) model and proceed to estimate its parameters. Alternatively, you may want continue with further exploratory work, but using more sophisticated, and often computationally intensive methods, to investigate the structure of the data even further. In part III we will investigate clustering or dimensionality reduction methods as examples of this form of analysis. (I will not formally introduce classical statistical methods in this book, but in chapter 10 I will give an overview of the strengths and limitations of that particular approach.)

A different kind of goal is to use the data to build causal models: models that purports to *explain* the data in terms of underlying causes. This is basically the (inductive) scientific method: starting with some observations (the data), we try to recognize patterns. Based on these, we develop a theory which is able to explain the observations that we started with. I'll give some hints about this kind of model building in part II.

Yet a different form of data analysis exists in some particularly well-established fields. In these situations, the environment out of which the data arises is fully understood (or at least believed to be understood), and the methods and models to use are likewise accepted well-known. (Clinical trials come to mind.) The purpose of an "analysis" in these cases is not to find out anything "new", but to determine the model parameters with the highest degree of accuracy and precision, for each new set of data points that is being generated. Since this is the kind of work where details matter, it should be left to specialists, and I won't have much (if anything) to say about it in this book.

Finally, I assume that most readers will, like me, work in a commercial (enterprise) environment, and we should therefore make sure that our results are

useful in that situation. In the last part (part IV) I will therefore discuss in some detail some typical applications or consumers of data analysis. Understanding the needs of your clients is the best way to make sure your results will end up being used!

1.4 My Philosophy

This is a rather personal book. Although I have tried to be reasonably comprehensive, I have selected those topics that I consider to be relevant and useful in practice, whether they are part of the “canon” or not. I have also included several topics that you won’t find in any other book on data analysis. Although not new or original, they are usually not used or discussed in this particular context — yet I find them indispensable.

You will also notice that throughout this book I express specific, explicit advice, opinions, and assessments. These are reflections of my personal interest, experience, and understanding. I do not claim that my point of view is necessarily correct: I will leave it to you to evaluate what I say for yourself and to adapt it for your needs. However, in my view, a specific, well-argued opinion is of greater use than a sterile laundry list of possible algorithms — even if you later decide to disagree with me. The value is not in the opinion, but in the arguments leading up to it. If yours are better than mine, or even just more agreeable to you, I will have achieved my purpose!

To give you a better sense for the position that I am coming from let me state a few maxims, which I hold to be self-evident:

- Simple is better than complex.
- Cheap is better than expensive.
- Manual is better than black-box.

While most people would probably agree with them (or at least *claim* to agree with them), the following may be a bit more controversial:

- Purpose is more important than process.
- Insight is more important than precision.

I will admit that they don’t *always* apply, in particular the latter: there certainly are scenarios where precision is all that matters. But these also tend to be the situations where not much insight is to be gained any more: the circumstances are fully understood, and now it is only a question of accurate execution.

That’s not the kind of situation that I tend to work in, nor the situation that I expect my typical reader to face. My quintessential assignment is something like: “Here are 20 GB of logfiles — find a way to improve our business!” If that describes your situation, then insight is indeed more important than precision.

As you read this book and start to work with data, I’d like to offer one final piece of advice:

Think more, work less.

1.5 Workshop: Four Unix Tools

I will admit that it feels a little silly to write about Unix command-line tools — like giving advice on the proper use of flint-stone tools. I do it regardless, for two reasons: the first one is that many of them seem to be less well known than one would expect. And there are always some incredibly useful, but virtually unknown options yet to discover!

The second, and more important, reason is that Unix command-line tools have about the lowest overhead possible. You don't need to start a (separate) program, you don't need to open an editor, you don't need to save a file...all you need is a couple of key strokes and you are done. Or at least: further along.

That may seem like a small, or even irrelevant point, but in my experience that is not so. Barriers to entry matter, and with exploratory or creative work more than for other activities. After all, the purpose of a data analysis project is not to write software, but to analyze data, hence all activities related to "writing software", which on a software project would be considered part of the job, are in effect wasted motion.

And I have observed again and again that people whose tools require more "ceremony" get less work done — no matter how much more "powerful" their tools are. If the ceremony gets in the way of the tool being used, the additional "power" is a moot point, anyway!

So, in this spirit, let's take a quick look at four totally and embarrassingly obsolete, yet totally indispensable...hand axes.

1.5.1 awk

The Unix `awk` utility is a complete little programming language by itself. It predates Perl — in fact, Perl took some of its early inspirations from it. Although it is no longer stylish (or in fact advisable) to write entire scripts in `awk`, since Perl, Python, and so on are so much more suitable for non-trivial programming tasks, it nevertheless *can* be done. (As it turns out, `awk` is part of the POSIX specification, and you can therefore rely on it being there on any POSIX-compliant platform!) Instead, `awk` "programs" are usually entered as part of the invocation of the `awk` command itself.

Basically, `awk` automatically iterates over all lines in a file and splits them into fields (by default on whitespace). The field values are automatically assigned to variables of the form `$1`, `$2`, ... and can be used in the body of the `awk` program. For instance, the following command line would read the file named `data` and print its first column, as well as the sum of the second and third columns:

```
> awk '{print $1, $2+$3}' data
```

The block inside the curly braces is the body of the awk program and is evaluated for every line in the input file (or files) named at the end of the command line. Notice that the body must be quoted to protect it from interpretation by the shell.

Besides a block that is executed for every line of input, it is also possible to have blocks that are executed before or after any input files are read. The following command line will generate ten random numbers between zero and nine and quit:

```
awk 'BEGIN { for(i=0; i<10; i++) { print int( 10*rand() ) } }'
```

The BEGIN block may also be useful for initializations: for example if we want to read a comma-separated file, we need to set the variable FS ("field separator") to the appropriate value before we begin reading from the file. In the following example, we calculate the total sum and the average value of all values from the first column of a comma-separated file:

```
awk 'BEGIN {FS=","} {a+=$1} END {print a, a/NR}' data.csv
```

Notice that there are *three* blocks: only the middle one is evaluated for each line of the input file, the other two are evaluated before and after handling reading the data. In this example, the middle block does not create any output: it only accumulates the grand total in the variable a. Once the file has been read completely, results are written to standard output. (The variable NR holds the number of records processed.) I actually use the snippet above (or a variation of it) quite frequently to calculate basic summary statistics for an unknown file.

The most exciting feature when awk was first developed was its support for *associative arrays* (what today we call hashmaps or dictionaries). The following snippet counts how often each string in column 1 occurs in a file:

```
awk '{a[$1]++} END {for( i in a ) { print i, a[i] } }' data
```

This is another extremely useful little idiom. You can see that with a little tweaking it can be used to generate histograms and even cumulative distribution functions on the quick. This is another one of the awk applications that I use all the time. (See chapter 2 for more information on histograms and cumulative distribution functions.)

Finally, awk also supports pattern matching using regular expressions. The pattern to match is written *before* the block containing the action to be performed for each match. The following program calculates the average of column two, but only for those records that match a given string:

```
awk '$1 ~ /smith/ {a+=$2; n+=1} END { print a/n }' data
```

Notice that we needed to keep track of the number of records which matched the pattern (in the variable n) to calculate the correct value for the average: the standard variable NR counts *all* records read, whether they matched the pattern or not.

1.5.2 sort

The `sort` command sorts files, and writes the sorted lines to standard output. What is remarkable is how *quickly* it does so, even for exceptionally large files. You can use `sort` on files that you would not sort (in memory) using Perl or Python.

By default, `sort` sorts alphabetically (using the sort order required by the `LC_COLLATE` environment variable). Using the `-n` option forces numerical (integer) sorting, to recognize numbers in scientific notation requires the `-g` option instead. The `-r` option reverses the sort order (largest first on output).

The field (or fields) on which to sort are specified using the `k` option. By default, `sort` splits each line in the input file on whitespace (a different separator can be specified using `-t`), and the number of the column to sort is passed as argument to the `-k` option. In other words, the following sorts alphabetically on the first column:

```
sort -k 1 data
```

The `-k` option can be given multiple times, to specify which columns should be used to break ties. The following command line sorts a comma-separated file, first alphabetically on the first column, then reverse numerically on the second column in case of ties:

```
sort -t , -k1 -nrk2 data.csv
```

It is also possible to extract only parts of a column for sorting purposes, and even to do a calendar sort, based on the names of the months. But most of all, `sort` can be unexpectedly helpful when dealing with files that are so large that they are uncomfortable any other way.

1.5.3 paste

While `awk` can be used to extract columns from files, `paste` does the opposite: it takes a list of files, reads a line from each, and glues them next to each other, separated by tab characters (or whatever you have specified using the `-d` option). If one of the files is shorter than the others, `paste` supplies the empty (not blank!) string instead.

Notice that `paste` does not perform a merge operation: it does not attempt to find corresponding records in its input files, so you probably want to make sure your files are sorted before passing them to `paste`.

1.5.4 gzip

Finally, the `gzip` and `gunzip` programs compress and uncompress files. They don't bundle multiple files into single archives: `gzip` and `gunzip` just compress and uncompress individual files. (Use `tar` for archiving.)

These programs are trivial to use, but I want to point out two options that can be particularly helpful: with the `-c` option, `gunzip` writes the uncompressed output to standard output, instead of to disk. Similarly, `gzip` can read from standard input. This makes it possible to process very large files without ever having to set aside disk space for them. The following pipeline is typical:

```
gunzip -c data1.gz | awk '{print $1, $2+$3}' | gzip > data2.gz
```

The other option I wanted to mention is `-r`, which forces `gzip` into recursive mode: starting at the current directory, it descends into every subdirectory recursively, compressing all uncompressed files it finds there. (Similarly for `gunzip`.) Handy for compacting entire data set collections with a single command!

1.6 Further Reading

- *Problem Solving: A statistician's guide*. Chris Chatfield. 2nd ed., Chapman & Hall/CRC. 1995.
This is a highly informative book about all the messy realities that are usually *not* mentioned in class: from botched experimental set-ups to effective communication with the public. The book is geared towards professional statisticians, and some of the technical discussion may be too advanced, but it is worthwhile for the practicality of its general advice nonetheless.
- *Unix Power Tools*. Shelley Powers, Jerry Peek, Tim O'Reilly, Mike Loukides. 3rd ed., O'Reilly. 2002.
The classic book on getting stuff done with Unix.
- *The Art of UNIX Programming*. Eric S. Raymond. Addison-Wesley. 2003.
The Unix philosophy has been expounded many times before, but rarely more eloquently. This is a partisan book, and one does not have to agree with every argument the author makes, but some of his observations on good design and the desirable features in a programming environment are particularly worth contemplating.

Part I

Graphics: Looking at Data

Chapter 2

A Single Variable: Univariate Data

When dealing with univariate data, we are usually mostly concerned with the overall *shape* of the distribution. Some of the initial questions we may ask include:

- Where are the data points located and how far do they spread? What are typical, as well as minimal or maximal values?
- How are the points distributed? Are they spread out evenly or do they cluster in certain areas?
- How many points are there? Is this a large data set or a relatively small one?
- Is the distribution symmetric or asymmetric? In other words, is the tail of the distribution much larger on one side compared to the other?
- Are the tails of the distribution relatively heavy, meaning that many data points lie far away from the central group of points, or are most of the points — with the possible exception of individual outliers — confined to a restricted region?
- If there are clusters: how many are there? Is there only one or are there several? Approximately where are the clusters located and how large are they — both in terms of spread and in terms of the number of data points belonging to each cluster?
- Are the clusters possibly superimposed on some form of unstructured background, or is the entire data set composed only out of the clustered data points?
- Does the data set contain any significant outliers, that is data points that seem to be different from all the others?

- And lastly, we may take note of any other unusual or significant feature in the data set — gaps, sharp cut-offs, unusual values, anything at all that we can observe.

As you can see, even a simple, single-column data set can contain a lot of different features!

To make this concrete, let's look at two examples. The first concerns a relatively small data set, namely the number of months that the various American presidents have spent in office. The second one is much larger and stems from an application domain that may be more familiar: we will be looking at the response times from a web server.

2.1 Dot and Jitter Plot

Imagine you are given the data set shown in full in example 2.1, showing all past American presidents, and the number of months each of them has spent in office.¹ Although this data set has three columns, we can treat it as univariate since we are only interested in the times spent in office — the names don't matter to us (at this point). What can we say about the typical tenure?

This is not a large data set (just over 40 records), yet it is a little too big to take in as a whole. A very simple way to get an initial sense of the data set is to create a *dot plot*. In a dot plot, we plot all points on a single (typically horizontal) line, letting the value of each data point determine the position along the horizontal axis. (See the top part of figure 2.1.)

A dot plot can be perfectly sufficient for a small data set such as this one. However, in our case it is slightly misleading, because whenever a certain tenure occurs more than once in the data set, the corresponding data points fall right on top of each other, making it impossible to distinguish them. This is a frequent problem, in particular if the data assumes only integer values or is otherwise "coarse-grained". A common remedy is to shift each point by a small random amount from its original position; this is called *jittering* and the resulting plot is a *jitter plot*. A jitter plot of the data set is shown in the bottom part of figure 2.1.

What does the jitter plot tell us about the data set? We see two values where data points seem to cluster, indicating that these values occur more frequently than others: not surprisingly, they are located at 48 and 96 months, corresponding to one or two full four-year terms in office. What may be a little surprising, however, is the relatively large number of points which occur *outside* these clusters. Apparently, quite a few presidents left office at irregular intervals! Even in this simple example, a plot reveals both something expected (the clusters at 48 and 96 months) and the unexpected (the larger number of points outside those clusters).

¹The inspiration for this example comes from a paper by Robert W. Hayden in the Journal of Statistics Education. The full text is available at www.amstat.org/publications/jse/v13n1/datasets.hayden.html.

1	Washington	94
2	Adams	48
3	Jefferson	96
4	Madison	96
5	Monroe	96
6	Adams	48
7	Jackson	96
8	Van Buren	48
9	Harrison	1
10	Tyler	47
11	Polk	48
12	Taylor	16
13	Filmore	32
14	Pierce	48
15	Buchanan	48
16	Lincoln	49
17	Johnson	47
18	Grant	96
19	Hayes	48
20	Garfield	7
21	Arthur	41
22	Cleveland	48
23	Harrison	48
24	Cleveland	48
25	McKinley	54
26	Roosevelt	90
27	Taft	48
28	Wilson	96
29	Harding	29
30	Coolidge	67
31	Hoover	48
32	Roosevelt	146
33	Truman	92
34	Eisenhower	96
35	Kennedy	34
36	Johnson	62
37	Nixon	67
38	Ford	29
39	Carter	48
40	Reagan	96
41	Bush	48
42	Clinton	96
43	Bush	96

Example 2.1: All the past US presidents, and the number of months they spent in office.

Before moving on to the second example, let me point out a few additional technical details regarding jitter plots:

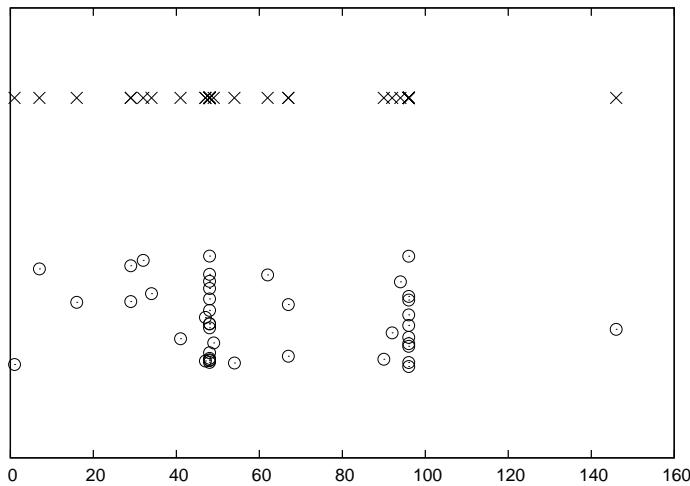


Figure 2.1: Dot and jitter plots of the data set in listing 2.1: number of months US presidents spent in office.

- It is important that the amount of “jitter” be small compared to the distance between points. The purpose of the random displacements is only to make sure that no two points fall exactly on top of one another. We must make sure that points are not shifted significantly from their true location.
- We can jitter points in either horizontal or in vertical direction, or in both, depending on the data set and the purpose of the graph. In figure 2.1, points were jittered only in the *vertical* direction, so that their horizontal position (which corresponds to the actual data in this case, namely the number of months in office) is not altered and therefore remains exact.
- I used open, transparent rings as symbols for the data points. This is no accident: among different symbols of equal size, open rings are most easily recognized as separate even when partially occluded by each other. (By contrast, filled symbols tend to hide any substructure when overlapping, and symbols made out of straight lines, such as boxes and crosses, tend to be misleading because of the occurrence of parallel lines: check the top part of figure 2.1.)

Jittering is a good trick, which can be used in many different contexts. We will see further examples later in this book.

```

452.42
318.58
144.82
129.13
1216.45
991.56
1476.69
662.73
1302.85
1278.55
627.65
1030.78
215.23
44.50
...

```

Example 2.2: Response times for a web server (in milliseconds). Only the first few data points are shown. See figures 2.2 and 2.7.

2.2 Histograms and Kernel Density Estimates

Dot and jitter plots are nice because they are so simple. On the other hand, they are neither pretty nor very intuitive, and most importantly, they make it hard to read off *quantitative* information from the graph. In particular if we are dealing with larger data sets, we need a better type of graph, such as a histogram.

2.2.1 Histograms

To form a *histogram*, we divide the range of values into a set of “bins” and then count the number of points (sometimes called “events”) that fall into each bin. We then plot the count of events for each bin as a function of the position of the bin.

Again, let’s look at an example. In example 2.2, you can see the beginning of a file containing response times (in milliseconds) for queries against a web server or a database. In contrast to the previous example, this data set is fairly large, containing 1000 data points.

Figure 2.2 shows a histogram of this data set. I divided the horizontal axis into 60 bins of 50 milliseconds width, and counted the number of events in each bin.

What does the histogram tell us? We observe a rather sharp cut-off at a non-zero value on the left, meaning that there is a minimum completion time, below which no request can be completed. Then there is a sharp rise to a maximum at the “typical” response time, and finally there is a relatively large tail on the right, corresponding to the smaller number of requests that take a long time. This kind of shape is rather typical for a data set of task completion times, such

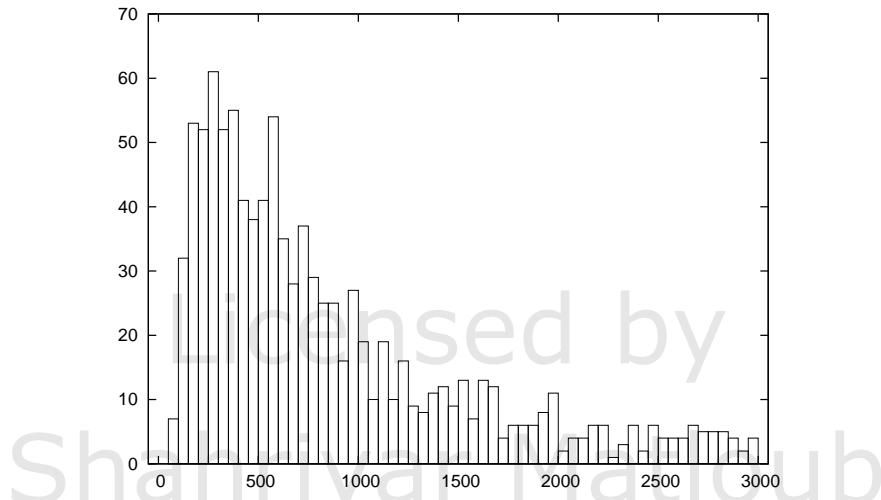


Figure 2.2: A histogram of the data in example 2.2.

as this one. If the data set had contained completion times for students to finish their homework, or for manufacturing workers to finish a work product, it would look qualitatively similar, except that the time scale would be a different one, of course. Basically, there is some minimum time that nobody can beat, a small group of very fast champions, a large majority, and finally a longer or shorter tail of “stragglers”.

It is important to realize that a data set does not determine a histogram uniquely. Instead, we have to fix *two* additional parameters to form a histogram: the bin width, and the alignment of the bins.

The quality of any histogram hinges on the proper choice of bin width: make it too large, and you loose a lot of detailed information about the data set. Make it too small, and you end up with few or no events in most of the bins and the shape of the distribution does not become apparent. Unfortunately, there is no simple rule-of-thumb that would provide a good bin width for a given data set: typically you have to try out several values for the bin width, until you obtain a satisfactory result. (As a first guess, you can start with *Scott’s Rule* for the bin width $w = 3.5\sigma / \sqrt[3]{n}$, where σ is the standard deviation for the entire data set and n is the number of points. This rule assumes that the data follows a Gaussian distribution, and will give a bin width that is likely to be too wide otherwise. See the end of this chapter for more information on the standard deviation.)

In addition to the bin width, there is another parameter, which we need to fix (whether we realize it, or not): the alignment of the bins on the x-axis. Let’s say we fixed the width of the bins at 1. Where do we now place the first bin? We can put it flush left, so that its left edge is at zero. Or we can center it at

zero. In fact, we can move all bins by half a bin width in either direction.

Unfortunately, this apparently insignificant little parameter can have a large influence on the appearance of the histogram. To demonstrate, look at this small data set:

```
1.4
1.7
1.8
1.9
2.1
2.2
2.3
2.6
```

Figure 2.3 shows two histograms of this data set. Both use the same bin width, namely 1, but use different alignment of the bins. In the top panel, where the bin *edges* have been aligned to coincide with the whole numbers ($1, 2, 3, \dots$), the data set appears to be flat. But in the bottom panel, where the bins have been *centered* on the whole numbers, the data set appears to have a rather strong central peak, and symmetric wings on both sides. It should be clear that we can construct even more pathological examples than this. In the next section, I will show you an alternative to histograms, which avoids this particular problem.

Before moving on, let's point out some additional technical details and variants relating to histograms.

- Histograms can be either normalized or unnormalized. In an unnormalized histogram, the value plotted for each bin is the absolute count of events in that bin. In a normalized histogram, we divide by the total number of points in the data set, so that the value for each bin becomes the *fraction* of points in that bin. If we want the *percentage* of points per bin instead, we need to multiply the fraction by 100.
- So far, I have always assumed that all bins have the same width. We can relax this constraint and allow bins of differing widths — narrower where points are tightly clustered, but wider in areas where there are only few points. This method can seem very appealing if the data set has outliers or areas with widely differing point density. Be warned, though, that you now have an additional ambiguity in your histogram: do you display the absolute number of points per bin, regardless of the width of each bin; or do you display the density of points per bin, by normalizing the point count per bin by the width of the bin? Either method is valid, and you cannot assume that your audience will know which convention you are following.
- It is customary to show histograms with rectangular boxes that extend from the horizontal axis, the way I have drawn figures 2.2 and 2.3. That

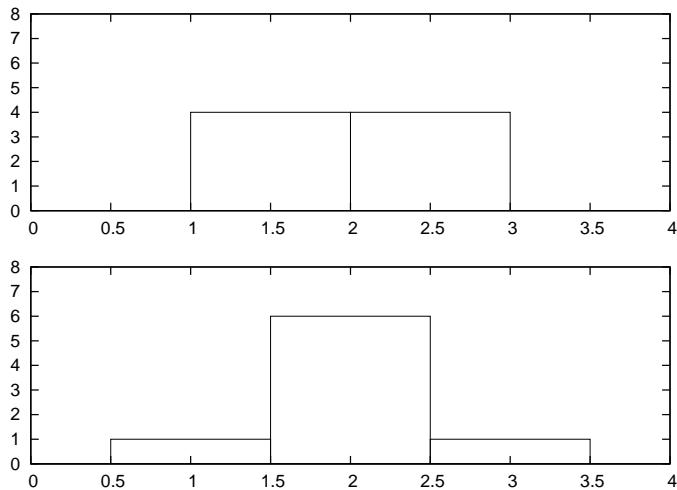


Figure 2.3: Histograms can look quite different, depending on the choice of anchoring point for the first bin. The figure shows two histograms of the same data set, using the same bin width. In the top panel, the bin edges are aligned on whole numbers, in the bottom panel, bins are centered on whole numbers.

is perfectly alright and has the advantage of explicitly displaying the bin width as well. (Of course, the boxes should be drawn in such a way that they align in the same way that the actual bins align — see figure 2.3.) This works well if you are only displaying a histogram for a single data set. But if you want to compare two or more data sets, the boxes start to get in the way, and you are better off drawing what is known as “frequency polygons”: you eliminate the boxes, and instead draw a symbol where the top of the box would have been. (The horizontal position of the symbol should be at the center of the bin!) Then connect consecutive symbols with straight lines. Now you can draw multiple data sets in the same plot, without cluttering the graph or unnecessarily occluding points.

- Don’t assume that the defaults of your graphics program are the best way to represent a histogram! I have already stated above why I consider frequency polygons almost always to be a better choice than using boxes for a histogram. If you nevertheless choose to use boxes, I suggest that you avoid filling them (with a color or hatch pattern) — your histogram will probably look cleaner and easier to read if you stick with just the outlines of the boxes. Finally, if you want to compare several data sets in the same graph, always use a frequency polygon, and stay away from stacked or clustered bar graphs, since these are particularly hard to read.

(We will come back to the problem of displaying composition problems in chapter 5.)

Histograms are very common, and they have a nice, intuitive interpretation. They are also very easy to generate — for a moderately sized data set, we could even do it by hand if we had to. On the other hand, histograms also have some serious problems as we have seen above. The most important ones are the following:

- The binning process required by all histograms loses information (by replacing individual data points locations with a “bin” of finite width). If we only have a few data points, we can ill afford to do so.
- Histograms are not unique. Depending on the alignment of the bins, the appearance of the histogram can be quite different, as we saw before in figure 2.3. (This non-uniqueness is a direct consequence of the information loss mentioned in the previous item.)
- On a more superficial level, histograms are ragged, not smooth. This matters little if we just want to draw a picture of them, but if we want to feed them back into a computer and use them as input for further calculations, a smooth curve would be easier to handle.
- Histograms don’t deal with outliers gracefully. A single outlier, far removed from the majority of the points, requires many empty cells in between, or forces us to use bins that are too wide for the majority of points. The possibility of outliers in particular makes it difficult to find an acceptable bin width in an automated fashion.

Fortunately, there is an alternative to classical histograms, which avoids all of these problems. It’s called *Kernel Density Estimate* or KDE for short. We’ll talk about it in the next section.

2.2.2 Kernel Density Estimates

Kernel Density Estimates (KDE) are a relatively new technique. In contrast to histograms, and to many other classical methods of data analysis, they pretty much *require* the calculational power of a reasonably modern computer to be effective — they cannot be done “by hand” with paper and pencil, even for rather moderately sized data sets. (It is interesting to see how the ready accessibility of computational and graphing power enables new ways to think about data!)

To form a KDE, we place a “kernel”, that is a smooth, strongly peaked function, at the position of each data point. We then add up the contributions from all kernels to obtain a smooth curve, which we can evaluate at any point along the x -axis.

Figure 2.4 shows an example. This is yet another representation of the data set we have studied before (example 2.1 and figure 2.1). The dotted boxes are

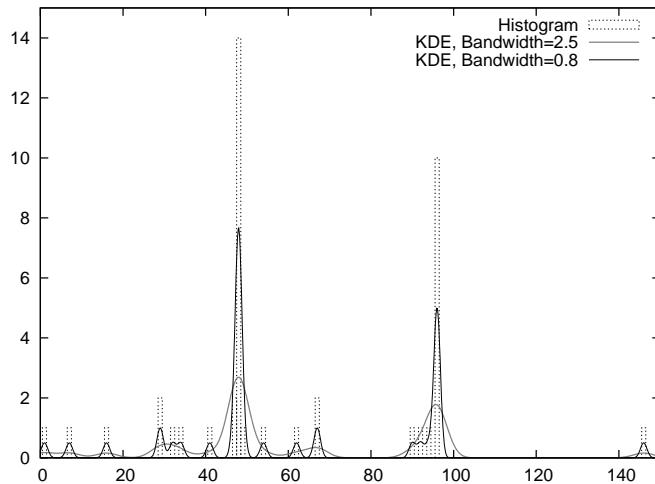


Figure 2.4: The data set from listing 2.1, shown as both a histogram and as a kernel density estimate.

a histogram of the data set (with bin width equal to 1), and the solid curves are two KDEs of the same data set with different bandwidths (I'll explain this concept in a moment). The shape of the individual kernel functions can be seen clearly, for example by considering the three data points below 20. You can also see how the final curve is composed out of the individual kernels, in particular when you look at the points between 30 and 40.

We can use any smooth, strongly peaked function as kernel, provided it integrates to 1; in other words, the area under the curve formed by a single kernel must be 1. (This is necessary to make sure the resulting KDE is properly normalized.) Some examples of kernel functions that you can find in the literature include (see figure 2.5):

$$\begin{aligned}
 K(x) &= \begin{cases} \frac{1}{2} & \text{if } |x| \leq 1 \\ 0 & \text{otherwise} \end{cases} && \text{Box or Boxcar Kernel} \\
 K(x) &= \begin{cases} \frac{3}{4}(1-x^2) & \text{if } |x| \leq 1 \\ 0 & \text{otherwise} \end{cases} && \text{Epanechnikov Kernel} \\
 K(x) &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right) && \text{Gaussian Kernel}
 \end{aligned}$$

The Box- and the Epanechnikov-Kernel are zero outside a finite range, whereas the Gaussian Kernel is non-zero everywhere, but negligibly small outside a limited domain. It turns out that the curve resulting for the KDE does not depend strongly on the particular choice of kernel function, so we are free to use

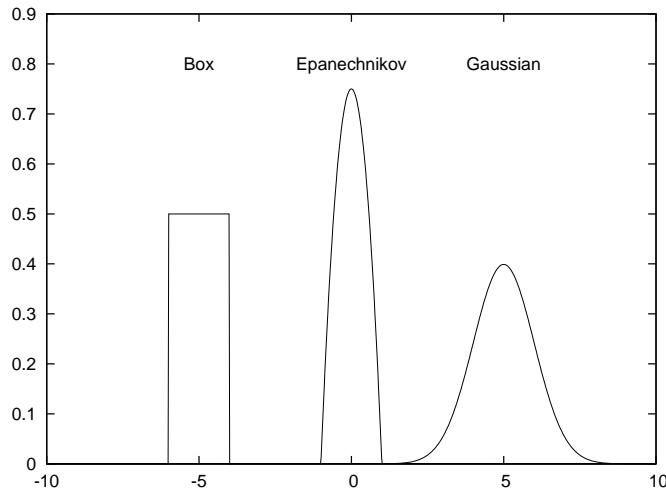


Figure 2.5: Examples of typical kernel functions.

the kernel that is the most convenient. Because it is so easy to work with, the Gaussian kernel is the most widely used. (See appendix B for more information on the Gaussian function).

To construct a KDE, we have to do two things: we have to move the kernel to the position of each point, by shifting it appropriately. For example, the function $K(x - x_i)$ will have its peak not at zero, but at x_i . And then, we have to choose the *bandwidth* of the kernel, which controls the spread of the kernel function. To make sure that the area under the curve stays the same as we shrink the width, we have to make the curve higher, and vice versa as we increase the width. The final expression for the shifted, rescaled kernel function of bandwidth h is:

$$\frac{1}{h} K\left(\frac{x - x_i}{h}\right)$$

This function has a peak at x_i , its width is approximately h , and its height is such that the area under the curve is still 1. Check figure 2.6 for some examples, using the Gaussian kernel. Keep in mind that the area under all three curves is the same!

Using this expression, we can now write down a formula for the KDE with bandwidth h for any data set $\{x_1, x_2, \dots, x_n\}$. This formula can be evaluated for any point x along the x -axis:

$$D_h(x; \{x_i\}) = \sum_{i=1}^n \frac{1}{h} K\left(\frac{x - x_i}{h}\right)$$

All of this is very straightforward, and easy to implement in any computer language. Be aware that for large data sets (with many thousands of points), all

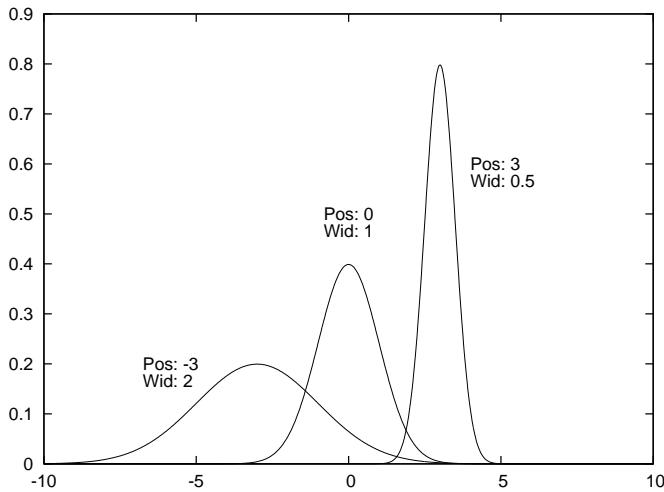


Figure 2.6: The Gaussian kernel for three different bandwidths. Note how the height of the kernel increases as the width decreases, so that the total area under the curve stays constant.

the kernel evaluations can become somewhat of a performance concern, in particular if the function $D(x)$ needs to be evaluated for many different positions (that is, many different values of x). If this is becoming a problem for you, you can try to find simpler kernel functions, or not evaluate a kernel if the distance $(x - x_i)$ is significantly larger than the bandwidth h .²

We can now come back to the wide gray line in figure 2.4: it is a KDE with a larger bandwidth. Using such a large bandwidth makes it impossible to resolve the individual data points, but it emphasizes entire *periods* of greater or smaller frequency. Which choice of bandwidth is right for you depends on your purpose.

A KDE, constructed in the way just described, is similar to a classical histogram, but avoids two of the aforementioned problems: given data set and bandwidth, a KDE is unique; and it is smooth, provided we have chosen a smooth kernel function, such as the Gaussian.

2.2.3 Optimal Bandwidth Selection

We still have to fix the bandwidth, though. This is a different *kind* of problem than the other two: it's not just a technical problem, which can be made to go away through a

²Yet another strategy starts with the realization that forming a KDE amounts to a convolution of the kernel function with the data set. You can now take the Fourier transform of both kernel and data set and make use of the *Fourier Convolution Theorem*. This approach is suitable for very large data sets, but is a bit outside the scope of our discussion.

cleverer method; instead, it's a fundamental problem that relates to the data set itself. If the data follows a very smooth distribution, a wider bandwidth is appropriate, but if the data comes from a very wiggly distribution instead, we need a smaller bandwidth to retain all relevant detail. In other words, the optimal bandwidth is a property of the data set and tells us something about the nature of the data.

So, how do we choose an optimal value for the bandwidth? Intuitively, the problem is clear: we want the bandwidth to be narrow enough to retain all relevant detail, but wide enough so that the resulting curve is not too "wiggly". This is a problem that we will encounter in every approximation problem: balancing the faithfulness of representation with simplicity of behavior. Statisticians speak of the "bias/variance trade-off".

To make matters concrete, we have to define a specific expression for the error of our approximation, which takes into account both bias and variance. We can then choose a value for the bandwidth that minimizes this error. For KDEs, the generally accepted measure is the "expected mean-square error" between the approximation and the true density. The big problem is that we don't know the true density that we try to approximate, so it seems impossible to calculate (and minimize) the error in this way. But clever ways have been developed to make progress. These methods fall broadly into two categories: we can either try and come up with explicit expressions for both bias and variance. Balancing them leads to an equation that has to be solved numerically. Alternatively, we can realize that the KDE is an approximation for the probability density from which the original set of points was chosen. We can now choose points from this approximation and see whether these points replicate the KDE that we started with. If it does, we can consider the result a good approximation to the true distribution. This latter method is known as *cross-validation*, and we will come back to it in chapter 12.

Although not particularly hard, the details of both methods would lead us too far afield, so I will skip them here. If you are interested, you will have no problem picking up the details from one of the references at the end of this chapter. Be warned though that these methods will find the optimal bandwidth *with respect to the mean-square error*, which tends to overemphasize bias over variance, and therefore result in rather narrow bandwidths and KDEs that appear too "wiggly". If you are using KDEs to generate graphs with the purpose of obtaining intuitive visualizations of point distributions, you might be better off with a little bit of manual trial-and-error, combined with visual inspection. Keep in mind that there is no "right" answer, only the most suitable one for a given purpose. And the most suitable to develop intuitive understanding might not be the one that minimizes a particular mathematical quantity.

2.3 Cumulative Distribution Function

The main advantage of histograms and kernel-density estimates is that they have an immediate intuitive appeal: they tell us how probable it is to find a data point with a certain value. For example, looking at figure 2.2, it is immediately clear that values around 250 milliseconds are very likely to occur, but that values greater than 2000 milliseconds are quite rare.

But how rare exactly? That is a question that is much harder to answer by looking at the histogram in figure 2.2. Besides wanting to know how much weight is in the tail, we might also be interested to know what fraction of requests completes in the typical band between 150 and 350 milliseconds. It's

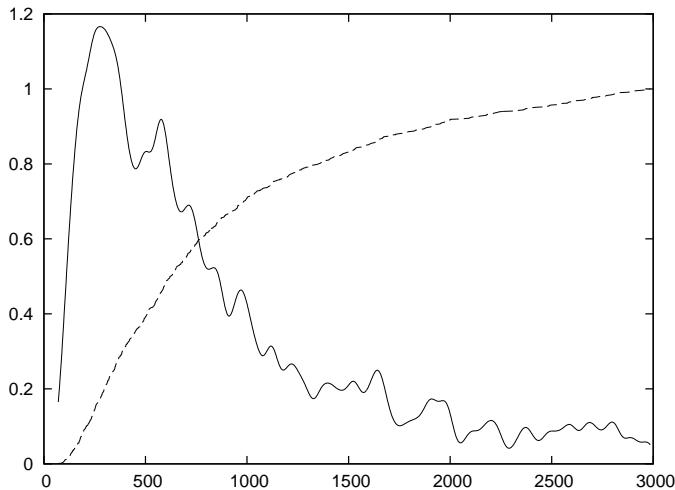


Figure 2.7: Kernel density estimate and cumulative distribution function of the of the server data in example 2.2.

certainly the majority of events, but if we wanted to know how many exactly, we would have to sum up the contributions from all bins in the respective region.

The cumulative distribution function (CDF) does just that. The CDF at point x tells us what fraction of events has occurred “to the left” of x . Put differently, the CDF is the fraction of all points with $x_i \leq x$. Figure 2.7 shows the same data set that we have already encountered in figure 2.2 before, but now it shows a KDE (with bandwidth $h = 30$) instead of a histogram, and in addition it shows the corresponding CDF. (Note that both KDE and CDF are normalized to 1.)

We can now read off several interesting observations from the CDF. For instance, we can see that at $t = 1500$ (which certainly puts us into the “tail” of the distribution), the CDF is nevertheless smaller than 0.85, meaning that fully 15 percent of all requests take longer than 1500 milliseconds. By contrast, less than a third of all requests are completed in the “typical” time frame from 150 to 500 milliseconds. (How do we know this? The CDF for $t = 150$ is about 0.05 and close to 0.39 for $t = 500$. In other words, about 39 percent of all requests are completed in less than 500 milliseconds. Of those, 5 percent are completed in less than 150 milliseconds. Hence, about 34 percent of all requests have response times between 150 and 500 milliseconds.)

It is worth to pause and contemplate these findings, because they demonstrate how misleading a histogram (or KDE) can be, despite (or because of) their intuitive appeal! Judging from the histogram or KDE alone, it seems quite reasonable to assume that “most” of the events occur within the major

peak near $t = 300$, and that the tail for $t > 1500$ contributes relatively little. Yet the CDF tells us very clearly that this is not so. (The problem is that the eye is much better at judging distances compared to areas, and so we are misled by the large values of the histogram near its peak and fail to see that nevertheless the area underneath the peak is not very large compared to the total area under the curve.)

CDFs are probably the least well-known and most under-appreciated tool in basic graphical analysis. They have less immediate intuitive appeal than histograms or KDEs, but they allow us to make the kind of quantitative statement that is very often required and that is difficult (if not outright impossible) to get from a histogram.

CDFs have a number of important properties that follow directly from the way they are calculated:

- Because the value of the CDF at position x is the fraction of points to the left of x , a CDF is always monotonically increasing with x .
- CDFs are less wiggly than a histogram (or KDE), but contain the same information in a representation that is inherently less noisy.
- Because CDFs do not involve any “binning”, they do not lose information and are therefore a more faithful representation of the data than a histogram.
- All CDFs approach zero as x goes to negative infinity. CDFs are usually normalized so that they approach 1 (or 100 percent) as x goes to positive infinity.
- A CDF is unique, given a data set.

If you are mathematically inclined, you probably already realized that the CDF is (an approximation to) the antiderivative of the histogram, and that the histogram is the derivative of the CDF:

$$\begin{aligned} \text{cdf}(x) &\approx \int_{-\infty}^x dt \text{histo}(t) \\ \text{histo}(x) &\approx \frac{d}{dx} \text{cdf}(x) \end{aligned}$$

CDFs have several uses. First, and most importantly, they let us answer questions such as those posed earlier in this section: what fraction of points falls between any two values? The answer can simply be read off from the graph. CDFs also help us to understand how imbalanced a distribution is: what fraction of the overall weight is carried by the tails.

CDFs also prove useful when we want to compare two distributions against one another. It is notoriously difficult to compare two bell-shaped curves in a histogram against each other. Comparing the corresponding CDFs is usually much more conclusive.

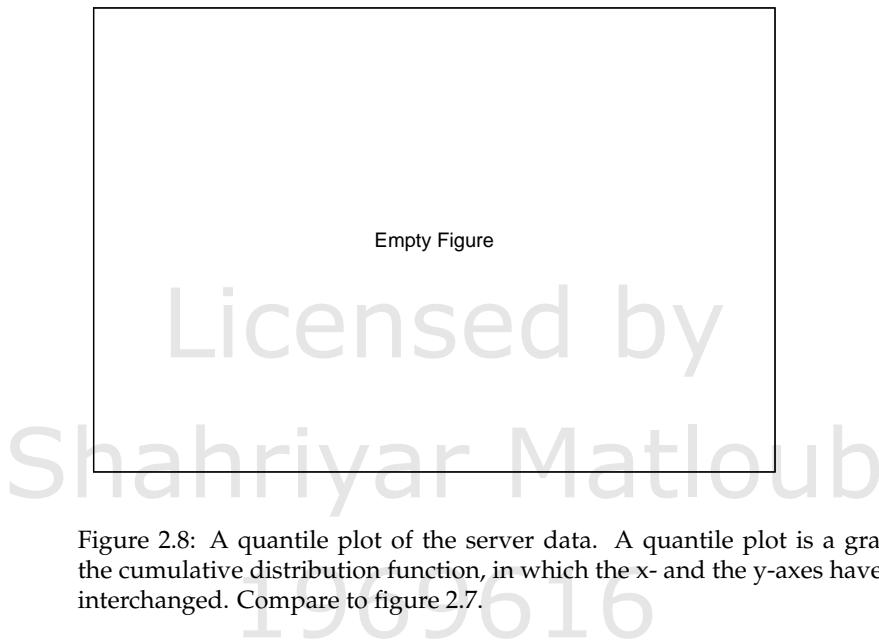


Figure 2.8: A quantile plot of the server data. A quantile plot is a graph of the cumulative distribution function, in which the x- and the y-axes have been interchanged. Compare to figure 2.7.

One last remark, before leaving this section: in the literature, you may find the term *quantile plot*. A quantile plot is nothing but the plot of a CDF, in which the x- and y-axes have been switched: figure 2.8 shows an example, using the familiar server response time data set. Plotted this way, we can easily answer questions such as “What response time corresponds to the 10th percentile of response times?”. But the information contained in this graph is of course exactly the same as in a graph of the CDF.

2.3.1 Comparing Distributions with Probability- or QQ-plots

Occasionally, you might want to confirm that a given set of points is distributed according to some specific, known distribution. For example, you have a data set and you would like to see whether it can be described well by a Gaussian, or some other, distribution.

You could compare a histogram or KDE of the data set against the theoretical density function directly, but it is notoriously difficult to compare distributions that way, in particular out in the tails. A better idea would be to compare the cumulative distribution functions, which are easier to handle since they are less wiggly and monotonically always increasing. But this is still difficult — also keep in mind that most probability distributions depend on location and scale parameters (such as mean and variance), which you would have to estimate *before* being able to make a meaningful comparison. Isn’t there a way to compare a set of points directly against a theoretical distribution, and in the process read off the estimates for all the parameters required?

Turns out, there is. And although the method is technically easy to do, the underly-

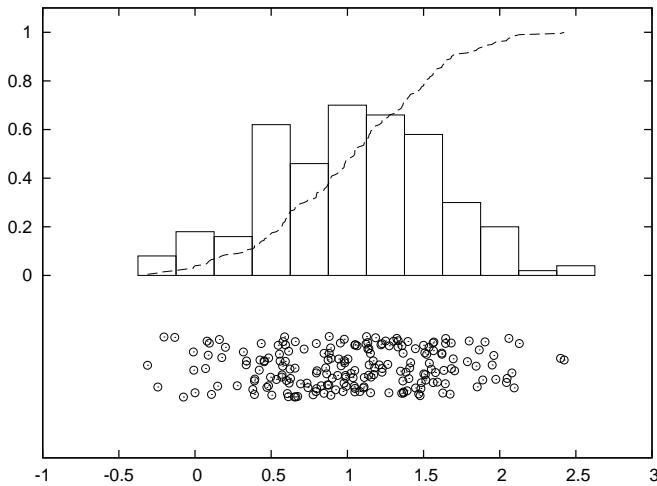


Figure 2.9: Jitter plot, histogram, and cumulative distribution function for a Gaussian data set.

ing logic is a bit convoluted, and tends to trip up even experienced practitioners.

Here is how it works. Consider a set of points $\{x_i\}$ that we suspect are distributed according to the Gaussian distribution. In other words, we expect the cumulative distribution function of the set of points: $y_i = \text{cdf}(x_i)$ to be the Gaussian cumulative distribution function $\Phi((x - \mu)/\sigma)$, with mean μ and standard deviation σ :

$$y_i = \Phi\left(\frac{x_i - \mu}{\sigma}\right) \quad \text{if data is Gaussian!}$$

Here, y_i is the value of the cumulative distribution function corresponding to the data point x_i , in other words, y_i is the *quantile* of the point x_i .

Now comes the trick. We apply the *inverse* of the Gaussian distribution function on both sides:

$$\Phi^{-1}(y_i) = \frac{x_i - \mu}{\sigma}$$

With a little bit of algebra, this becomes:

$$x_i = \mu + \sigma\Phi^{-1}(y_i)$$

In other words, if we plot the values in the data set as a function of $\Phi^{-1}(y_i)$, then they should fall onto a straight line, with slope σ and zero-intercept μ . If, on the other hand, the points do not fall onto a straight line after applying the inverse transform, then we can conclude that the data is not distributed according to a Gaussian distribution in the first place.

The resulting plot is known as a *probability plot*. Because it is easy to spot deviation from a straight line, it is a relatively sensitive test to determine whether a set of points behaves according to the Gaussian distribution. As an added benefit, we can read off estimates for the mean and the standard deviation directly from the graph: μ is the

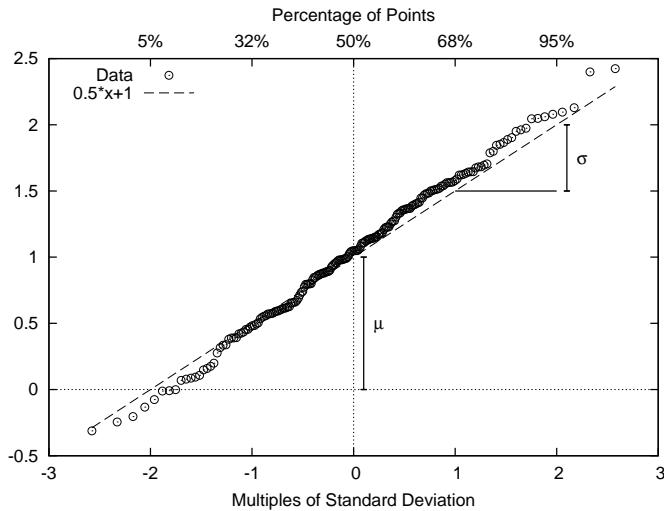


Figure 2.10: Probability plot for the data set shown in figure 2.9.

intercept of the curve with the y-axis, and σ is given by the slope of the curve. (See figure 2.9 and figure 2.10.)

One important question concerns the *units* that we plot along the axes. For the vertical axis, the case is clear: we use whatever units the original data was measured in. But what about the horizontal axis? We plot the data as a function of $\Phi^{-1}(y_i)$, which is the inverse Gaussian distribution function, applied to the percentile y_i for each point x_i . We can therefore choose between two different ways to dissect the horizontal axis: we can either use the percentiles y_i directly (the tick marks will not be distributed uniformly in this case). Alternatively, we can divide the horizontal axis uniformly, which means that we are using *the width of the standard Gaussian distribution* as a unit. (You can convince yourself that this is really true by realizing that $\Phi^{-1}(y)$ is the inverse of the Gaussian distribution function $\Phi(x)$. Now ask yourself: what units is x measured in? We use the same units for the horizontal axis in a Gaussian probability plot.) There is a special terminology for these units: they are called *probits* in parts of the literature. Figure 2.10 shows both sets of units side-by-side. (Beware of confused and confusing explanations of this point elsewhere in the literature.)

There is one more technical detail that we need to discuss: to produce a probability plot, we need not only the data itself, but for each point x_i , we also need its quantile y_i (we will discuss quantiles and percentiles in more detail later in this chapter). The simplest way to obtain the percentiles, given the data, is to:

1. sort the data points in ascending order.
2. assign to each data point its rank (basically, its line number in the sorted file), starting at 1 (not 0).
3. the percentile y_i now is the rank divided by $n + 1$, where n is the number of data points

This prescription makes sure that each data point is assigned a percentile which is

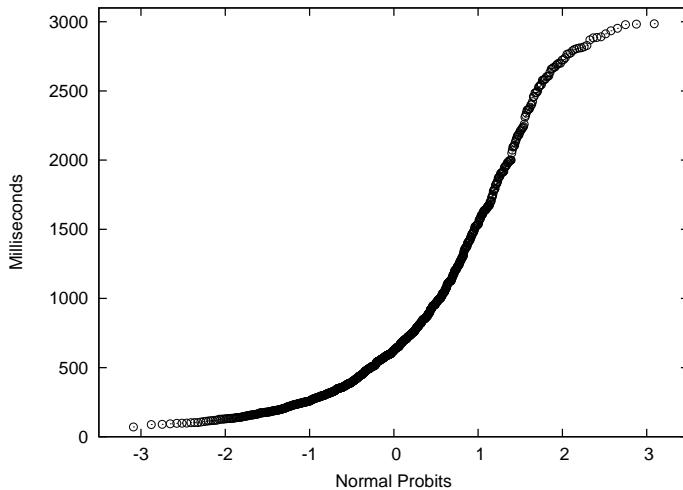


Figure 2.11: TBD

strictly greater than zero and strictly less than one. This is important, because $\Phi^{-1}(x)$ is only defined for $0 < x < 1$. This prescription is easy to understand and easy to remember, but I should point out that you will find other, slightly more complicated prescriptions discussed in the literature. For all practical purposes, the differences are going to be small.

Finally, let's look at an example where the data is clearly *not* Gaussian. Figure 2.11 shows the data from 2.2, plotted in a probability plot. The points don't fall on a straight line at all — not surprisingly, since we already knew from figure 2.2 that the data is not Gaussian. But for cases that are less clear-cut, the probability plot can be a helpful tool to detect deviations from Gaussian behavior.

A few additional comments:

- Nothing in the preceding discussion requires that the distribution be Gaussian! You can use almost any other commonly used distribution function (and its inverse) to generate the respective probability plots. In particular, many of the commonly used probability distributions depend on location and scale parameters in exactly the same way as the Gaussian distribution, and therefore all the arguments discussed above go through as before.
- So far, I have always assumed that we want to compare an *empirical* data set against a *theoretical* distribution. But there may also be situations where we want to compare two empirical data sets against each other — for example to find out whether they were drawn from the *same* family of distributions (without having to specify the family explicitly). The process is easiest to understand if both data sets we want to compare contain the same number of points: you sort them both, and then align the points from both data sets that have the same rank (once sorted). You now plot the resulting pairs of points in a regular scatter plot (see chapter 3). The resulting graph is known as *QQ-plot*. (If the two data sets do not

contain the same number of points, you will have to interpolate or truncate them to make them so.)

Probability plots are an relatively advanced, specialized technique, and you need to evaluate whether you really need them. Their purpose is to determine whether a given data set stems from a specific, known distribution. Occasionally, this is of interest in by itself, and there are other situations, where subsequent analysis depends on proper identification of the underlying model. For example, many statistical techniques assume that the errors or residuals are Gaussian, and are not applicable if this condition is violated: probability plots are a convenient technique to test this assumption.

2.4 Rank-Order Plots and Lift Charts

There is a technique related to histograms and CDFs, which is worth knowing about. Consider the following scenario: A company which is selling textbooks, curriculum materials, and so on is planning an email marketing campaign to reach out to its existing customers. For this campaign, the company wants to use personalized email messages, tailored to the job title of each recipient. (In other words, teachers would get a different email than their principals.) Problem is: the customer data base contains about 250,000 individual customer records, totaling 16,000 different job titles! Now what?

The trick is to sort the job titles by the number of individual customer records corresponding to each job title! The first few records are shown in table 2.3. The columns are: the job title, the number of customers for that job title, the fraction of all customers having that job title, and finally the cumulative fraction of customers. For the last column, we sum up the number of customers for the current and all previously seen job titles, then divide by the total number of customer records. It is the equivalent to the CDF we learned about earlier.

We can see immediately that fully two thirds of all customers have one of only ten different job titles. If we take into account the first 30 job titles, we get 75 percent coverage of customer records. That's much more manageable than the 16,000 job titles we started with!

Let's step back for a moment to understand how this example is different than the examples we have seen previously. What is important to notice here is that *the independent variable has no intrinsic ordering*. What does that mean?

For the web server example above, we counted the number of events for each response time, hence the count of events per bin was the dependent variable, and it was determined by the independent variable, namely the response time. In that case, the independent variable had an inherent ordering: 100 milliseconds are always less than 400 milliseconds (and so on). But in the case of counting customer records matching a certain job title, the independent variable (namely the job title) has no corresponding ordering relation. It may appear differently, because we can sort the job titles alphabetically. But realize that this ordering is entirely arbitrary! There is nothing "fundamental" about it. If we choose a different font encoding or locale, the order will change.

Teacher	66470	0.34047	0.340
Principal	22958	0.11759	0.458
Superintendent	12521	0.06413	0.522
Director	12202	0.06250	0.584
Secretary	4427	0.02267	0.607
Coordinator	3201	0.01639	0.623
Vice Principal	2771	0.01419	0.637
Program Director	1926	0.00986	0.647
Program Coordinator	1718	0.00880	0.656
Student	1596	0.00817	0.664
Consultant	1440	0.00737	0.672
Administrator	1169	0.00598	0.678
President	1114	0.00570	0.683
Program Manager	1063	0.00544	0.689
Supervisor	1009	0.00516	0.694
Professor	961	0.00492	0.699
Librarian	940	0.00481	0.704
Project Coordinator	880	0.00450	0.708
Project Director	866	0.00443	0.713
Office Manager	839	0.00429	0.717
Assistant Director	773	0.00395	0.721
Administrative Assistant	724	0.00370	0.725
Bookkeeper	697	0.00357	0.728
Intern	693	0.00354	0.732
Program Supervisor	602	0.00308	0.735
Lead Teacher	587	0.00300	0.738
Instructor	580	0.00297	0.741
Head Teacher	572	0.00292	0.744
Program Assistant	572	0.00292	0.747
Assistant Teacher	546	0.00279	0.749

Example 2.3: The first 30 job titles and their relative frequencies.

Contrast this with ordering relationship on numbers — there are no two ways about it: 1 is always less than 2.

In such a case, where the independent variable does not have an intrinsic ordering, it is very often a good idea to sort entries by the *dependent* variable. That's what we did in the example above: rather than defining some (arbitrary) sort order on the job titles, we instead sorted by the number of records (that is, by the dependent variable). Once the records have been sorted in this way, we can form a histogram and a CDF as before.

This trick of sorting by the dependent variable is more generally useful whenever the independent variable does not have a meaningful ordering relation, and is not limited only to situations where we count events per bin. Figures 2.12 and 2.13 show two typical examples.

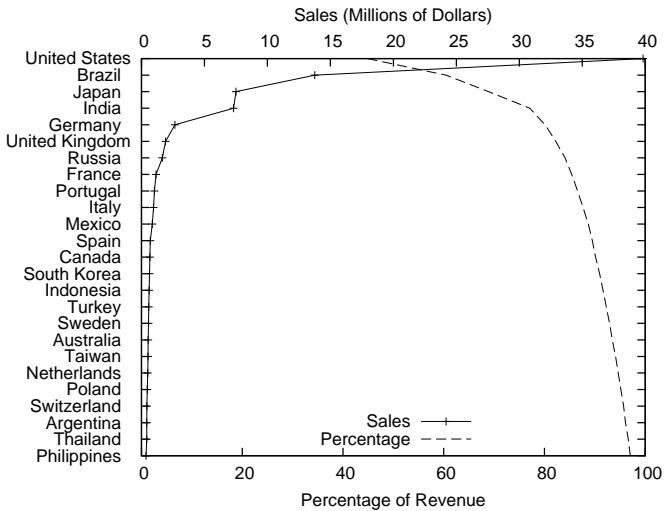


Figure 2.12: A rank-order plot: sales per country. Notice the independent variable along the *vertical* axis: this makes the text labels more convenient to read.

In figure 2.12 we show the sales by a certain company to different countries. Not only the sales to each country are shown, but also the cumulative sales, which lets us assess the importance of the remaining “tail” of the distribution of sales.

In this example, I choose to plot the independent variable along the vertical axis — this is often a good idea when the values are strings: there are easier to read this way. (If you plot them along the horizontal axis, you have to turn the strings by 90 degrees to make them fit, but then they are hard to read.)

Figure 2.13 finally is what in quality engineering is known as a *Pareto chart*. In quality engineering and process improvement, the goal is to reduce the number of defects in a certain product or process. You collect all known causes of defects and then observe how often each one occurs. The results can be summarized conveniently in a chart like the one in figure 2.13. Note how we have sorted the causes of defects by their frequency of occurrence. From this chart we can see immediately that problems with the engine and the electrical system are much more common than problems with the air conditioning, the brakes, or the transmission. In fact, by looking at the cumulative error curve, we can tell that fixing just the first two problem spots would reduce the overall defect rate by 80 percent.

Two more bits of terminology: the term “Pareto chart” is not used widely outside the specific engineering disciplines I have mentioned in the previous paragraph. I personally prefer the expression *rank-order chart* for any plot which has been generated by first sorting all entries by the dependent variable (that is by the *rank* of the entry). And the cumulative distribution curve is

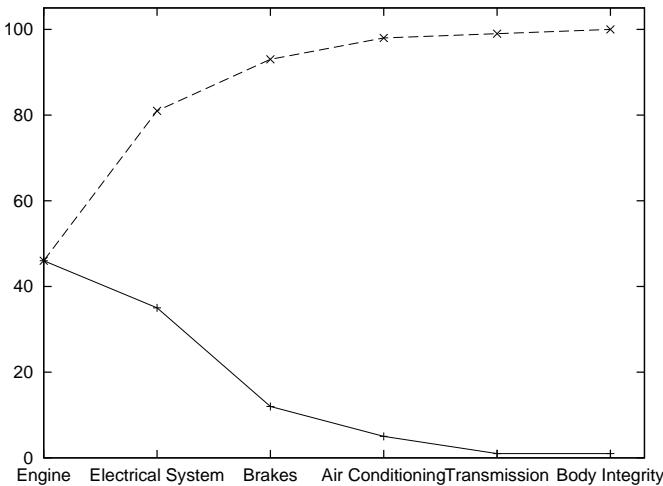


Figure 2.13: A “Pareto chart”: another example of a rank-order plot.

occasionally referred to as *lift curve*, because it tells us how much “lift” we get from each entry or a certain range of entries.

2.5 Only when Appropriate: Summary Statistics and Box Plots

You may have noticed that so far I have not spoken at all about such simple topics as mean and median, standard deviation and percentiles. That is quite intentional. These so-called *summary statistics* apply only under certain assumptions and are misleading, if not downright wrong, if these assumptions are not fulfilled. I know that they are easy to understand and easy to calculate, but if there is one message I would like you to take away from this book it is this: the fact that something is convenient and popular to do is no reason to follow suit. For any method that you want to use, make sure you understand the underlying assumptions, and *always* check that they are fulfilled for the specific application you have in mind!

Mean, median, and related summary statistics only apply to distributions that have a single, central peak, so-called *unimodal* distributions. If this basic assumption is not fulfilled, conclusions based on simple summary statistics are going to be wrong — even worse, nothing will tip you off that they are wrong: the numbers will look quite reasonable. (We will see an example for this problem in a moment.)

2.5.1 Summary Statistics and Box-and-Whisker Plots

If a distribution has only a central peak, then it makes sense to ask about the properties of this peak: where is it located, and what is its width? We may also want to know whether the distribution is symmetric, and if there are any outliers present.

Mean and standard deviation are two popular measures for location and spread. The *mean* or average is both familiar and intuitive:

$$m = \frac{1}{n} \sum_i x_i$$

The standard deviation measures how far points spread “on average” from the mean — that is, we take all the differences between each individual point and the mean, and then form the average of all these differences. Because individual points can both overshoot or undershoot the mean, and we don’t want positive and negative deviations to cancel each other, we sum the *square* of the individual deviations, and then form the mean of the square deviations. (The second equation is very useful in practice and can be found from the first after plugging in the definition of the mean.)

$$\begin{aligned}s^2 &= \frac{1}{n} \sum_i (x_i - m)^2 \\ &= \frac{1}{n} \sum_i x_i^2 - m^2\end{aligned}$$

This quantity is known as the variance and is the more important quantity from a theoretic point of view. But as a measure of the spread of a distribution we are better off using its square-root, which is known as the *standard deviation*. Why take the square root? Because now both the measure for the location and the measure for the spread have the same units, which are also the units of the actual data. (If our data set consists of the prices for a basket of goods, then the variance would be given in “square-dollars”, whereas the standard deviation is given in dollars.)

For many (but certainly not all!) data sets arising in practice, one can expect about two-thirds of all data points to fall into the interval $[m - s, m + s]$, and 99 percent of all points to fall into the wider interval $[m - 3s, m + 3s]$.

Mean and standard deviation are very easy to calculate, and they have certain very nice mathematical properties — provided the data is symmetric and does not contain crazy outliers. Unfortunately, many data sets violate at least one of these assumptions. Here is an example for the kind of trouble that one may encounter: let’s assume we have 10 items, costing 1 dollar each, and one item, costing 20 dollars. The mean item price comes out to be 2.73 dollars, despite the fact that no item has a price anywhere close to this value. The standard deviation is even worse: it comes out to 5.46 dollars, implying that most items have a price between 2.73-5.46 and 2.73+5.46 dollars. The “expected range”

now includes *negative prices* — an obviously absurd result. Note that the data set itself is *not* particularly pathological: going to the grocery store and picking up a handful of candy bars and a bottle of wine will do it. (Pretty good wine, to be sure, but nothing outrageous.)

A different set of summary statistics that is both more flexible and more robust is based on the concept of *median* and *quantiles* and *percentiles*. The median is conventionally defined as that value from a data set, so that half of all points in the data set are smaller than it, and the other half greater. Percentiles are the generalization of this concept to other fractions (the 10th percentile is that value, so that 10 percent of the points in the data set are smaller than it, etc). Quantiles are similar to percentiles, only that they are taken with respect to the fraction of points, not the percentage of points (in other words, the 10th percentile equals the 0.1 quantile).

Simple as it is, the percentile concept is nevertheless ambiguous and we need to work a little harder to make it really concrete. As an example of the problems that occur, consider the following data set: (1,2,3). What is the median? It is not possible to break this data set into two equal parts, each containing exactly half of the points. The problem gets even more uncomfortable when we are dealing with arbitrary percentile values (rather than the only median).

The internet standard laid down in RFC 2330 (“Framework for IP Performance Metrics”) gives a definition of percentiles in terms of the CDF, which is unambiguous and practical, as follows: *The p-th percentile is the smallest value x, such that the cdf(x) is larger or equal p/100, i.e. the min(x) such that cdf(x) ≥ p/100.* (This definition assumes that the CDF is normalized to 1, not to 100. If it was normalized to 100, the condition would be $\text{cdf}(x) \geq p$.)

Using this definition, the median (that is the 50th percentile) of the data set (1,2,3) is 2, since the $\text{cdf}(1) = 0.33\dots$, $\text{cdf}(2) = 0.66\dots$, $\text{cdf}(3) = 1.0$. The median of the data set (1,2) would be 1, because now $\text{cdf}(1) = 0.5$, and $\text{cdf}(2) = 1.0$.

The median is a measure for the location of the distribution, and we can use percentiles to construct a measure for its width. Probably the most frequently used quantity for this purpose is the *inter-quartile range* (IQR), which is the distance between the 75th percentile and 25th percentile.

When should you favor median and percentile over mean and standard deviation? Whenever you suspect that your distribution is not symmetric or has important outliers.

If a distribution is symmetric and well-behaved, mean and median will be quite close together and there is little difference in using either. Once the distribution becomes skewed however, the basic assumption that underlies the mean as a measure for the location of the distribution is no longer fulfilled, and you are better off using the median. (This is why the average wage in official publications is usually given as the median family income, not the mean: the latter would be distorted significantly by the very few households with very high incomes.) Furthermore, the moment you have outliers, the assumptions going into the standard deviation as a measure of the width of the distribution are violated, and you should favor the IQR (remember our shopping basket

example earlier!).

If median and percentiles are so great, why don't we always use them? A large part of the preference for mean and variance is historical. In the days before readily available computing power, percentiles were simply not practical to calculate. (Keep in mind that to find percentiles, you need to *sort* the data set, whereas to find the mean you just need to add up all elements in any order. The latter is an $\mathcal{O}(n)$ process, but the former is an $\mathcal{O}(n^2)$ process, since humans — being non-recursive — cannot be taught Quicksort and have to resort to much less efficient sorting algorithms.) A second reason is that is much harder to prove rigorous theorems on percentiles, while mean and variance are mathematically very well-behaved and easy to work with.

2.5.2 Box-and-Whisker Plots

There is an interesting graphical way to represent these quantities, together with information about potential outliers, known as *box-and-whisker plot*, or box plot for short. Figure 2.15 illustrates all parts of a box plot. A box plot consists of:

- A *marker* or symbol for the median, as indicator of the *location* of the distribution.
- A *box*, spanning the inter-quartile range, as a measure of the *width* of the distribution.
- A set of *whiskers*, extending from the central box to the upper and lower adjacent values, as indicator of the *tails* of the distribution. (See below for the definition of the concept of "adjacent value".)
- Finally, individual *symbols* for all values outside the range of adjacent values, as a representation for *outliers*.

You can see that a box plot combines a lot of information in a single graph. We have encountered almost all of these concepts before, with the exception of upper and lower *adjacent value*. While the inter-quartile range is a measure for the width of the central "bulk" of the distribution, the adjacent values are a way to understand how far its tails reach. The upper adjacent value is the largest value in the data set, that is less than twice the inter-quartile range greater than the median. In other words: extend the whisker upwards from the median to twice the length of the central box. Now trim the whisker down to the largest value that actually occurs in the data set. That value is the upper adjacent value. (A similar construction holds for the lower adjacent value.)

You may wonder about the reason for this peculiar construction. Why not simply extend the whiskers to (say) the 5th and 95th percentile, and be done with it? The problem with this suggestion is that it does not allow us to recognize true outliers! Outliers are data points that are, *compared to the width of the distribution*, unusually far from the center. Such values may or may not be

present. The top and bottom 5 percent, on the other hand, are always present, even for very compact distributions. To recognize outliers, we therefore cannot simply look at the most extreme values, but we have to *compare their distance from the center to the overall width of the distribution*. That is what box-and-whisker plots, as described in the previous paragraph, do.

The logic behind the preceding argument is very important (not just in this application, but more generally), and therefore I will restate the steps: *first*, we calculated a measure for the width of the distribution, *then* we used this width to identify outliers as those points which are far from the center, where (and this is the crucial step) “far” is measured in units of the width of the distribution. We don’t impose a distance from the outside, nor do we simply label the most extreme x percent of the distribution as outliers — instead, we determine the width of the distribution (as the range into which points “typically” fall) and then use it to identify outliers as those points that deviate from this. The important insight here is that the distribution itself determines a typical *scale*, which provides a natural unit in which to measure other properties of this distribution. This idea of using some typical property of the system to describe other parts of the system will come up again later (see chapter 8).

Box plots combine many different measures for a distribution into a single, compact plot. Looking at a box plot, it is possible to see whether the distribution is symmetric or not, and how the weight is distributed between the central peak and the tails. Finally, outliers (if present) are not dropped, but shown explicitly.

Box plots are best when used to compare several distributions against one another — for a single distribution, the overhead of preparing and managing a graph compared to just quoting the numbers often does not appear justified. Here is an example that compares different data sets against each other.

Let’s say we have a data set containing the index of refraction of 121 samples of glass from crime scenes.³ The data set is broken down by the type of glass: 70 samples of window glass, 29 from headlamps, 13 from containers of various kinds, and 9 from tableware. Figures 2.14 and 2.15 are two representations of the same data: one as a kernel-density estimate, the other as a box plot.

The box plot emphasizes the overall structure of the data sets and makes it very easy to compare the data sets based on their location and width, but at the same time, it also loses much information. The KDE gives a more detailed view of the data, in particular the occurrence of multiple peaks in the distribution functions, but makes it more difficult to quickly sort and classify the data sets. Depending on our needs, we may prefer one or the other technique at any given time.

A couple of additional notes on box plots:

- The specific way to draw a box plot that I described here is particularly useful, but is far from universal. In particular the specific definition of

³The raw data can be found in the “Glass Identification Data Set” on the UCI Machine Learning Repository (<http://www.ics.uci.edu/~mlearn/MLRepository.html>)

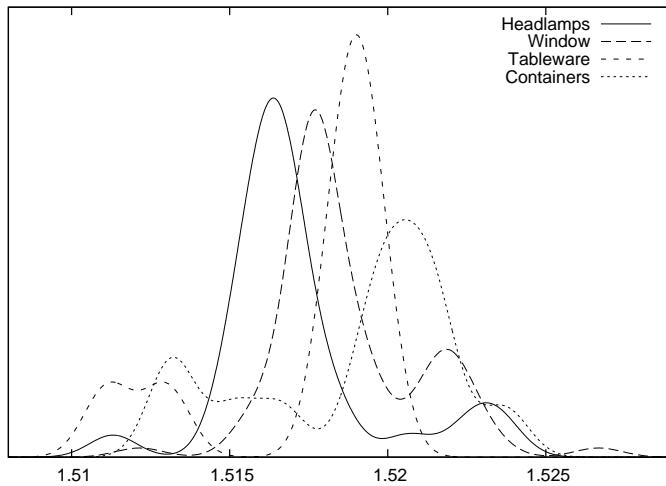


Figure 2.14: Comparing data sets using KDEs: refractive index of different types of glass.

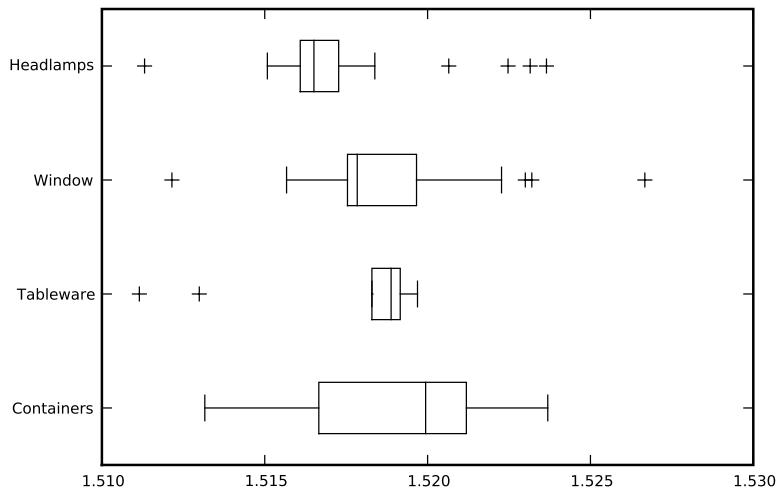


Figure 2.15: Comparing data sets using box plots: refractive index of different types of glass.

the adjacent values is often not properly understood. Whenever you find yourself looking at a box plot, always ask what exactly is shown, and whenever you prepare one, make sure to include an explanation.

- The box plot that I described here can be further modified and enhanced. One suggestion, for example, uses the width of the central box (that is, the direction orthogonal to the whiskers) to indicate the size of the underlying data set: the more points are included, the wider the box. Another suggestion is to give up the rectangular shape of the box altogether and use local width of the box to display the density of points at that point — which brings us almost full circle to KDEs.

2.6 Workshop: NumPy

The NumPy module provides efficient and convenient handling of large numerical arrays in Python. It is the successor to both the earlier Numeric and the alternative numarray modules. (See the appendix A for more on the history of scientific computing with Python.) The NumPy module is used by many other libraries and projects, and in this sense is a “base” technology.

Let’s look at some quick examples, before delving a bit deeper into technical details.

2.6.1 NumPy in Action

NumPy objects are of type `ndarray`. There are different ways of creating them. We can create an `ndarray` by:

- converting a Python list.
- using a factory function that returns a populated vector.
- reading data from file directly into a NumPy object.

Listing 2.4 shows five different ways of creating NumPy objects. First, we create one by converting a Python list. Then we show two different factory routines, which generate equally spaced grid points. Both routines differ in the way they interpret the provided boundary values: one includes both, the other includes one and excludes the other. Then we create a vector filled with zeros and set each element in a loop. Finally we read the data from a text file. (I am showing only the simplest or default cases here — all these routines have many more options which can be used to influence their behavior.)

In the end, all five vectors contain identical data. You should note that the values in the Python list used to initialize `vec1` are floating point values, and that we specified the *type* desired for the vector elements explicitly when using the `arange()` function to create `vec2`. (We will come back to types in a moment.)

```

import numpy as np

# From a Python list
vec1 = np.array( [ 0., 1., 2., 3., 4. ] )

# arange( start inclusive, stop exclusive, step size )
vec2 = np.arange( 0, 5, 1, dtype=float )

# linspace( start inclusive, stop inclusive, number of elements )
vec3 = np.linspace( 0, 4, 5 )

# zeros( n ) returns a vector filled with n zeros
vec4 = np.zeros( 5 )
for i in range( 5 ):
    vec4[i] = i

# read from a text file, one number per row
vec5 = np.loadtxt( "data" )

```

Example 2.4: TBD

Now that we have created these objects, we can operate with them (see listing 2.5). One of the major conveniences provided by NumPy is that we can operate with NumPy objects as if they were atomic data types: we can add, subtract, multiply them, and so forth, *without the need for explicit loops*. Avoiding explicit loops makes our code clearer. It also makes it faster (because the entire operation is performed in C without overhead — see below).

All operations are performed element-by-element: if we add two vectors, the corresponding elements from each vector are combined to give the element in the resulting vector. In other words, the compact expression `vec1 + vec2` for `v1` in the listing is equivalent to the explicit loop construction used to calculate `v2`. This is true even for multiplication: `vec1 * vec2` will result in a vector in which corresponding elements of both operands have been multiplied element-by-element. (If you want a true vector or “dot” product, you need to use the `dot()` function instead.) Obviously, this requires that *all operands have the same number of elements!*

Next, we demonstrate two further convenience features, which in the NumPy documentation are referred to as *broadcasting* and *ufuncs* (short for “universal functions”). The term “broadcasting” in this context has nothing to do with messaging. Instead, it means that if you try to combine two arguments of different shapes, then the smaller one will be extended (“cast broader”) to match the larger one. This is particularly useful when combining scalars with vectors: the scalar is expanded to a vector of appropriate size, all elements of which have the value given by the scalar, then the operation proceeds, element-by-element, as before. The term “ufunc” refers to a scalar function, which can be

```

# ... continuation from previous listing

# Add a vector to another
v1 = vec1 + vec2

# Unnecessary: adding two vectors using an explicit loop
v2 = np.zeros( 5 )
for i in range( 5 ):
    v2[i] = vec1[i] + vec2[i]

# Adding a vector to another in place
vec1 += vec2

# Broadcasting: combining scalars and vectors
v3 = 2*vec3
v4 = vec4 + 3

# Ufuncs: applying a function to a vector, element-by-element
v5 = np.sin(vec5)

# Converting to Python list object again
lst = v5.tolist()

```

Example 2.5: TBD

applied to a NumPy object. The function is applied, element-by-element, to all entries in the NumPy object, and the result is a new NumPy object with the same shape as the original one.

Using these features skillfully, a function to calculate a kernel density estimate can now be written as a *single* line of code (see listing 2.6).

The program in listing 2.6 will calculate and print the data needed to generate figure 2.4 (but it does not actually draw the graph — that will have to wait until the Workshop in chapter 3).

Most of the listing is boiler-plate code, such as reading and writing files. All the actual work is done in the one-line function `kde(z, w, xv)`. This function is making use both of “broadcasting” and “ufuncs”, and is a good example for the style of programming typical of NumPy. Let’s dissect it — inside out.

First recall what we need to do to evaluate a KDE: for each location z , at which we want to evaluate the KDE, we need to find its distance to all the points in the data set. For each point, we evaluate the kernel for this distance and sum up the contributions from all the individual kernels to obtain the value of the KDE at z .

The expression $z-xv$ generates a *vector*, containing the distance between z and all the points in xv (that’s broadcasting). We then divide by the required bandwidth, multiply with $1/2$ and square each element. Finally, we apply the

```

from numpy import *

# z: position, w: bandwidth, xv: vector of points
def kde( z, w, xv ):
    return sum( exp(-0.5*((z-xv)/w)**2)/sqrt(2*pi*w**2) )

d = loadtxt( "presidents", usecols=(2,) )

w = 2.5

for x in linspace( min(d)-w, max(d)+w, 1000 ):
    print x, kde( x, w, d )

```

Example 2.6: TBD

exponential function `exp()` to this vector (that's a ufunc). The result is a vector, which contains the exponential function, evaluated at the distances between the points in the data set and the location `z`. All we have to do is to sum all the elements in the vector (that's what `sum()` does), and we are done, calculating the KDE at position `z`. If we want to plot the KDE as a curve, we have to repeat this process for each location that we want to plot — that's what the final loop in the listing is for.

2.6.2 NumPy in Detail

You may have noticed that none of the warm-up examples in listings 2.4 and 2.5 contained any matrices or other data structures of higher dimensionality — just one-dimensional vectors. To understand how NumPy treats objects with dimensionality higher than one, we need to develop at least a superficial understanding for the way NumPy is implemented.

It is misleading to think of NumPy as a “matrix package for Python” (although it's commonly used as such). I find it more helpful to think of NumPy as a wrapper and access layer for underlying C buffers. These buffers are contiguous blocks of C memory, which — by their nature — are one-dimensional data structures. All elements in those data structures must be of the same size, and we can specify almost any native C type (including C structs) as type of the individual elements. The default type corresponds to a C `double` and that is what we will be using in the examples below, but keep in mind that other choices are possible. All operations that apply to the data overall are performed in C and are therefore very fast.

To interpret the data as matrix or other multi-dimensional data structure, the shape or layout is imposed during element access. The same twelve element data structure can therefore be interpreted as a 12 element vector, or a 3×4 matrix, or a $2 \times 2 \times 3$ tensor — the shape only comes into play through the way we access the individual elements. (Keep in mind that while reshaping

a data structure is very easy, resizing it is not.)

The encapsulation of the underlying C data structures is not perfect: when we choose the types of the atomic elements, we specify C data types, not Python types. Similarly, some features provided by NumPy allow us manage memory manually (see below), rather than have the memory be managed transparently by the Python runtime. This is an intentional choice, because NumPy has been designed to accommodate *large* data structures — large enough that you might want (or need) to exercise a greater degree of control over the way memory is managed. For this reason, you have the ability to choose types that take up less space as elements in a collection (for instance C `float` elements, rather than the default `double`). For the same reason, all ufuncs accept an optional argument pointing to an (already allocated) location where the results will be placed, to avoid having to claim additional memory themselves. Finally, several access and structuring routines return a *view* (not a copy!) of the same underlying data. This does pose an aliasing problem that you need to watch out for!

Listing 2.7 shows a quick example to demonstrate the concept of shape and views. Here, I assume that the commands are entered at an interactive Python prompt (shown as `>>>` in the listing). Output generated by Python is shown without a prompt.

Let's step through this. We create two vectors of 12 elements each. Then we `reshape` the first one into a 3×4 matrix. Note that the `shape` property is a data member — not an accessor function! For the second vector, we create a *view* in the form of a 3×4 matrix. Now `d1` and the newly created view on `d2` have the same shape, and we can combine them (by forming their sum, in this case). Note that `reshape()`, although a member function, does *not* change the shape of instance itself, but instead returns a new view object: `d2` is still a one-dimensional vector. (There is also a stand-alone version of this function, so that we could also have written: `view = np.reshape(d2, (3,4))`. The presence of such redundant functionality is due to the desire to maintain backwards compatibility with *both* of NumPy's ancestors.)

We can now access individual elements of the data structures, depending on their shape. Since both `d1` and `view` are matrices, they are indexed by a pair of indices (in the order `[row,col]`). On the other hand, `d2` is still a one-dimensional vector, and hence takes only a single index. (We will have more to say about indexing in a moment.)

Finally, we examine some diagnostics regarding the shape of the data structures, emphasizing their precise semantics. The `shape` is a tuple, giving the number of elements in each dimension. The `size` is the total number of elements and corresponds to the value returned by `len()` for entire data structure. Finally, `ndim` gives the number of dimensions (that is: `d.ndim == len(d.shape)`), and is equivalent to the "rank" of the entire data structure. (Again, the redundant functionality is there to maintain backwards compatibility.)

Finally, let's take a closer look at the ways that we can access elements or larger subsets of an `ndarray`. In listing 2.7, we have seen how to access an individual element, by fully specifying an index for each dimension. However,

we can also specify larger subarrays of a data structure using two techniques known as *slicing* and *advanced indexing*. Listing 2.8 shows some representative examples. (Again, consider this an interactive Python session.)

We create a twelve element vector and reshape it as a 3×4 matrix as before. Slicing uses the standard Python slicing syntax `start:stop:step`, where the start position is inclusive, while the stopping position is exclusive. (In the listing, I am only using the simplest form of slicing, selecting all available elements.)

There are two potential gotchas with slicing. First of all, specifying an explicit subscripting index (not a slice!) reduces the corresponding dimension to a scalar. By contrast, slicing does not reduce the dimensionality of the data structure. Consider the two extreme cases: in the expression `d[0,1]`, indices for both dimensions are fully specified, so that we are left with a scalar. By contrast, `d[0:1,1:2]` is sliced in both dimensions. Neither dimension is removed, and the resulting object is still a (two-dimensional) matrix, but of smaller size: it has shape 1×1 .

The second stumbling point is that *slices return views*, not copies.

Besides slicing, we can also index an `ndarray` with a vector of indices, in an operation called “advanced slicing”. The listing shows two simple examples: in the first, we use a Python list object, containing the integer indices (that is: positions) of the desired columns and in the desired order, to select a subset of columns. In the second example, we form an `ndarray` of Boolean entries, to select only those rows for which the Boolean evaluates to `True`.

In contrast to slicing, *advanced indexing returns copies*, not views.

This completes our overview of the basic capabilities the NumPy module. NumPy is easy and convenient to use for simple use cases, but can get very confusing otherwise (for example, check out the rules for general broadcasting when both operators are multi-dimensional, or for advanced indexing).

We will see some more straightforward applications in chapters 3 and 4.

2.7 Summary

This concludes our survey of methods to study univariate data. The most important questions when exploring univariate data sets concern the overall shape of the distribution of points; and, more specifically, its location and shape.

We looked at simple graphical methods, such as dot and jitter plots, and at more sophisticated methods, such as histograms and kernel density estimates, which allows us to discern more detail about the data. I stressed the importance of cumulative distribution functions, which — although less intuitive than a simple histogram — nevertheless allow us to make quantitative statements about the distribution of points.

Finally, I introduced summary statistics, such as mean, median, and so on, and discussed their limitations. We also learned about box plots as a compact method to compare several data sets against each other.

We also introduced some more advanced or specialized techniques, such as rank-order plots (for data sets where the independent variable has not specific ordering), and QQ-plots, which can be used to determine if two data sets have the same distribution.

2.8 Further Reading

- *The Elements of Graphing Data.* William S. Cleveland. 2nd ed., Hobart Press. 1994.
A book-length discussion of graphical methods for data analysis like those described in this chapter. In particular, you will find more information here on topics such as box plots and QQ-plots. Cleveland's methods are particularly careful and well thought out.
- *All of Statistics: A Concise Course in Statistical Inference.* Larry Wasserman. Springer. 2004.
A thoroughly modern treatment of mathematical statistics, very advanced and condensed. You will find some additional material here on the theory of "density estimation", that is on histograms and KDEs.
- *Multivariate Density Estimation.* David W. Scott. 2nd ed., Wiley. 2006.
A research monograph on density estimation, by the creator of "Scott's Rule".
- *Kernel Smoothing.* M. P. Wand and M. C. Jones. Chapman & Hall. 1995.
Accessible treatment of Kernel Density Estimation.

```

>>> import numpy as np

>>> # Generate two vectors with 12 elements each
>>> d1 = np.linspace( 0, 11, 12 )
>>> d2 = np.linspace( 0, 11, 12 )

>>> # Reshape the first vector to a 3x4 (row x col) matrix
>>> d1.shape = ( 3, 4 )
>>> print d1
[[ 0.   1.   2.   3.]
 [ 4.   5.   6.   7.]
 [ 8.   9.  10.  11.]]

>>> # Generate a matrix VIEW to the second vector
>>> view = d2.reshape( (3,4) )

>>> # Now: possible to combine the matrix and the view
>>> total = d1 + view

>>> # Element access: [row,col] for matrix
>>> print d1[0,1]
1.0
>>> print view[0,1]
1.0
>>> # ... and [pos] for vector
>>> print d2[1]
1.0

>>> # Shape or layout information
>>> print d1.shape
(3,4)
>>> print d2.shape
(12,)
>>> print view.shape
(3,4)

>>> # Number of elements (both commands equivalent)
>>> print d1.size
12
>>> print len(d2)
12

>>> # Number of dimensions (both commands equivalent)
>>> print d1.ndim
2
>>> print np.rank(d2)
1

```

Example 2.7: TBD

```
>>> import numpy as np

>>> # Create a 12 element vector and reshape into 3x4 matrix
>>> d = np.linspace( 0, 11, 12 )
>>> d.shape = ( 3,4 )
>>> print d
[[ 0.   1.   2.   3.]
 [ 4.   5.   6.   7.]
 [ 8.   9.  10.  11.]]

>>> # Slicing...
>>> # First row
>>> print d[0,:]
[ 0.  1.  2.  3.]

>>> # Second col
>>> print d[:,1]
[ 1.  5.  9.]

>>> # Individual element: scalar
>>> print d[0,1]
1.0

>>> # Sub-vector of shape 1
>>> print d[0:1,1]
[ 1.0]

>>> # Sub-array of shape 1x1
>>> print d[0:1,1:2]
[[ 1.0]]

>>> # Indexing...
>>> # Integer indexing: third and first column
>>> print d[ :, [2,0] ]
[[ 2.   0.]
 [ 6.   4.]
 [10.   8.]]

>>> # Boolean indexing: second and third column
>>> k = np.array( [False, True, True] )
>>> print d[ k, : ]
[[ 4.   5.   6.   7.]
 [ 8.   9.  10.  11.]]
```

Example 2.8: TBD

Chapter 3

Two Variables: Establishing Relationships

When we are dealing with a data set consisting of *two* variables, we are mostly interested to see whether some kind of relationship exists between the two variables, and if so, what kind this relationship is.

Plotting one variable against another is very straightforward, so most of our effort will be spent on various tools and transformations, which can be applied to characterize the nature of the relationship between the two inputs.

3.1 Scatter Plots

Plotting one variable against another is very simple — you just *do it!* In fact, this is precisely what most people mean when they speak about “plotting” something. Yet, there are differences, as we will see.

Figures 3.1 and 3.2 show two examples. The data in figure 3.1 might come from an experiment that measures the force between two surfaces at short separation. Clearly, the force is a complicated function of the distance — on the other hand, the data points fall on a relatively smooth curve, and we can have confidence that it represents the data accurately. (To be sure, we should ask for the accuracy of the measurements shown in this graph: are there significant error bars attached to the data points? But it doesn’t matter: the data itself shows clearly that there is no significant *random* noise in the data. This isn’t to say that there aren’t problems with the data, but if there are any, they will be *systematic* problems, for instance with the apparatus, and statistical methods will not be helpful.)

In contrast, figure 3.2 shows the the kind of data typical of much of statistical analysis. Here, we might be showing the prevalence of skin cancer as a function of the mean income for a group of individuals, or the unemployment rate as a function of the frequency of high-school drop-outs for a number of

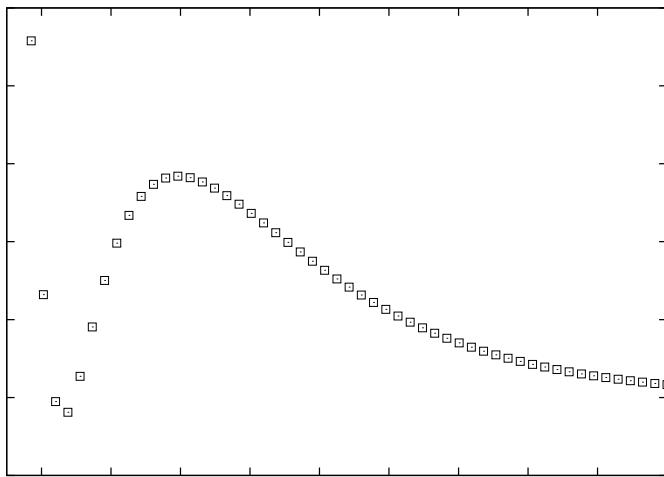


Figure 3.1: Data that clearly shows that there is relationship, although a complicated one, between x and y .

counties, and the primary question is whether there is any relationship at all between the two quantities. The situation here is quite different from the one shown in figure 3.1, where it was clear that there was indeed a strong relationship between x and y and therefore our main concern was to determine the precise nature of that relationship.

A figure such as figure 3.2 is usually referred to as a *scatter plot*, but I prefer the term *xy-plot*. Scatter plot sounds to me too much like “splatter plot”, and it insinuates that the data necessarily will be noisy — but we don’t know that! Once we plot it, the data may turn out to be very clean and regular, as in figure 3.1, hence I am more comfortable with the neutral term.

Whenever we create a graph such as figure 3.1 or figure 3.2, we usually want to understand whether there is a relationship between x and y , and what this relationship is. Figure 3.3 shows a number of different possibilities (clockwise, from top left) that we may find: no relationship; strong, simple relationship; strong not-so-simple relationship; and finally a relationship that is not unique.

3.2 Conquering Noise: Smoothing

When the data is very noisy, we are more concerned with establishing *whether* the data exhibits a meaningful relationship, rather than its precise character. To see this, it is often very helpful to find a smooth curve which represents the noisy data set. Trends and structure of the data may be more easily visible from

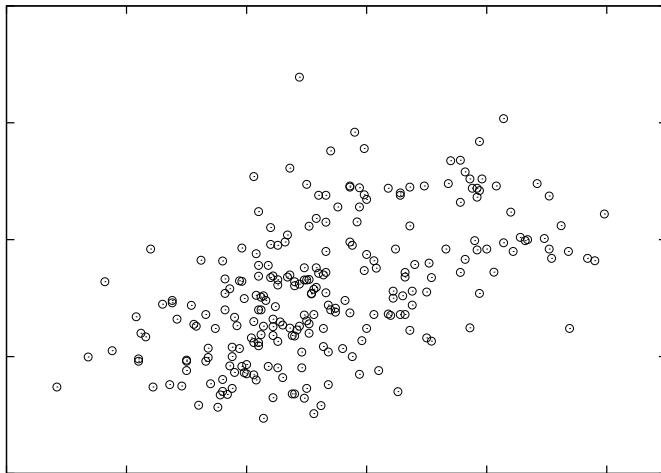


Figure 3.2: A noisy data set. Is there any relationship between x and y at all?

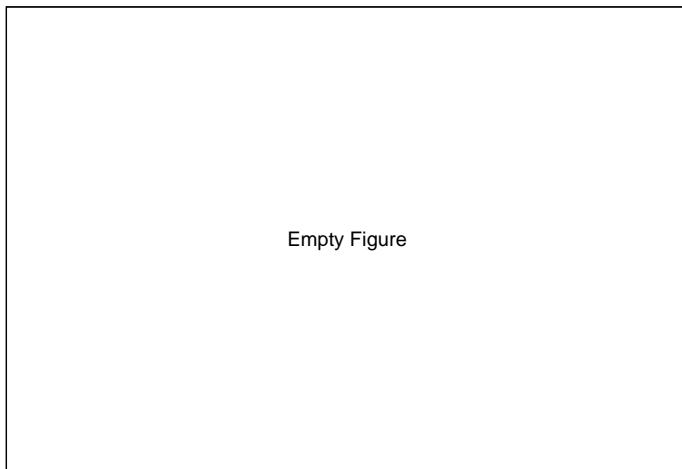


Figure 3.3: TBD

such a curve than from the cloud of points.

Two different methods are frequently used to provide smooth representation of noisy data sets: *weighted splines* and a method known as *LOESS* (or LOWESS), which is short for locally weighted regression.

Both methods work by approximating the data in a small neighborhood

(that is: locally) by a polynomial of low order (at most cubic). The trick is to string the various local approximations together to form a single smooth curve! Both methods contain an adjustable parameter, which controls the “stiffness” of the resulting curve: the stiffer it is, the smoother it appears, but the less accurately does it reflect individual data points. Striking the right balance between smoothness and accuracy is the main challenge when it comes to smoothing methods.

3.2.1 Splines

Splines are constructed from piece-wise polynomial functions (typically cubic), which are joined together in a smooth fashion. In addition to the local smoothness requirements at each joint, splines must also fulfill a global smoothness goal by optimizing the following functional:

$$J[s] = \alpha \int \left(\frac{d^2s}{dt^2} \right)^2 dt + (1 - \alpha) \sum_i w_i (y_i - s(x_i))^2$$

Here, $s(t)$ is the spline curve, (x_i, y_i) are the coordinates of the data points, the w_i are weight factors (one for each data point), and finally α is a mixing-factor. The first term controls how “wiggly” the spline is overall, because the second derivative measures the curvature of $s(t)$ and becomes large if the curve has many wiggles. The second term measures how accurately the spline represents the data points, by measuring the squared deviation: it becomes large if the spline does not pass close to the data points. Each term in the sum is multiplied by a weight factor w_i , which can be used to give greater weight to data points known more accurately than others. (Put differently, we can regard $w_i = 1/d_i^2$, where d_i measures how close the spline should pass by y_i at x_i .) The mixing parameter α finally controls how much weight we give to the first term (emphasizing overall smoothness) compared to the second term (emphasizing accuracy of representation). In a plotting program, α is usually the dial we use to tune the spline for a given data set.

To construct the spline explicitly, we form cubic interpolation polynomials for each consecutive pair of points, and require that these individual polynomials have the same values, as well as first and second derivatives at the points where they meet. These smoothness conditions lead to a set of linear equations for the coefficients in the polynomials, which can be solved. Once the coefficients have been found, the spline curve can be evaluated at any desired location.

3.2.2 LOESS

Splines have an *overall* smoothness goal, which means that they are less responsive to *local* details in the data set. The LOESS smoothing method addresses that. It consists of approximating the data locally through a low-order

(typically linear) polynomial (that is the regression part), but weighting all data points in such a way that points close to the location of interest contribute more strongly than data points further away (thus constituting the local weighting).

Let's consider the case of first-order (linear) LOESS, so that the local approximation takes the particularly simple form $a + bx$. To find the "best fit" in a least-squares sense, we must minimize:

$$\chi^2 = \sum_i w(x - x_i; h) (a + bx_i - y_i)^2$$

with respect to the two parameters a and b . Here, $w(x)$ is the weight function. It should be smooth and strongly peaked — in fact, it is basically a "kernel", similar to the ones we have encountered in figure 2.5, when we discussed Kernel Density Estimates. The kernel most often used with LOESS is the "tri-cube kernel" $K(x) = (1 - |x|^3)^3$ for $|x| < 1$, $K(x) = 0$ otherwise, but any of the other kernels will work, too. The weight depends on the distance between the point x where we want to evaluate the LOESS approximation and the location of the data points. In addition, the weight function also depends on a parameter h , which controls the bandwidth of the kernel: this is the primary control parameter for LOESS approximations. Finally, the value of the LOESS approximation at position x is given by $y(x) = a + bx$, where a and b minimize the expression for χ^2 above.

This is the basic idea behind LOESS. You can see that it is easy to generalize it, for example to two or more dimensions, or two higher order approximation polynomials. (One problem, though: if you use only first order polynomials, then explicit expressions for the parameters a and b can be found, whereas for quadratic or higher polynomials you will have to resort to numerical minimization techniques. Unless you have truly compelling reasons you want to stick to the linear case!)

LOESS is a very computationally intensive method. Keep in mind that we need to perform the entire calculation again for *every* point at which we want to obtain a smoothed value. (In other words, the parameters a and b that we calculated are themselves functions of x .) This is in contrast to splines: once the spline coefficients have been calculated, the spline can be evaluated easily at any point that we wish. In this way, splines provide a summary or approximation to the data. LOESS, on the other hand, does not lend itself easily to semi-analytical work: what you see is pretty much all you get.

One final observation: if we replace the linear function $a + bx$ in the fitting process with the constant function a , the LOESS becomes a weighted moving average

3.2.3 Examples

Let's look at two examples where smoothing reveals behavior that would otherwise not be visible.

The first is a famous data set that has been analyzed in many places: the 1970 draft lottery. During the Vietnam war, men in the US were drafted based

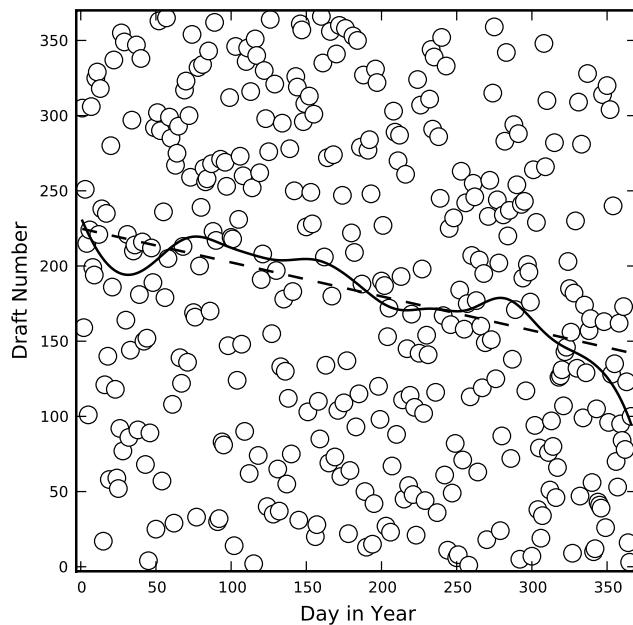


Figure 3.4: TBD

on their date of birth: each possible birth date was assigned a draft number between 1 and 366 using a lottery process, and men were drafted in the order of their draft numbers. However, soon complaints became loud that the lottery was biased, such that men born later in the year had a higher probability to have a low draft number, and consequentially a higher chance to be drafted early.¹

Figure 3.4 shows all possible birth dates (as days since the beginning of the year) and their assigned draft numbers. If the lottery had been fair, these points should form a completely random pattern. Looking at the data alone, it is basically impossible to tell whether there is any structure to the data. However, the smoothed LOESS lines reveal a strong falling tendency of the draft number over the course of the year: birth dates later in the year indeed have a higher chance of having a smaller draft number!

The LOESS lines have been calculated using a Gaussian kernel — for the solid line, I used a kernel bandwidth equal to 5, but for the dashed line, I used a

¹More details and a description of the lottery process can be found in *The Statistical Exorcist*. M. Hollander, F. Proschan. CRC Press. 1984.

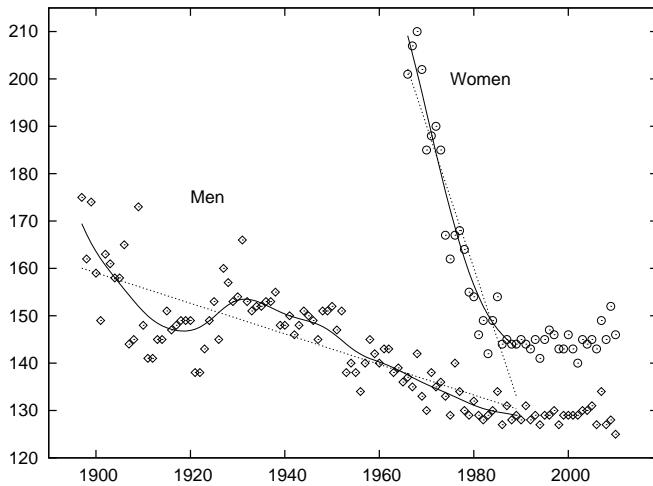


Figure 3.5: TBD

much larger bandwidth of 100. For such a large bandwidth, basically all points in the data set contribute equally to the smoothed curve, so that the LOESS operation reverts to a linear regression of the entire data set. (In other words, if we make the bandwidth very large, LOESS amounts to a least-squares fit of a straight line to the data.)

In the previous example, we mostly cared about a *global* property of the data, namely the presence or absence of an overall trend. Because we were looking for a global property, a stiff curve (such as a straight line) was sufficient to reveal what we were looking for. However, if we are want to extract more detail, in particular if we want to extract *local* features, then we need a “softer” curve, which can follow the data on smaller scales.

Figure 3.5 shows an amusing example.² Displayed are the finishing times for the winners in a marathon, separately for men and women. Also shown are the “best fit” straight line approximations, for all events up to 1990. According to this model, women should start finishing faster than men before the year 2000 and then continue to become faster at a dramatic rate! This expectation is not borne out by the actual observations: finishing times for women (and men) have largely leveled off.

This example demonstrates the danger of attempting to describe data using a model of fixed form (a “formula”) — and a straight line is one or the most rigid models out there! If the model is not appropriate for the data, it will lead to incorrect conclusions. Moreover, it will not necessarily be obvious that the

²This example was inspired by *Graphic Discovery: A Trout in the Milk and Other Visual Adventures*. Howard Wainer. 2nd ed., Princeton University Press. 2007.

model is inappropriate — look again at figure 3.5: don't the straight lines seem reasonable?

Also shown in figure 3.5 are smoothed curves calculated using a LOESS process. Because these curves are “softer”, they have a greater ability to capture features contained in the data. And, indeed, the LOESS curve for the women's result does give an indication that the trend of dramatic improvements, seen since they first started competing in the mid-60s, had already begun to level off before the year 1990. (All curves are based strictly on data prior to 1990.) This is a good example how an adaptive smoothing curve can highlight local behavior that is present in the data, but may not be obvious when just looking at the individual data points.

3.2.4 Residuals

Once you have obtained a smoothed approximation to the data, you usually also want to check out the *residuals*, that is the remainder when you subtract the smooth “trend” from the actual data.

There are several details to look for when studying residuals.

- Residuals should be balanced: symmetrically distributed around zero.
- Residuals should be free of a trend. The presence of a trend, or any other, large-scale systematic behavior in the residuals is an indicator that the model is inappropriate! (By construction, this is never a problem if the smooth curve has been obtained from an adaptive smoothing model, but it is an important indicator if the smooth curve comes from an analytic model.)
- Residuals will necessarily straddle the zero value, that is, take on both positive and negative values. You may also want to plot their absolute values: this will give an indication whether the overall magnitude of the residuals is the same for the entire data or not. The assumption that the magnitude of the variance around a model is constant throughout (“homoscedasticity”) is often an important assumption in statistical methods. If it is not fulfilled, such methods may not apply.
- Finally, you may want to check whether the residuals are distributed according to a Gaussian distribution, using a QQ-plot (chapter 2). This also is an assumption that is often important for more advanced statistical methods.

Finally, it may be useful to apply a smoothing routine to the *residuals* to be able to recognize their features more clearly. Figure 3.6 shows the residuals for the women's marathon results (before 1990), both for the straight line and the LOESS smoothing curve. For the latter, the residuals are small overall hardly exhibit any trend. For the straight line model, however, there is a strong systematic trend in the residuals, which is increasing in magnitude for years past

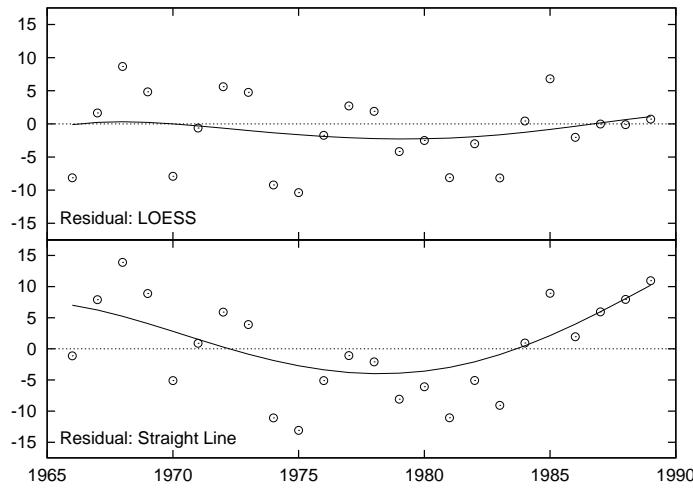


Figure 3.6: TBD

1985. This kind of systematic trend in the residuals is a clear indicator that the model is not appropriate for the data!

3.2.5 Additional Ideas and Warnings

Here are some additional ideas that you might want to play with.

As we have discussed before, you can calculate the residuals between the real data and the smoothed approximation. Now, an isolated large residual is certainly odd: it suggests that the corresponding data point is somehow “different” than the other points in the neighborhood — in other words, an outlier. Now we argue as follows: if the data point is an outlier, it should contribute less to the smoothed curve than other points. Taking this consideration into account, we now introduce an additional weight factor for each data point into the expressions for $J[s]$ or χ^2 given earlier. With this new weight factor reducing the influence of points with large residuals, we calculate a *new* version of the smoothed approximation. This process is now iterated until the smooth curve no longer changes.

Another idea is to split the original data points into two classes: those that give raise to a positive residual and those that have a negative residual. Now calculate a smooth curve for each class separately. The resulting curves can be interpreted as “confidence bands” for the data set (meaning that the majority of points will lie between the upper and the lower smooth curve). We are particularly interested to see whether the width of this band varies along the curve. Figure 3.7 shows an example, using the men’s results from figure 3.5.

Personally, I am a bit uncomfortable with either of these suggestions: they

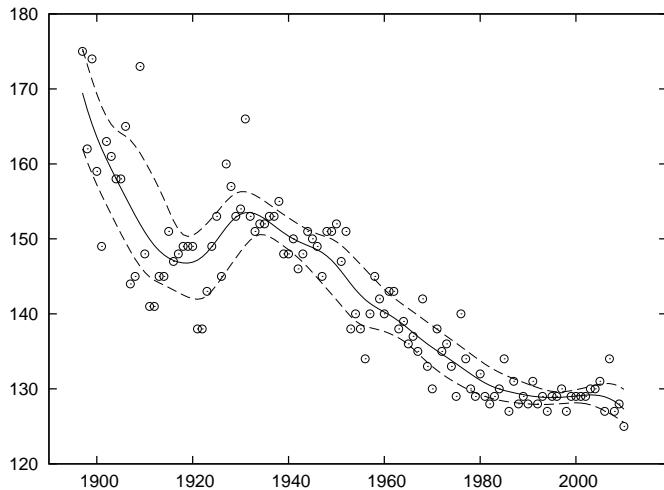


Figure 3.7: TBD

certainly have an unpleasant air of circular reasoning about them.

But there is a deeper reason as well: in my opinion, smoothing methods are a quick and useful, but entirely non rigorous way to explore the structure of a data set. With some of the more sophisticated extensions (such as the two suggestions discussed a minute ago), we let go of the simplicity of the approach, without gaining anything in rigor! If we need or want better (or deeper) results than simple graphical methods can give us: isn't it time to consider a more rigorous toolset?

This is a concern I have with many of the more sophisticated graphical methods that you will find discussed in the literature. Yes, we certainly *can* squeeze ever more information into a graph, using lines, colors, symbols, textures, and what have you. But this does not mean that we *should*. The primary benefit of a graph is that it speaks to us directly — without the need for formal training or long explanations. Graphs that require training or complicated explanations to be properly understood are missing their mark, no matter how "clever" they may be otherwise.

Similarly for some of the more involved ways of graph preparation: a smooth curve such as a spline or LOESS approximation is a very rough approximation to the data set — and by the way contains a huge degree of arbitrariness, in the form of the smoothing parameter α or d , respectively! Given this situation, it is not clear to me that we need to worry about details such as the effect of individual outliers on the curve.

Focusing too much on graphical methods may also lead us to miss the essential point: once we start worrying about confidence bands for example, we should really start *thinking more deeply* about the nature of the local distribution

of residuals (Are the residuals normally distributed? Are they independent? Do we have a reason to prefer one statistical model over another?), and possibly consider a more reliable estimation method (for example bootstrapping, see chapter 12), rather than blundering along using hand-waving (semi-)graphical methods.

Remember: The purpose of computing is insight, not pictures! (L. N. Trefethen)

3.3 Logarithmic Plots

Logarithmic plots are a standard tool of scientists, engineers, and ... stock analysts everywhere. They are so popular, because they have three invaluable benefits:

- They rein in large variations in the data.
- They turn multiplicative variations into additive ones.
- They reveal exponential and power law behavior.

In a logarithmic plot, we plot the *logarithm* of the data, instead of the raw data. Most plotting programs can do this for us (so that we don't have to transform the data explicitly) and also take care of labeling the axes appropriately.

There are two forms of logarithmic plots: *single* or *semi-logarithmic* plots and *double logarithmic* or *log-log* plots, depending whether only one (usually the vertical or y-axis) or both axes have been scaled logarithmically.

All logarithmic plots are based on the fundamental property of the logarithm to turn products into sums and powers into products:

$$\begin{aligned}\log(xy) &= \log(x) + \log(y) \\ \log(x^k) &= k \log(x)\end{aligned}$$

Let's consider a semi-log plots first. Imagine you have data that has been generated by evaluating the function

$$y = C \exp(\alpha x) \quad \text{where } C \text{ and } \alpha \text{ are constants}$$

on a set of x values. If you plot y as a function of x , you will see an upwards or downwards sloping curve, depending on the sign of α (see B). But if you instead plot the logarithm of y as a function of x , the points will fall on a straight line. This can be easily understood by applying the logarithm to the equation above:

$$\log y = \alpha x + \log C$$

In other words, the logarithm of y is a linear function of x , with slope α and offset $\log C$. In particular, by measuring the slope of the line we can determine the scale factor α , which is often of great interest in applications.

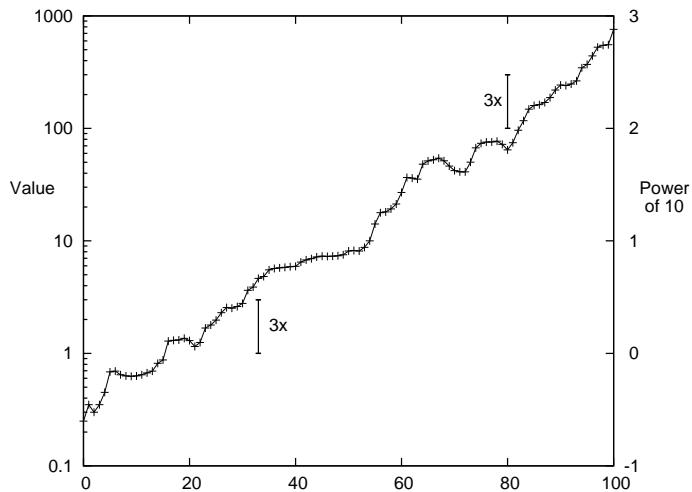


Figure 3.8: A semi-logarithmic plot.

Figure 3.8 shows an example of a semi-logarithmic plot, containing both some experimental data points as well as an exponential function for comparison. I'd like to point out a few details: in a logarithmic plot, we plot the logarithm of the values, but the axes are usually labeled with the actual values (not their logarithms). Figure 3.8 shows both: the actual values on the left, and the logarithms on the right (the logarithm of 100 to base 10 is 2, the logarithm of 1000 is 3, and so on). We can see how in a logarithmic plot, the logarithms are equidistant, but the actual values are not. (Note how the distance between consecutive tick marks is constant on the right, but not on the left.)

Another aspect I want to point out is how on a semi-log plot all *relative* changes have the same size, no matter how large the corresponding absolute change. It is this property that makes semi-log plots popular for long-running stock charts and the like: if you lost \$100, your reaction may be quite different if originally you had \$1000 or \$200: in the first case you lost 10 percent, but you lost 50 percent in the latter. In other words, relative change is what matters.

The two scale arrows in figure 3.8 have the same length, and correspond to the same relative change, but the underlying absolute change is quite different (1 to 3 in one case, 100 to 300 in the other)! This is another application of the fundamental property of the logarithm: if the value before the change is y_1 and $y_2 = \gamma y_1$ after the change, where $\gamma = 3$, then the change in absolute terms is:

$$y_2 - y_1 = \gamma y_1 - y_1 = (\gamma - 1)y_1$$

which clearly depends on y_1 . But if we consider the change in the logarithms, we find:

$$\log y_2 - \log y_1 = \log(\gamma y_1) - \log y_1 = \log \gamma + \log y_1 - \log y_1 = \log \gamma$$

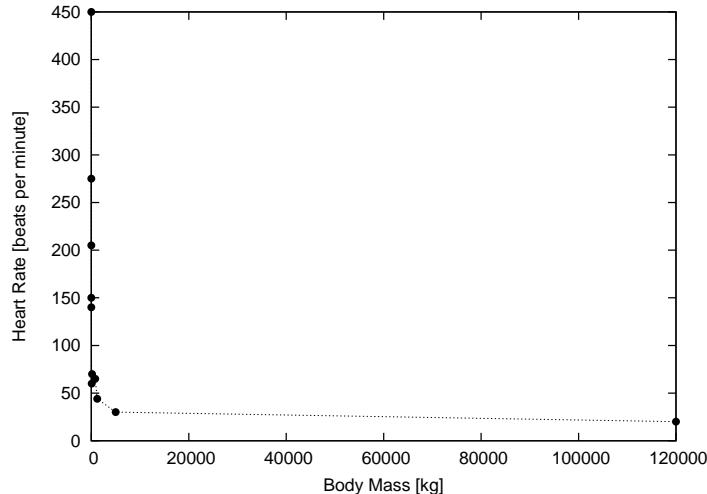


Figure 3.9: Heart rate versus body mass for a range of mammal. Compare figure 3.10.

which is independent of the underlying value and only depends on the size of the relative change, γ .

Double logarithmic plots are now easy to understand — the only difference is that we plot logarithms of both x and y . This will render all power law relations as straight lines, that is functions of the form $y = Cx^k$ or $y = C/x^k$, where again C and k are constants. (Take logarithms on both sides of the first equation and you find: $\log y = k \log x + \log C$, so that now $\log y$ is a linear function of $\log x$, with a slope depending on the exponent k .)

Figures 3.9 and 3.10 provide a pretty stunning example for both uses of double logarithmic plots: their ability to make data spanning many order of magnitude accessible and their ability to reveal power law relationships by turning them into straight lines. Figure 3.9 shows the typical resting heart rate (in beats per minute) as a function of the body mass (in kilograms) for a selection of mammals, from the hamster to large whales. Whales weigh in at 120 tons — nothing else even comes close! The consequence is that almost all of the data points are squished against the lefthand side of the graph: literally crushed by the whale.

On the double logarithmic plot, the distribution of data points becomes much clearer. Moreover, we find that the data points are not randomly distributed, but seem to fall roughly on a straight line, with slope $-1/4$: the signature of power law behavior. In other words, a mammal's typical heart rate is related to its mass! Larger animals have slower heart beats. If we let f be the heart rate and m be the mass, we can summarize this observation as:

$$f \sim m^{-1/4}$$

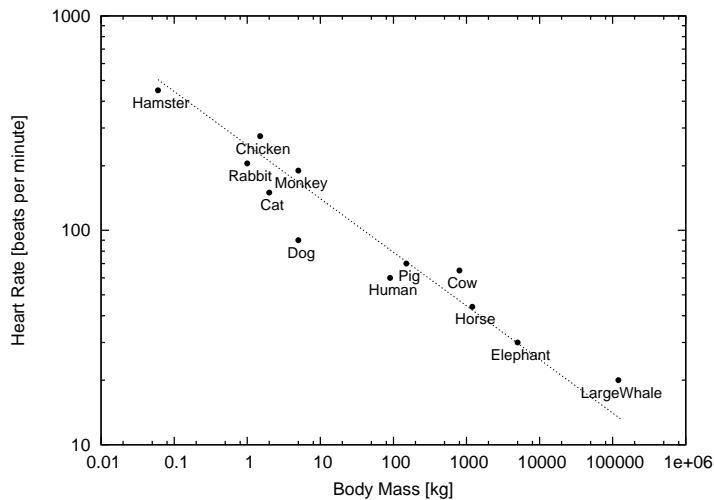


Figure 3.10: The same data as in figure 3.9, but now plotted on a double logarithmic plot. The data points seem to fall on a straight line, indicating a power law relationship between resting heart rate and body mass.

This surprising finding is known as *allometric scaling*, and seems to hold more generally, not just for the specific animals and quantities shown in the figures. (For example, it turns out that the lifetime of an individual organism also obeys a 1/4 power law relationship with the body mass: bigger animals live longer. The surprising consequence is that the total number of heartbeats per life of an individual is approximately constant for all species!) Allometric scaling has been explained in terms of the geometric constraints that the vascular network (the veins and arteries) which brings nutrients to the cells making up a biological system: it is sufficient to assume that the network must be a space-filling fractal, that the capillaries where the actual exchange of nutrients takes place are the same size in all animals, and that the overall energy required for transport through the network is minimized, to derive the power law relationships observed experimentally!³ We'll have more to say about scaling laws and their uses in part II.

3.4 Banking

Smoothing methods and logarithmic plots are both tools that help us to recognize structure in a data set: smoothing methods reduce noise, and logarithmic

³The original reference is: “A General Model for the Origin of Allometric Scaling Laws in Biology” by G. B. West, J. H. Brown, B. J. Enquist in the journal “Science” (Volume 276, page 122 (1997)). Additional references can be found on the web.

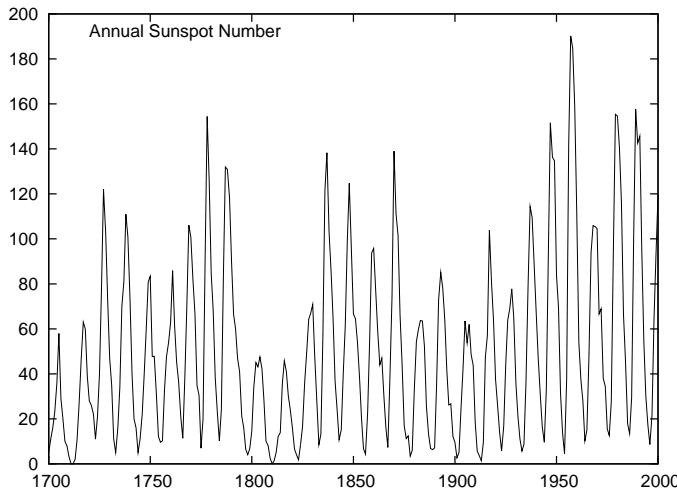


Figure 3.11: The annual sunspot numbers for the last 200 years. The aspect ratio of the plot makes it hard to recognize details of each cycle.

plots help with data sets spanning many orders of magnitude.

Banking (or “banking to 45 degrees”) is another graphical method. It is different than the preceding ones, because it does not work on the *data*, but on the plot as a whole, by changing its aspect ratio.

We can recognize *change* (that is: the slopes of curves) most easily if they make approximately a 45 degree angle on the graph. It is much harder to see change if the curves are nearly horizontal or (even worse) nearly vertical. The idea behind *banking* is therefore to adjust the aspect ratio of the whole plot in such a way that most slopes are at an approximate 45 degree angle.

Chances are, you have been doing this already by changing the plot *ranges*. Often when we “zoom” in on a graph, we don’t do this so much to see more detail, but to adjust the slopes of curves to become more easily recognizable. The purpose is even more obvious when we zoom *out*. Banking helps us recognize what it is we have been doing, and opens up a way to control the appearance of a plot by actively adjusting the aspect ratio.

Figures 3.11 and 3.12 show the classical example for this technique: the annual number of sunspots, measured over the last 300 years.⁴ In figure 3.11, the oscillation is very compressed, and it is difficult to make out much detail about the shape of the curve. In figure 3.12, the aspect ratio of the plot has been adjusted so that most line segments are now at roughly a 45 degree angle, and we can see something interesting, namely that the raising edge of each sunspot cycle is steeper than the falling edge. We would probably not have recognized

⁴The discussion here is adapted from my book “Gnuplot in Action”, Manning Publications, 2010.

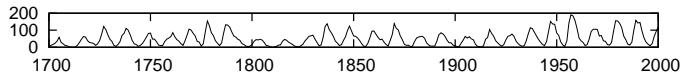


Figure 3.12: The same data as in figure 3.11. The aspect ratio has been changed so that raising and falling flanks of the curve make approximately a 45 degree angle with the horizontal (banking to 45 degrees). However, the figure has become so small that it is hard to recognize much detail.

this by looking at figure 3.11.

Personally, I would probably not use a graph such as figure 3.12: by shrinking the vertical axis down to almost nothing, we lose too much detail. It also becomes difficult to compare the behavior on the far left and far right of the graph. Instead, I would break up the time series and plot it as a *cut-and-stack plot*, such as the one in figure 3.13. Note that in this plot, the aspect ratio of each sub-plot is such that the lines are in fact banked to 45 degrees.

As this example demonstrates, banking is a good technique, but it can be taken too literally. Generally, when the aspect ratio required to achieve proper banking is too skewed, it is better to rethink the entire graph. No amount of banking will make the data set in figure 3.9 look right — you need a double-logarithmic transform.

There is also another aspect. The purpose of banking is to improve human perception of the graph (it is, after all, exactly the same data that is displayed!). But graphs with very skewed aspect ratios violate the great affinity humans seem to have to proportions of roughly 4:3 (or 11 by 8.5 or $\sqrt{2}$ by 1). I would not do so needlessly — just witness the abundance of display formats (paper, books, screens) that roughly adhere to these proportions the world over. Whether we favor this display format because we are so used to it, or (more likely, I think) it is so predominant because it works well for humans, is rather irrelevant in this context. (And keep in mind that squares seem to work particularly badly — notice how squares, when used for furniture or appliances, are considered a “bold” design. Unless there is a good reason for them (such as graphing a square matrix), I recommend you try to avoid them.)

3.5 Linear Regression and all that

Linear regression is a method to find a straight line through a two-dimensional scatter plot. It is very simple to calculate and has considerable intuitive appeal — both of which together make it easily the single most-often misapplied technique in all of statistics!

There is a fundamental misconception regarding linear regression, namely that it is a good and particularly rigorous or significant way to *summarize* the data in a two-dimensional scatter plot. This misconception is often associated with the notion that linear regression provides the “best fit” to the data.

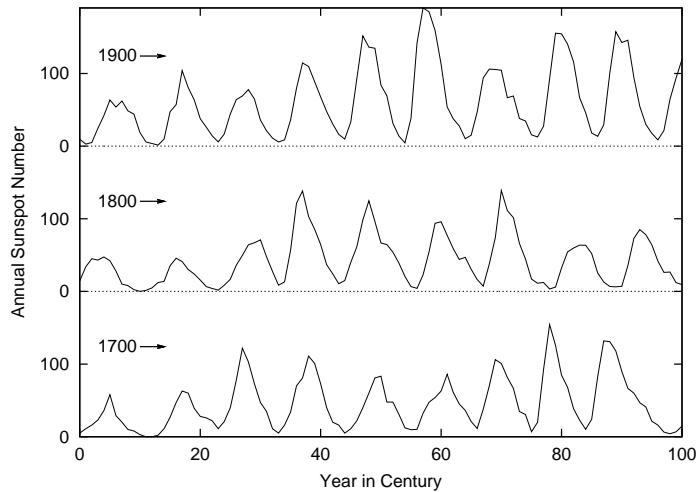


Figure 3.13: A cut-and-stack plot of the data from figure 3.11. By breaking the time axis into three chunks, we can bank each century to 45 degrees and still fit all the data into a standard-size plot. Note how we can now easily recognize an important aspect of the data: the raising flank tends to be steeper than the falling one.

This is not so. Linear regression is not a particularly good way to summarize data, and it provides a “best fit” in a much more limited sense than is generally realized.

Linear regression applies to situations where we have a set of input values (the controlled variable) and for each of them, we measure an output value (the response variable). Now we are looking for a linear function $f(x) = a + bx$ as a function of the controlled variable x that reproduces the response with the least amount of error. The result of a linear regression is therefore a function which minimizes the error in the responses for a given set of inputs.

This is an important understanding: the purpose of a regression procedure is *not* to summarize the data — the purpose is to obtain a function which allows us to *predict* the value of the response variable (which is affected by noise) that we expect for a certain value of the input variable (which is assumed to be known exactly).

As you can see, there is a fundamental asymmetry between the two variables: the two are not interchangeable. In fact, you will obtain a *different* solution if you regress x on y than when you regress y on x . This simple observation should dispel the notion that linear regression provides *the best fit* — how could there be two different “best fits” for a single data set? Instead, linear regression provides the most faithful representation of an output in response to an input. In other words, *linear regression is not so much a best fit, but a best*

predictor.

How do we find this “best predictor”? We require it to minimize the error in the responses. But the error in the responses is simply the sum over the errors for all the individual data points. Because errors can be positive or negative (the function over- or under-shoots the real value), they may cancel each other. To avoid this, we do not sum the errors themselves, but their squares (here, (x_i, y_i) with $i = 1 \dots n$ are the data points):

$$\begin{aligned}\chi^2 &= \sum_i (f(x_i) - y_i)^2 \\ &= \sum_i (a + bx_i - y_i)^2\end{aligned}$$

Using the values for the parameters a and b that minimize this value will give a function that best explains y in terms of x .

Because the dependence of χ^2 on a and b is particularly simple, we can work out expressions for the optimal choice of both parameters explicitly. The results are:

$$\begin{aligned}b &= \frac{n \sum x_i y_i - (\sum x_i)(\sum y_i)}{n(\sum x_i^2) - (\sum x_i)^2} \\ a &= \frac{1}{n} (\sum y_i - b \sum x_i)\end{aligned}$$

These results are simple and beautiful, and in their simplicity very suggestive. But they can also be highly misleading. Table 3.1 and figure 3.14 show a famous example: “Anscombe’s Quartet”. If you go and calculate regression coefficients a and b for each of the four data sets shown in table 3.1, you will find that they are exactly the same for all four data sets! Yet, if you look at the corresponding scatter plots, it is clear that only the first data set is described properly by the linear model: the second data set is not linear, the third contains an outlier, and the fourth does not contain enough independent x values to form a regression at all! Looking only at the results of the linear regression, you would never know.

I think this example should demonstrate once and for all how dangerous it can be to rely on linear regression (or in fact any form of aggregate statistics) to summarize a data set. (In fact, the situation is even worse than what I have presented: with a little bit more work, you can calculate confidence intervals on the linear regression results, and even *they* turn out to be equal for all four members of Anscombe’s quartet!)

Having seen this, here are some questions to ask *before* computing linear regressions:

Do you need regression? Remember that regression coefficients are not a particularly good way to *summarize* data. Regression only makes sense when you want to use it for *prediction*. If this is not the case, calculating regression coefficients is not useful.

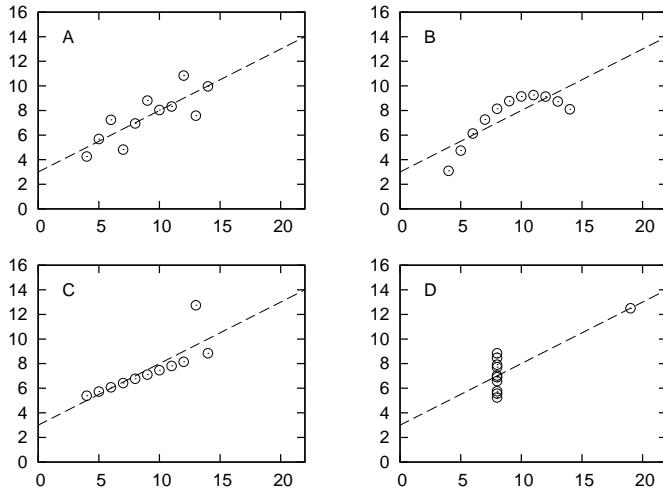


Figure 3.14: Anscombe's Quartet: all summary statistics and in particular the regression coefficients for all four data sets are numerically equal, yet only data set A is well represented by the linear regression function.

A		B		C		D	
x	y	x	y	x	y	x	y
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

Table 3.1: Anscombe's Quartet

Is the linear assumption appropriate? Linear regression is only appropriate if the data can somehow be described by a straight line. If this is obviously not the case (such as in the second data set in Anscombe's quartet), then linear regression does not apply.

Is something else entirely going on? Linear regression, like all summary statistics, can be led astray by outliers or other "weird" data sets, as is demonstrated by the last two examples in Anscombe's quartet.

Historically, one of the attractions of linear regression has been that it is easy to calculate: all you need to do is to calculate the four sums $\sum x_i$, $\sum x_i^2$, $\sum y_i$ and $\sum x_i y_i$, which can be done in a single pass through the data set. Even with moderately sized data sets (dozens of points), this is arguably easier than plotting them using paper and pencil! However, in this day and age, that argument simply does not hold anymore: graphs are easy to produce and contain so much more information than a set of regression coefficients that they should be the preferred way to analyze, understand, and summarize data!

Remember: The purpose of computing is insight, not numbers! (R. W. Hamming)

3.6 Showing What's Important

Maybe this is a good moment to express what I believe to be the most important principle in graphical analysis:

Plot the pertinent quantities!

Obvious as it may appear, this principle is often overlooked in practice.

For example, if you look through one of those books that show and discuss examples of *bad* graphics, you will find that most examples fall into one of two classes: there are those graphs that failed *visually*, with garish fonts, unhelpful symbols, and useless embellishments. (These are mostly presentation graphics gone wrong, not examples of bad graphical analysis.)

The other large class of graphical failures consists of those plots that failed *conceptually* or I might better say *analytically*. The problem with these is not in the technical aspects of drawing the graph, but in the conceptual understanding of what the graph is trying to show: they showed something, but they failed to show what was most *pertinent* to the question at hand.

The problem, of course, is that it is usually not at all obvious what it is that we want to see, and it is certainly not obvious at the beginning. It usually takes several iterations, while a mental model of the data is forming in your head, to articulate the proper question that a data set is suggesting, and to come up with the best way of answering it. Usually, this involves some form of transformation or manipulation of the data: instead of the raw data, maybe we should show the difference between two data sets. Or the residual after subtracting a trend or the results from a model. Or maybe we need to normalize data sets from different sources by subtracting their means and dividing by their spreads. Or maybe we should not use the original variables to

display the data, but apply some form of transformation on them (logarithmic scales are only the simplest example of such transformations). Whatever it is, it will typically involve some form of transformation on the data — it's rarely the raw data that is most interesting; but any deviation from the expected is almost always an interesting discovery.

Very roughly, I think we can identify a three- (maybe four-) step process. Not in the sense of a prescriptive checklist, but in the sense of a gradual process of learning and discovery.

First: The Basics Initially, we are mostly concerned with displaying what is there.

- Select proper ranges.
- Subtract a constant offset.
- Decide whether to use symbols (for scattered data), or lines (for continuous data), or maybe both (for scarce data sets: connecting individual symbols can help emphasize trends in sparse data sets).

Second: The Appearance Next, we work with aspects of the plot that influence its overall appearance.

- Log plots.
- Add a smoothed curve.
- Consider banking.

Third: Build a Model At this point, we start building a mathematical model and try to compare it against the raw data. The comparison often involves finding the differences between the model and the data (typically subtracting the model or forming a ratio).

- Subtract a trend.
- Form the ratio to a base value or base line.
- Rescale a set of curves to collapse them onto each other.

Fourth (only for Presentation Graphics): Add Embellishments Embellishments and decorations (labels, arrows, special symbols, explanations, and so on) can be very helpful in making a graph informative and self-explanatory. However, they are intended for an audience beyond the actual creator of the graph. You will rarely need them during the *analysis* phase when you are trying to find out something new about the data set, but they are an essential part when *presenting* your results. This step only occurs if you want to communicate your results to a wider and more general audience.

3.7 Graphical Analysis and Presentation Graphics

I have used the terms *graphical analysis* and *presentation graphics* above, without explaining them properly. In short:

Graphical Analysis Graphical analysis is an investigation of data using graphical methods. The purpose is the discovery of *new* information about the underlying data set. In graphical analysis, the “proper” question to ask is often not known from the outset, but is discovered as part of the analysis.

Presentation Graphics Presentation graphics are concerned with the communication of information and results that are *already understood*. The discovery has been made, now it merely needs to be communicated clearly.

The distinction between these two activities is important, because they do require different techniques and yield different work products.

During the analysis process, convenience and ease of use are the predominant concerns — any amount of polishing is too much! Nothing should keep you from redrawing a graph, changing some aspect of it, zooming in or out, applying transformations, and changing styles. (When I am working with a new data set that I haven’t seen before, I probably create dozens of graphs within a few minutes: basically “looking at the data from all angles”.) Any form of embellishment (labels, arrows, special symbols) is inappropriate — you know what it is you are showing, and creating any form of decoration on the graph will only make you more reluctant to throw the graph away and start over.

For presentation graphics, the opposite applies. Now, you already know the results, but you would like to communicate them to others. Textual information therefore becomes very important: how else will people know what it is they are looking at?

You will find plenty of advice out there on how to prepare “good” presentation graphics, often strongly worded and with an altogether unfortunate tendency to use emotional responses (ridicule or derision) in place of factual arguments.

In the absence of good empirical evidence one way or the other, I am not going to add to the discussion. But I present a *checklist* below, listing some points often overlooked when preparing graphs for presentation.

- Try to make the text self-explanatory. Don’t rely on a (separate) caption for basic information — it might get removed during reproduction. Place basic information on the graph itself.
- Explain what is plotted on the axis. This can be done with explicit labels on the axes, or through explanatory text elsewhere. Don’t forget units!
- Make labels self-explanatory. Be careful with non-standard abbreviations. Ask yourself: If that’s all the context the reader has, are you *certain* that the reader will be able to figure out what you mean? (In a recent book on data graphing, I found a histogram labeled *Married*, *Nvd*, *Dvd*,

Spd, and *Wdd*. I could figure out most of them, because at least *Married* was given in clear, but *Nvd* had me struggle for quite a while!)

- Given how important *text* is on a graph, make sure to pick a suitable font. Don't automatically rely on the default provided by your plotting software. Generally, Sans-Serif fonts (such as Helvetica) are preferred for short labels (such as those on a graph), while Serif fonts (such as Times) are more suitable for body text. Also pick a suitable size — text fonts on graphics are often too large, making them look particularly garish. (Text font is usually 10 to 12 points. There is no need for font on graphics to be significantly larger.)
- If there are error bars, make sure to explain their meaning. What are they? Standard deviations? Inter-quartile ranges? Or the limits of experimental apparatus? Also: choose an appropriate measure of uncertainty. Don't use standard deviations for highly skewed data.
- Don't forget the basics. Choose appropriate plot ranges. Make sure data is not unnecessarily obscured by labels.
- Proof-read graphs! Particularly common sources of error include: typos in textual labels, interchanged data sets or switched labels, missing units, and incorrect order-of-magnitude qualifiers (such as milli-, micro-, and so on).
- Lastly: choose an appropriate output format for your graph! Don't use bitmap formats (GIF, JPG, PNG) for print publication — use a scalable format such as PostScript or PDF.

One final piece of advice: creating good presentation graphics is also a matter of *taste*, and taste can be *acquired*. If you want to work with data, you should get interested in graphs — not just the ones you create yourself, but all that you see. If you notice one that seem to work (or that doesn't!), give it a moments thought to figure out what makes it so. Are the lines too thick? The labels too small? The choice of colors just right? The combination of curves helpful? Details matter.

3.8 Workshop: matplotlib

The `matplotlib` module is a Python module to create two-dimensional xy-plots, scatter plots, and other plots typical for scientific applications. It can be used both in an interactive session (with the plots being shown immediately in a GUI window), or from within a script to create a file in a common file format.

Let's first look at some examples to demonstrate how `matplotlib` can be used from within an interactive session. Afterwards, we will take a closer look at the structure of the library and give some pointers for more detailed investigations.

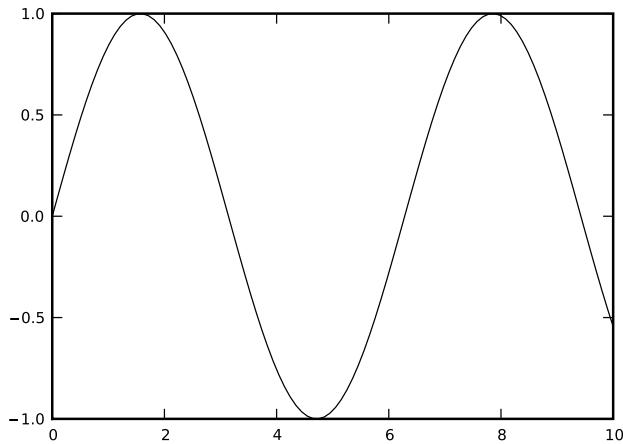


Figure 3.15: TBD

3.8.1 Using Matplotlib Interactively

To start an interactive matplotlib session, start IPython (the enhanced interactive Python shell) with the `-pylab` option, executing the following command line like at the shell prompt:

```
ipython -pylab
```

This will start IPython, load matplotlib *and* NumPy and import both into the global namespace. The idea is to give a Matlab-like experience of interactive graphics together with numerical and matrix operations. (It is important to use IPython here — the flow of control between the Python command interpreter and the GUI eventloop for the graphics windows requires it. Other interactive shells can be used, but may require some tinkering.)

We can now create plots right away:

```
In [1]: x = linspace( 0, 10, 100 )
In [2]: plot( x, sin(x) )
Out[2]: [

```

This will pop up a new window, showing a graph like the one in figure 3.15, but decorated with some GUI buttons. (Note that the `sin()` function is a ufunc from the NumPy package: it takes a vector and returns a vector of the same size, having applied the sine function to each element in the input vector. See the Workshop in chapter 2.)

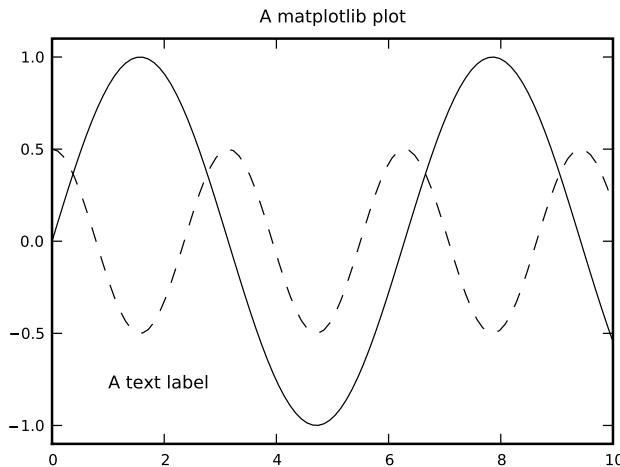


Figure 3.16: TBD

We can now add additional curves and decorations to the plot. Continuing in the same session as before, we add another curve and some labels:

```
In [3]: plot( x, 0.5*cos(2*x) )
Out[3]: <matplotlib.lines.Line2D object at 0x1cee8d0>

In [4]: title( "A matplotlib plot" )
Out[4]: <matplotlib.text.Text object at 0x1cf6950>

In [5]: text( 1, -0.8, "A text label" )
Out[5]: <matplotlib.text.Text object at 0x1f59250>

In [6]: ylim( -1.1, 1.1 )
Out[6]: (-1.1000000000000001, 1.1000000000000001)
```

In the last step, we also increased the range of values plotted on the vertical axis. (There is also an `axis()` command, which allows you to specify limits for both axes at the same time. Don't confuse it with the `axes()` command, which creates a new coordinate system. See below.) The plot should now look like the one in figure 3.16 (except that in an interactive terminal the different lines are distinguished by their color, not their dash pattern).

Let's pause for a moment and point out a few details. First of all, you should have noticed that the graph in the plot window was updated after every operation. That is typical for the interactive mode, but is not how matplotlib works in a script: in general, matplotlib tries to defer the (possibly expensive) creation

of an actual plot until the last possible moment. (In a script, you would use the `show()` command to force generation of an actual plot window.)

Furthermore, matplotlib is “stateful”: a new plot command does not erase the previous figure, but adds to it. This behavior can be toggled with the `hold()` command; the current state can be queried using `ishold()`. (Decorations like the text labels are not affected by this.) You can clear a figure explicitly using `clf()`.

This implicit state may come as a surprise: haven’t we learned to make things explicit, when possible? In fact, this stateful behavior is a hold-over from the way Matlab works. Here is another example. Start a new session and execute the following commands:

```
In [1]: x1 = linspace( 0, 10, 40 )

In [2]: plot( x1, sqrt(x1), 'k-' )
Out[2]: [<matplotlib.lines.Line2D object at 0x1cfef50>]

In [3]: figure(2)
Out[3]: <matplotlib.figure.Figure object at 0x1cee850>

In [4]: x2 = linspace( 0, 10, 100 )

In [5]: plot( x1, sin(x1), 'k--', x2, 0.2*cos(3*x2), 'k:' )
Out[5]:
[<matplotlib.lines.Line2D object at 0x1fb1150>,
 <matplotlib.lines.Line2D object at 0x1fba250>]

In [6]: figure(1)
Out[6]: <matplotlib.figure.Figure object at 0x1cee210>

In [7]: plot( x1, 3*exp(-x1/2), linestyle='None', color='white', marker='o', markersize=10 )
Out[7]: [<matplotlib.lines.Line2D object at 0x1d0c150>]

In [8]: savefig( 'graph1.png' )
```

This snippet of code demonstrates several things. We begin as before, by creating a plot. But this time, we pass a third argument to the `plot()` command, which controls the appearance of the graph elements. Matplotlib supports Matlab-style mnemonics for plot styles; the letter `k` stands for the color “black” and the single dash `-` for a solid line.

Then we create a *second* figure in a new window, and switch to it, using the `figure(2)` command. All graphics commands will now be directed to this second figure — until we switch back to the first figure using `figure(1)`. This is another example of “silent state”. Also note that figures are counted starting from 1, not 0.

In line five, we see another way to use the `plot` command, namely by specifying two sets of curves to be plotted together. (The formatting commands

request a dashed and a dotted line, respectively.) Line seven shows yet a different way to specify plot styles, namely using named (keyword) arguments.

Finally, we save the currently active plot (that is: figure 1) to a PNG file. The `savefig()` function determines the desired output format from the extension of the filename given. Other formats that are supported out of the box are PostScript, PDF, and SVG. Additional formats may be available, depending on the libraries installed on your system.

3.8.2 Managing Properties

Up to now, we have ignored the values returned by the various plotting commands. If you look at the output generated by IPython, you can see that all the commands that add graph elements to the plot return a reference to the object just created. The one exception is the `plot()` command itself, which always returns a *list* of objects (because, as we have seen, it can add more than one “line” to the plot).

These references are important, because through them we can control the appearance of graph elements once they have been created. In a final example, let’s study how we can use them.

```
In [1]: x = linspace( 0, 10, 100 )
In [2]: ps = plot( x, sin(x), x, cos(x) )
In [3]: t1 = text( 1, -0.5, "Hello" )
In [4]: t2 = text( 3, 0.5, "Hello again" )
In [5]: t1.set_position( [7, -0.5] )
In [6]: t2.set( position=[5, 0], text="Goodbye" )
Out[6]: [None, None]
In [7]: draw()
In [8]: setp( [t1, t2], fontsize=10 )
Out[8]: [None, None]
In [9]: t2.remove()
In [10]: Artist.remove( ps[1] )
In [11]: draw()
```

In the first four lines, we create a graph with two curves and two text labels as we have done before, but now we are holding on to the object references.

This allows us to make changes to these graph elements. Lines 5, 6, and 8 demonstrate different ways to do this: for each property of a graph element, there is an explicit, named accessor function (line 5). Alternatively, we can use a generic setter with keyword arguments — this allows us to set several properties (but on a single object) in a single call (line 6). Finally, we can use the standalone `setp()` function, which takes a list of graph elements and applies the requested property update to all of them. (It can also take a single graph element, instead of a one-member list.) Notice that `setp()` generates a redraw event, whereas individual property accessors do not: hence the need to generate an explicit redraw event in line 7. Finally, we remove one of the text labels and one of the curves, using the `remove()` function. The `remove()` function is defined for objects that are derived from the `Artist` class and we can therefore either invoke it using member syntax (as a “bound” function, line 9) or using the class syntax (as an “unbound” function, line 10). Keep in mind that `plot()` returns a *list* of objects.

There are some useful functions that can help us handle object properties: if you issue `setp(r)` with only a single argument in an interactive session, it will print all properties that are available for object `r`, together with information about the values that each property is allowed to take on. The `getp(r)` function, on the other hand, prints all properties of `r`, together with their current values.

In case we did not save the references to the objects that we created, or if we want to change the properties on an object that we did not create explicitly, we can use the functions `gcf()` and `gca()`, which return a reference to the current figure or axes object, respectively. To make use of them, we need to develop at least a passing familiarity with matplotlib’s object model.

3.8.3 Case Study: LOESS with Matplotlib

As a quick example how to put the different aspect of Matplotlib together, let’s discuss the script I used to generate figure 3.4. This also gives us another opportunity to look at the LOESS method in a bit more detail.

To recap: LOESS stands for *locally weighted* linear regression. The difference to regular linear regression is that we introduce a weight factor, which emphasizes those data points, which are close to the location x at which we want to evaluate the smoothed curve. As explained earlier, the expression for squared error, which we want to minimize, now becomes:

$$\chi^2(x) = \sum_i w(x - x_i; h) (a + bx_i - y_i)^2$$

Keep in mind that this expression now depends on x , the location at which we want to evaluate the smoothed curve!

If we minimize this expression with respect to the parameters a and b , we obtain the following expressions for a and b (remember that we will have to evaluate them from scratch for every point x):

$$b = \frac{\sum w_i \sum w_i x_i y_i - (\sum w_i x_i) (\sum w_i y_i)}{\sum w_i (\sum w_i x_i^2) - (\sum w_i x_i)^2}$$

$$a = \frac{(\sum w_i y_i - b \sum w_i x_i)}{\sum w_i}$$

This can be quite easily translated into NumPy and plotted with Matplotlib — the code is in listing 3.1. The actual LOESS calculation is entirely in the function `loess()`. (Check the Workshop in chapter 2 for a discussion of this type of programming.)

We evaluate the smoothed curve at the location of all data points, using two different values for the bandwidth, and then proceed to plot the data, together with the smoothed curves. Two details require an additional word of explanation. The function `gca()` returns the current “set of axes” (that is: the current coordinate system on the plot — see below for more information on `gca()`), and we require the aspect ratio of both x- and y-axis to be equal (so that the plot is a square). And in the last command before we save the figure to file, we adjust the plot range, using the `axis()` command: this function has to come *after* the `plot()` commands, because they automatically adjust the plot range depending on the data.

3.8.4 The Matplotlib Object Model and Architecture

Matplotlib’s object model is constructed similar to the object model for a GUI widget set: a plot is represented by a tree of widgets, with each widget being able to render itself. Perhaps surprisingly, the object model is not flat. In other words, all elements of a plot, such as axes, labels, arrows, and so on, are not properties of a high-level “plot” or “figure” object. Instead, you need to descend down the object tree to find the element that you want to modify, and then change the appropriate property on the element once you have an explicit reference to it.

The top-level element (the root node of the tree) is an object of class `Figure`. A figure contains one or more `Axes` objects: this class represents a “coordinate system” on which actual graph elements can be placed. (If you are looking for the actual axes that are drawn on the graph, you want to look for objects of the `Axis` class!) The `gcf()` and `gca()` functions therefore return a reference to the root node of the entire figure, or to the root node of a single plot in a multi-plot figure.

Both `Figure` and `Axes` are subclasses of `Artist`. This is the base class of all “widgets” that can be drawn onto a graph. Other important subclasses of `Artist` are `Line2D` (a polygonal line, connecting multiple points, optionally with a symbol at each point), `Text`, and `Patch` (a geometric shape that can be placed onto the figure). The top-level `Figure` instance is owned by an object of type `FigureCanvas` (in the `matplotlib.backend_bases` module). Most likely you won’t have to interact with this class yourself directly, but it provides the

“bridge” between the (logical) object tree that makes up the graph and a “backend”, which does the actual rendering. Depending on the backend, matplotlib creates a file or a graph window that can be used in an interactive GUI session.

Although it is easy to get started with matplotlib from within an interactive session, it can be quite challenging to really get one’s arms around the whole library. This can become painfully clear when one wants to change some tiny aspect of a plot — and can’t find out *how* to do that.

As so often, it helps to investigate how things came to be. Matplotlib was conceived as a plotting library to emulate the behavior found in Matlab. Matlab traditionally uses a programming model based on functions, and (being 30 years old) employs some conventions that are no longer popular (such as implicit state). Matplotlib, on the other hand, was implemented using object-oriented design principles in Python, with the consequence that these two different paradigms frequently clash.

One consequence of having these two different paradigms side by side is redundancy. Many operations can be performed in several different ways (using standalone functions, using Python-style keyword arguments, using object attributes, or using a Matlab-compatible alternative syntax). We have seen examples of this redundancy in the third example when we changed object properties. This matters, because it drastically increases the size of the API, making it that much harder to develop a comprehensive understanding. What is worse, it tends to spread information around (where should I be looking for plot attributes — among functions, among members, among keyword attributes? Answer: everywhere!).

Another consequence is inconsistency. Matplotlib (at least in its favored function-based interface) uses some conventions that are rather unusual for Python programming: look at the way a figure is created *implicitly* at the beginning of every example, and how the pointer to the current figure is maintained through an invisible “state variable”, which is opaquely manipulated using the `figure()` function. (The `figure()` function actually returns the figure object just created, therefore the invisible state variable is not even necessary.) Similar surprises can be found throughout the library.

A last problem is namespace pollution (this is another Matlab heritage — they didn’t have namespaces back then!). Matplotlib includes a range of operations into its function-based interface, that are not actually graphics related, but that generate plots as *side effects*. For example, `hist()` calculates (and plots) a histogram, `acorr()` calculates (and plots) an autocorrelation function, and so on. From a user’s perspective, it makes more sense to adhere to a separation of concerns: do all calculations in NumPy/SciPy, and pass the results explicitly to matplotlib for plotting.

3.8.5 Odds and Ends

There are three different ways to import and use matplotlib. The original method was to use:

```
from pylab import *
```

This would load all of NumPy as well as matplotlib and import both APIs into the global namespace! This is no longer the preferred way to use matplotlib — only for interactive use with IPython it is still required (using the `-pylab` command line option to IPython).

The recommended way to import matplotlib's the function-based interface together with NumPy is by using:

```
import matplotlib.pyplot as plt
import numpy as np
```

The pyplot interface is a function-based interface using the same Matlab-like stateful conventions that we have seen in the examples in this section, but does not include the NumPy interface. Instead, NumPy is imported separately (and into its own namespace).

Finally, if all you want is the object-oriented API to matplotlib, you can import just the explicit modules from within matplotlib that contain the class definitions that you need (although it is customary to import pyplot instead, to get access to the whole collection).

Of course, there are many details that we have not discussed. Let me mention just a few:

- Many more options (to configure the axes and tick marks, to add legend or arrows)
- Additional plot types (density or “false-color” plots, vector plots, polar plots)
- Digital image processing (matplotlib can read and manipulate PNG images, and can also call into the Python Image Library (PIL) if it is installed)
- Matplotlib can be embedded in a GUI and can handle GUI events.

We will see one more example, involving matplotlib being called from a script to generate image files in the Workshop of chapter 4.

3.9 Further Reading

In addition to the books listed below, you may check the references in chapter 10 for additional material on linear regression.

- *The Elements of Graphing Data*. William S. Cleveland. 2nd ed., Hobart Press. 1994.

This is probably the definitive reference on graphical analysis (in particular compared to presentation graphics). Cleveland is the inventor of both

the LOESS and banking techniques discussed in this chapter. My own thinking has been influenced strongly by Cleveland's careful approach. There is also a companion volume, "Visualizing Data" by the same author.

- *Exploratory Data Analysis with MATLAB*. Wendy L. Martinez and Angel R. Martinez. Chapman & Hall/CRC. 2004.

This is an interesting book — it covers almost the same topics as the book you are reading, but in *opposite* order, starting with dimensionality reduction and clustering techniques, and ending with univariate distributions! Since this book demonstrates all techniques by way of Matlab, it does not develop the conceptual background in great depth. However, I found the chapter on smoothing to be quite useful.

```

from pylab import *

# h: location; h: bandwidth; xp, yp: data points
def loess( x, h, xp, yp ):
    w = exp( -0.5*((x-xp)/h)**2 )/sqrt(2*pi*h**2) )

    b = sum(w*xp)*sum(w*yp) - sum(w)*sum(w*xp*yp)
    b /= sum(w*xp)**2 - sum(w)*sum(w*xp**2)
    a = ( sum(w*yp) - b*sum(w*xp) )/sum(w)

    return a + b*x

d = loadtxt( "draftlottery" )

s1, s2 = [], []
for k in d[:,0]:
    s1.append( loess( k, 5, d[:,0], d[:,1] ) )
    s2.append( loess( k, 100, d[:,0], d[:,1] ) )

xlabel( "Day in Year" )
ylabel( "Draft Number" )

gca().set_aspect( 'equal' )

plot( d[:,0], d[:,1], 'o', color="white", markersize=7, linewidth=3 )
plot( d[:,0], array(s1), 'k-', d[:,0], array(s2), 'k--' )

q = 4
axis( [1-q, 366+q, 1-q, 366+q] )

savefig( "draftlottery.eps" )

```

Example 3.1: TBD

Chapter 4

Time as a Variable: Time Series Analysis

If we follow the variation of some quantity over time, we are dealing with a *time series*. Time series are incredibly common: examples range from stock market movements to the tiny icon that constantly displays the CPU utilization of your desktop computer for the last 10 seconds. What makes time series so common and so important is that they allow us to see not only a single quantity by itself, but at the same time give us the typical “context” for this quantity. Because we have not only a single value, but a bit of history as well, we can recognize any changes from the typical behavior particularly easily.

On the face of it, time series analysis is a bivariate problem (see chapter 3). Nevertheless, we are dedicating a separate chapter to them. Time series raise a different set of issues than many other bivariate problems, and a rather specialized set of methods has been developed to deal with them.

4.1 Examples

To get started, let’s look at a few different time series to get a sense for the scope of the task.

Figure 4.1 shows the concentration of Carbon Dioxide (CO_2) in the atmosphere, as measured by the observatory on Mauna Loa on Hawaii, recorded at monthly intervals over the years 1959 through 1990.

This data set shows two features we often find in a time series plot: trend and seasonality. There is clearly a long term, steady growth in the overall concentration of CO_2 : that is the *trend*. In addition, there is a very regular periodic pattern overlayed the trend: this is the *seasonality*. If we look closely, we see that the period covers exactly 12 months, but we will use the term seasonality for any regularly recurring feature, independent of the length of the period. We should also note that the trend, although smooth, does appear to be non-linear

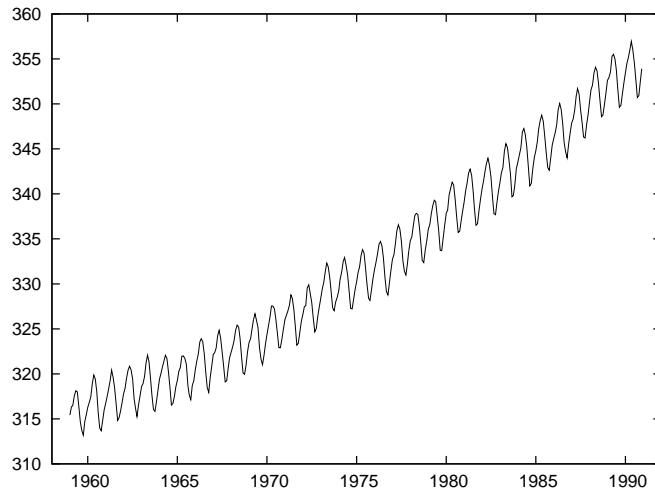


Figure 4.1: Trend and seasonality: the concentration of CO₂ (in parts per million) in the atmosphere, as measured by the observatory at Mauna Loa on Hawaii at monthly intervals.

(in fact, it the growth in CO₂ concentration seems to be accelerating).

Figure 4.2 displays the concentration of a certain gas in the exhaust of a gas furnace over time. In many ways, this example is the exact opposite from the previous example. Whereas the data in figure 4.1 showed a lot of regularity and a strong trend, the data in the current example shows no trend, but a lot of noise.

Figure 4.3 shows the dramatic drop in the cost of a typical long distance phone call in the US over the last century. The strongly non-linear trend is obviously the most outstanding feature of this data set. As with many growth- or decay-processes, we may suspect an exponential time development, and in fact in a semi-logarithmic plot (inset in figure 4.3) the data follows almost a straight line, confirming our expectation. Any analysis that fails to account explicitly for this behavior of the original data is likely to lead us astray. We should therefore work with the logarithms of the cost, rather than with the absolute cost.

(There are some additional questions that we should ask, given a long-running data set like this. What exactly is a “typical” long distance call and has that definition changed over the observation period? Are the costs adjusted for inflation or not? The data itself also begs deeper scrutiny: the uncharacteristically low prices for a couple of years in the late '70s make me suspicious. Are they the result of a clerical error (that is: a typo) or are they real? Did the break-up of the AT&T system have anything to do with them? We will not follow up on these questions here, because I am only interested in this example

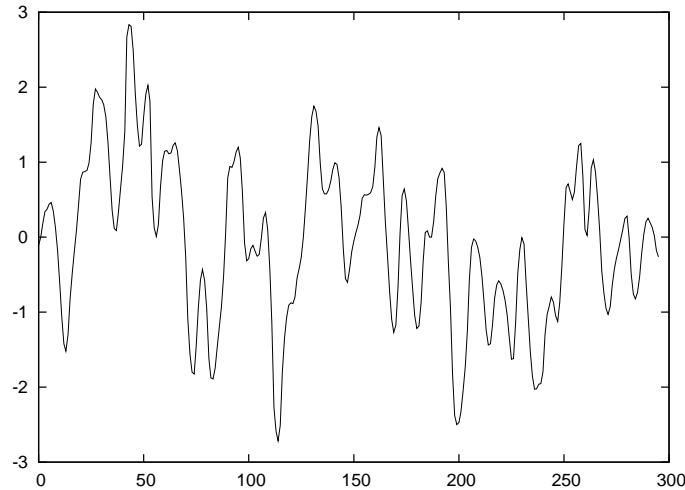


Figure 4.2: No trend, but relatively smooth variation over time: concentration of a certain gas in a furnace exhaust (in arbitrary units).

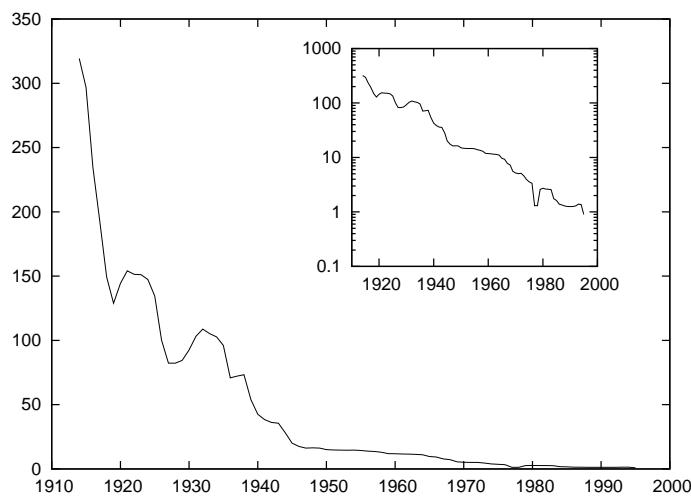


Figure 4.3: Non-linear trend: the cost of a typical long distance phone call in the US.

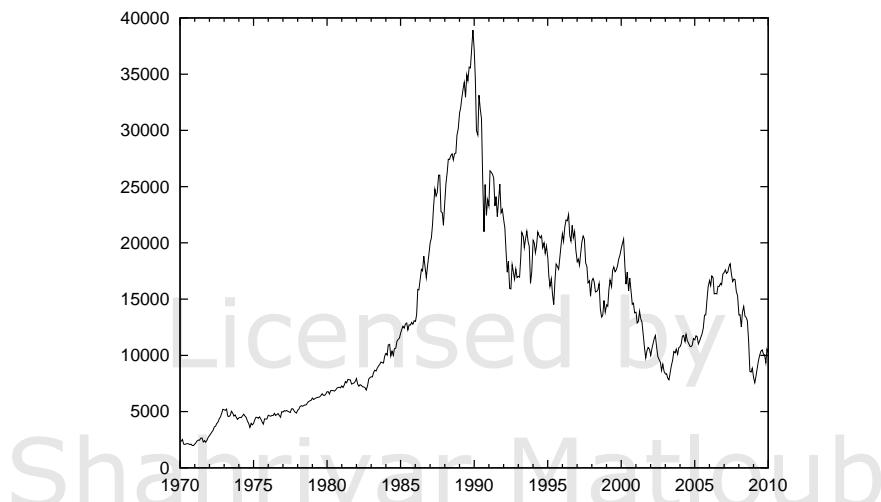


Figure 4.4: Change in behavior: the Nikkei Stock Index over the last 40 years.

as an illustration of an exponential trend, but any serious analysis of this data set would have to follow up on these questions.)

Figure 4.4 finally shows the development of the Japanese stock market as represented by the Nikkei Stock Index over the last forty years, as an example of a time series exhibiting a marked change in behavior. Clearly, whatever was true before New Year 1990 was no longer true afterwards. (In fact, if you look closely, you will make out a second change in behavior, more subtle than the bursting of the big Japanese bubble: namely its beginning, sometime around 1985–1986.)

This data set should serve as a cautionary example: all time series analysis is based on the assumption that the processes generating the data are stationary in time — if the rules of the game change, then time series analysis is the wrong tool for the task, and instead we have to investigate what caused the break in behavior. More benign examples than the bursting of the big Japanese bubble can be found: a change in sales or advertising strategy may significantly alter the sales patterns at some company. In such cases, it is more important to inquire about any further plans that the sales department might have, rather than to continue working with data that is no longer representative!

After these examples which have been chosen for their “text book” properties, let’s look at a “real-world” data set. Figure 4.5 shows the number of daily calls placed to a call center for a time period slightly longer than two years. In comparison to the previous examples, this data set has a lot more structure, which makes it hard to determine even basic properties: we can see some high-frequency variation, but it is not clear whether this is noise or has some degree of regularity to it. It is also not clear whether there is any sort of regularity on a

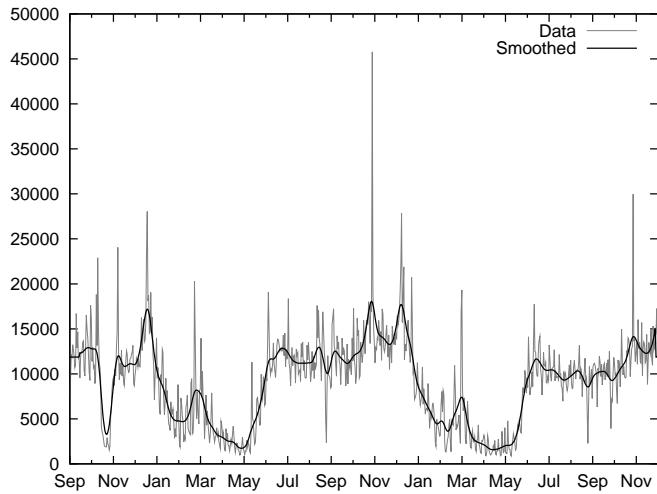


Figure 4.5: A real-world data set. Short and long term seasonality, noise, possibly changes in behavior: number of calls placed to a callcenter each day Also shown is the result of applying a 31-point Gaussian smoothing filter.

longer time scale. The amount of variation makes it hard to recognize any further structure. For instance, it is not clear whether there is a longer term trend in the data, or not. We will come back to this example later in this chapter.

4.2 The Task

After this tour of possible time series scenarios, we can identify the main components of every time series:

- trend
- seasonality
- noise
- other (!)

The trend may be linear or non-linear, and we may want to investigate its magnitude. The seasonality pattern may be either additive or multiplicative — in the first case, the seasonal change has the same *absolute* size, no matter what the magnitude of the current “baseline” of the series is; whereas in the latter case the seasonal change has the same *relative* size compared to the current magnitude of the series. Noise, that is some form of random variation is almost always part of a time series. Finding ways to reduce the noise in the

data is usually a significant part of the analysis process. Finally, “other” stands for anything else that we may observe in a time series, such as particular significant changes in the overall behavior, special outliers, missing data — anything remarkable at all.

Given this list of components, we can summarize what it means to “analyze” a time series. We can distinguish three basic tasks:

- description
- prediction
- control

Description attempts to identify components of a time series (such as trend and seasonality or abrupt changes in behavior). Prediction tries to forecast future values. Control in this context means the monitoring of a process over time, with the purpose of keeping it within a pre-defined band of values — a typical task in many manufacturing or engineering environments. We can distinguish the three tasks by the time frame that they address: description looks into the past, prediction looks to the future, and control concentrates on the present.

4.2.1 Requirements and the Real World

Most standard methods of time series analysis make a number of assumption on the underlying data.

- Data points have been taken at equally spaced time steps, with no missing data points.
- The time series is sufficiently long: 50 points are often seen as an absolute minimum.
- The series must be *stationary*, that is have no trend, no seasonality, and the character (amplitude and frequency) of the “noise” does not change with time.

Unfortunately, most of these assumptions will be more or less violated by any real-world data set that you are likely to encounter, and so you may have to perform a certain amount of data cleaning before you can apply the methods described in this chapter.

If the data has been sampled at irregular time steps or if some of the data points are missing, you can try to interpolate the data and resample it at equally spaced intervals. Time series obtained from electrical systems or scientific experiments can be almost arbitrarily long, but most series arising in a business context will be quite short, containing possibly no more than two dozen data points. The exponential smoothing methods which I will introduce in the next section are relatively robust even for relatively short series, but somewhere

there is a limit. Three or four data points don't constitute a series! Finally, most interesting series will not be stationary in the sense of the definition given above, and we may have to identify and remove trend and seasonal components explicitly (we'll talk later about how to do that.) Drastic changes in the nature of the series obviously violate the stationarity condition as well — in such cases we must not continue blindly, but deal with the break in the data, for example by treating the data set as two different series: one before and one after the event.

4.3 Smoothing

An important aspect of most time series is the presence of *noise*, that is random (or apparently random) changes in the quantity of interest. Noise occurs in many real-world data sets, but in many other situations, we can reduce the “noise”, either by improving the apparatus used to measure the data, or by collecting a larger sample and averaging over it. But the particular structure of time series makes this impossible: the sales figures for the last 30 days are fixed and they are all we have. So, ways to remove noise or at least reduce its influence are of particular importance in time series analysis. In other words, we are looking for ways to *smooth* the signal.

4.3.1 Running Averages

The simplest smoothing algorithm that we can come up with is the *running*, *moving* or *floating average*. The idea is very simple: for any odd number of consecutive points, replace the center-most value with the average of the other points (here, the $\{x_i\}$ are the data points and the smoothed value at position i is s_i):

$$s_i = \frac{1}{2k+1} \sum_{j=-k}^k x_{i+j}$$

This naive approach has a serious problem, as you can see in figure 4.6. The figure shows the original signal together with the 11-point moving average. Unfortunately, the signal has some sudden jumps and occasional large “spikes”, and we can see how the smoothed curve is affected by these events: whenever a spike enters the smoothing window, the moving average is abruptly distorted by the single unusually large value, until the outlier leaves the smoothing window again, at which point the floating average equally abruptly drops again.

We can avoid this problem by using a *weighted moving average*, which places less weight on the points at the edge of the smoothing window. Using such a weighted average, any new point that enters the smoothing window is only

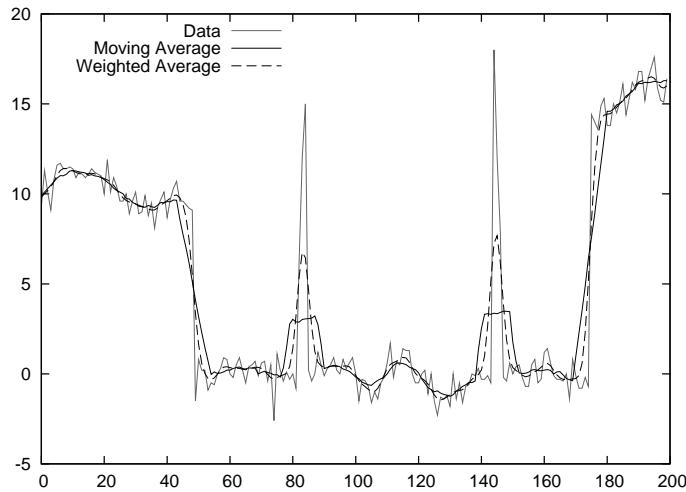


Figure 4.6: The difference between a plain and a Gaussian weighted moving average: sudden jumps in the data lead to corresponding jumps in the smoothed signal. The weighted average follows the original data more faithfully.

gradually added to the average and equally gradually removed again.

$$s_i = \sum_{j=-k}^k w_j x_{i+j} \quad \text{where} \quad \sum_{j=-k}^k w_j = 1$$

Here w_j are the weighting factors. For example, for a 3-point moving average we might use $(1/4, 1/2, 1/4)$. The particular choice of weight factors is not very important, as long as they are peaked at the center, drop towards the edges, and add up to one. I like to use the Gaussian function

$$f(x, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\left(\frac{x}{\sigma}\right)^2\right)$$

to build smoothing weight factors. The parameter σ in the Gaussian controls the width of the curve, and the function is essentially zero for values of x larger than about 3.5σ . So, $f(x, 1)$ can be used to build a 9-point kernel, by evaluating $f(x, 1)$ at the points $[-4, -3, -2, -1, 0, 1, 2, 3, 4]$. Setting $\sigma = 2$, we can form a 15-point kernel, by evaluating the Gaussian for all integer arguments between -7 and $+7$. And so on.

4.3.2 Exponential Smoothing

All moving average-schemes have a number of problems:

- They are painful to evaluate. For each point, the calculation has to be performed from scratch. It is not possible to evaluate weighted moving averages by updating a previous result.
- Moving averages can never be extended to the true edge of the available data set, because of the finite width of the averaging window. This is particularly problematic, since often it is precisely the behavior at the leading edge of a data set that we are most interested in.
- Similarly, moving averages are not defined *outside* the range of the existing data set. As a consequence, they are of no use in forecasting.

Fortunately, there exists a very simple calculational scheme which avoids all of these problems. It is called *exponential smoothing* or *Holt-Winters Method*. There are various forms of exponential smoothing: single exponential smoothing for series that have neither trend nor seasonality, double exponential smoothing for series exhibiting a trend but no seasonality, and finally triple exponential smoothing for series with both trend and seasonality. The name Holt-Winters method is sometimes reserved for triple exponential smoothing only.

All exponential smoothing methods work by updating the result from the previous time step using the new information contained in the data of the current time step. They do so by “mixing” the new information with the old one; the relative weight of old and new information is controlled through an adjustable mixing parameter. The methods differ by the number of quantities they track and the corresponding number of mixing parameters.

The recurrence relation for single exponential smoothing is particularly simple:

$$s_i = \alpha x_i + (1 - \alpha)s_{i-1} \quad \text{with } 0 \leq \alpha \leq 1$$

Here, s_i is the smoothed value at time step i and x_i is the actual (un-smoothed) data at that time step. You can see how s_i is a mixture between the raw data and the previous smoothed value s_{i-1} . The mixing parameter α can be anywhere between 0 and 1 and controls the balance between the new and the old information: as α approaches 1, we retain only the current data point (that is, the series is not smoothed at all); as α goes to 0, we retain only the smoothed past (that is, the curve is totally flat).

Why is this method called exponential smoothing? To see this, simply expand the recurrence relation:

$$\begin{aligned}
 s_i &= \alpha x_i + (1 - \alpha)s_{i-1} \\
 &= \alpha x_i + (1 - \alpha)[\alpha x_{i-1} + (1 - \alpha)s_{i-2}] \\
 &= \alpha x_i + (1 - \alpha)[\alpha x_{i-1} + (1 - \alpha)[\alpha x_{i-2} + (1 - \alpha)s_{i-3}]] \\
 &= \dots \\
 &= \alpha [x_i + (1 - \alpha)x_{i-1} + (1 - \alpha)^2 x_{i-2}] + (1 - \alpha)^3 s_{i-3} \\
 &= \dots \\
 &= \alpha \sum_{j=0}^i (1 - \alpha)^j x_{i-j}
 \end{aligned}$$

What this shows is that in exponential smoothing *all* previous observations contribute to the smoothed value, but the contribution is suppressed by increasing powers of the parameter α . The fact that observations further in the past are suppressed multiplicatively is characteristic for exponential behavior. In a way, exponential smoothing is like a floating average with infinite memory, but with exponentially falling weights.¹

The results of the simple exponential smoothing procedure can be extended beyond the end of the data set, and thus be turned into a forecast. The forecast is particularly simple:

$$x_{i+h} = s_i$$

where s_i is the last calculated value. In other words, single exponential smoothing yields a forecast which is absolutely flat for all times.

Single exponential smoothing as described above works well for time series without an overall trend. However, in the presence of an overall trend, the smoothed values tend to lag behind the raw data, unless α is chosen to be close to 1; however, in this case the resulting curve is not sufficiently smoothed.

Double exponential smoothing corrects for this by retaining explicit information about the trend as well. In other words, we maintain and update the state of two quantities: the smoothed signal, *and* the smoothed trend. There are two equations and two mixing parameters:

$$\begin{aligned}
 s_i &= \alpha x_i + (1 - \alpha)(s_{i-1} + t_{i-1}) \\
 t_i &= \beta(s_i - s_{i-1}) + (1 - \beta)t_{i-1}
 \end{aligned}$$

Let's look at the second equation first. It describes the smoothed trend. The current un-smoothed "value" of the trend is calculated as the difference between the current and the previous smoothed signal: in other words, the current trend tells us how much the smoothed signal changed in the last step. To

¹Also note the fact that the sum of the weights: $\sum_j \alpha(1 - \alpha)^j$ equals 1 as required, by virtue of the geometric series: $\sum_i q^i = 1/(1 - q)$ for $q < 1$. (See appendix B.)

form the smoothed trend, we perform a simple exponential smoothing process on the trend, using the mixing parameter β . To obtain the smoothed signal, we perform a similar mixing as before, only that now we don't just consider the previous smoothed signal, but take into account the trend as well: the last term in the first equation is the best guess for the current smoothed signal, if we had followed the previous trend for a single time step.

To turn this result into a forecast, we take the last smoothed value and for each additional time step keep adding the last smoothed trend to it:

$$x_{i+h} = s_i + h t_i$$

Finally, for triple exponential smoothing, we add yet a third quantity, which describes the seasonality. We have to distinguish between additive and multiplicative seasonality. For the additive case, the equations are:

$$\begin{aligned}s_i &= \alpha(x_i - p_{i-k}) + (1 - \alpha)(s_{i-1} + t_{i-1}) \\t_i &= \beta(s_i - s_{i-1}) + (1 - \beta)t_{i-1} \\p_i &= \gamma(x_i - s_i) + (1 - \gamma)p_{i-k} \\x_{i+h} &= s_i + h t_i + p_{i-k+h}\end{aligned}$$

and for the multiplicative case they are:

$$\begin{aligned}s_i &= \alpha \frac{x_i}{p_{i-k}} + (1 - \alpha)(s_{i-1} + t_{i-1}) \\t_i &= \beta(s_i - s_{i-1}) + (1 - \beta)t_{i-1} \\p_i &= \gamma \frac{x_i}{s_i} + (1 - \gamma)p_{i-k} \\x_{i+h} &= (s_i + h t_i)p_{i-k+h}\end{aligned}$$

where p_i is the "periodic" component and k is the length of the period. I have also included the expressions for forecasts.

All exponential smoothing methods are based on recurrence relations. To actually use them, we still need to fix the start-up values. Luckily, the specific choice for these values is not very critical: the exponential damping implies that all exponential smoothing methods have a short "memory", so that after only a few steps, any influence of the initial values is greatly diminished. Some reasonable choices for start-up values are:

$$s_0 = x_0 \quad \text{or} \quad s_0 = \frac{1}{n} \sum_i^n x_i \quad \text{with } 1 < n < 5 \dots 10$$

and

$$t_0 = 0 \quad \text{or} \quad t_0 = x_1 - x_0$$

For triple exponential smoothing, we need to provide one full season of values for start-up, but we can simply fill them with 1s (for the multiplicative

model) or 0s (for the additive model). Only if the series is short do we need to worry very much about finding good starting values.

The last question concerns how to choose the mixing parameters α , β , and γ . My advice is: trial and error. Try a few values between 0.2 and 0.4 (very roughly) and see what results you get. Alternatively, you can define a measure for the error (between the actual data and the output of the smoothing algorithm) and then use a numerical optimization routine to minimize this error with respect to the parameters. In my experience, this is usually more trouble than it's worth, for at least the following two reasons. The numerical optimization is an iterative process, which is not guaranteed to converge and you may end up spending way too much time to coax the algorithm to convergence. Furthermore, any such numerical optimization is slave to the expression which you have chosen for the "error" to be minimized. The problem is that the parameter values minimizing that error may not have some other property you would want to see in your solution (for example in regards to the balance between the accuracy of the approximation and the smoothness of the resulting curve), so that in the end the manual approach often comes out ahead. On the other hand, if you have many series to forecast, it may make sense to expend the effort and build a system which can determine the optimal parameter values automatically. It's not going to be easy to really make this work, however.

Finally, I want to show an example for the kind of results we can expect from exponential smoothing. Figure 4.7 is a classical dataset, showing the monthly number of international airline passengers (in thousands of passengers)² I show the actual data, together with a triple exponential approximation. The years 1949 through 1957 have been used to "train" the algorithm and the years 1958 through 1960 are forecasted. Note how well the forecasts agree with the actual data, in particular given the strong seasonal pattern, for a rather long forecasting time frame (fully 3 years!). Not bad for a method that is as simple as this.

4.4 Don't Overlook the Obvious!

On a recent consulting assignment, I was discussing monthly sales numbers with the client, when he made the following comment: "Oh, yes, sales for February are always somewhat lower — that's an after-effect of the Christmas peak." Sales are always lower in *February*? How very interesting...

Sure enough, if you plotted the monthly sales numbers for the last few years, there was a rather visible dip from the overall trend — every February. But in contrast, there wasn't much of a Christmas spike! (The client's business was not particularly seasonal.) So, why should there be a corresponding dip, two months later?

By now, I am sure you know the answer already: February is *shorter* than any of the other months. And it's not a small effect either: February with 28

²This data is available in the "airpass.dat" data set from R. J. Hyndman's Time Series Data Library <http://www.robjhyndman.com/TSDL>.

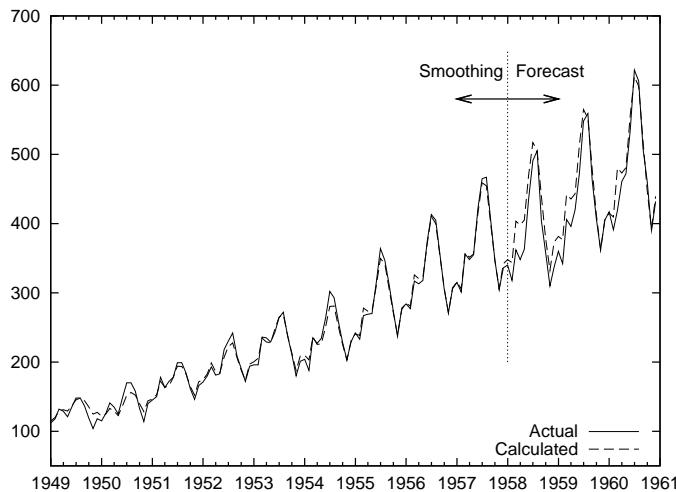


Figure 4.7: Triple exponential smoothing in action: comparison between the raw data (solid line) and the smoothed curve (dashed). For the years after 1957, the dashed curve shows the *forecast* calculated with only the data available in 1957.

days is about 3 days shorter than the other months with 30–31 days. That's about 10% — about the size of the dip in the client's sales numbers.

When I normalized monthly sales numbers by the number of days in the month, the February dip all but disappeared, and the *adjusted* February numbers were perfectly in line with the rest of the months. (The average number of days per month is $365/12 = 30.4$.)

Whenever you are tracking aggregated numbers in a time series (such as weekly, monthly, quarterly results), make sure that you have adjusted for possible variation in the aggregation time frame. Besides the numbers of days in the month, another likely candidate for hiccups is the number of *business days* in a month month (if five weekends fall into a month, you can expect a 20% drop for most business metrics). But the problem is of course much more general, and can occur whenever you are reporting aggregate *numbers*, as opposed to *rates*. (If the client had been reporting average sales per day for each month, the problem would never have occurred.)

This specific problem (non-adjusted variations in aggregation periods) is a particular concern for all business reports and dashboards. Keep an eye out for it!

4.5 Correlation Function

The *auto-correlation function* is the primary diagnostic tool for time series analysis. Whereas the smoothing methods that we have talked about so far deal with the raw data in a very direct way, the correlation function provides us with a very *different* view on the same data. I will first explain how the auto-correlation function is calculated and then discuss what it means and how it can be used.

The basic algorithm works as follows: start with two copies of the data set and subtract the overall average from all values. Align the two sets and multiply the values at corresponding time steps with each other. Sum up the results for all time steps. The result is the (unnormalized) correlation coefficient at *lag 0*. Now shift the two copies against each other by a single time step. Again multiply and sum: the result is the correlation coefficient at lag 1. Proceed in this way for the entire length of the time series. The set of all correlation coefficients for all lags is the auto-correlation function. Finally, divide all coefficients by the coefficient for lag 0 to normalize the correlation function, so that the coefficient for lag 0 is now equal to 1.

All this can be written compactly in a single formula for $c(k)$, that is the correlation function at lag k :

$$c(k) = \frac{\sum_{i=1}^{N-k} (x_i - \mu)(x_{i+k} - \mu)}{\sum_{i=1}^N (x_i - \mu)^2} \quad \text{with } \mu = \frac{1}{N} \sum_{i=1}^N x_i$$

Here, N is the number of points in the data set. The formula follows the mathematicians' convention to start indexing into sequences at 1, rather than the programmer's convention to start indexing at 0. Notice that we have subtracted the overall average μ from all values, and that the denominator is simply the expression of the numerator for lag $k = 0$. Figure 4.8 demonstrates the process pictorially.

The meaning of the correlation function should be clear: initially, the two signals are perfectly aligned, and the correlation is 1. Then, as we shift the signals against each other, they slowly get out of phase with each other, and the correlation drops. How quickly it drops tells us how much "memory" there is in the data: if the correlation drops very quickly, then that is a sign that after a few steps the signal has lost all memory of its recent past. If, on the other hand, the correlation drops slowly, we know that we are dealing with a process which is relatively steady over longer periods of time. It is also possible that the correlation function first drops and then raises up again to form a second (and possibly a third, or fourth, ...) peak. This tells us that if we shift the two signals far enough, they do align again — in other words, that there is periodicity (that is, seasonality) in the data set. The position of the secondary peak gives us the number of time steps per season.

```

Lag 0:
0 | 1 | 2 | 3 | 4 | 5 | 6
*   *   *   *   *   *
0 | 1 | 2 | 3 | 4 | 5 | 6

Lag 1:
0 | 1 | 2 | 3 | 4 | 5 | 6 | .
*   *   *   *   *   *
0 | 1 | 2 | 3 | 4 | 5 | 6

Lag 2:
0 | 1 | 2 | 3 | 4 | 5 | 6 | . | .
*   *   *   *   *
0 | 1 | 2 | 3 | 4 | 5 | 6

...

```

Figure 4.8: The algorithm to compute the correlation function.

4.5.1 Examples

Let's look at a couple of examples. Figure 4.9 shows the correlation function of the gas furnace data in figure 4.2. This is a rather typical correlation function for a time series that has *short time* correlations: the correlation falls quickly, but not immediately, to zero. There is no periodicity: after the initial drop, the correlation function does not exhibit any further significant peaks.

Figure 4.10 is the correlation function for the call center data from figure 4.5. This data set shows a very different behavior. First of all, the time series has a much longer "memory": it takes the correlation function almost 100 days to fall to zero, indicating that the frequency of calls to the call center changes more or less once per quarter, but not more frequently. The second notable feature is the pronounced secondary peak, at a lag of 365 days. In other words, the call center data is highly seasonal, repeating itself on a yearly basis. The third feature is the small, but regular "saw tooth" structure. If we look closely, we will find that the first peak of the saw tooth is at a lag of 7 days and that all repeating ones occur at multiples of 7. This is the signature of the high-frequency component that we could see in figure 4.5: the traffic to the call center displays secondary seasonal component with a 7-day periodicity: in other words, traffic is weekday dependent (not too surprisingly).

4.5.2 Implementation Issues

So far, I have mostly talked about the correlation function from a conceptual point of view. If we want to proceed to an actual implementation, there are some fine points we need to worry about.

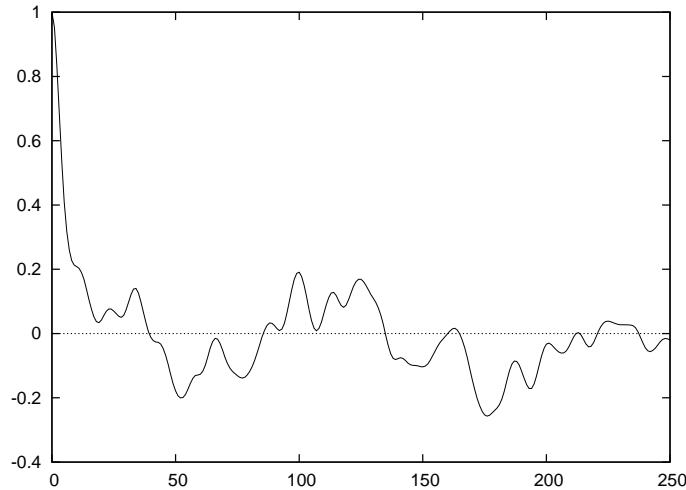


Figure 4.9: The correlation function for the exhaust gas data shown in figure 4.2. The data has short time correlation: the correlation function falls quickly, but not immediately to zero. There are no secondary peaks.

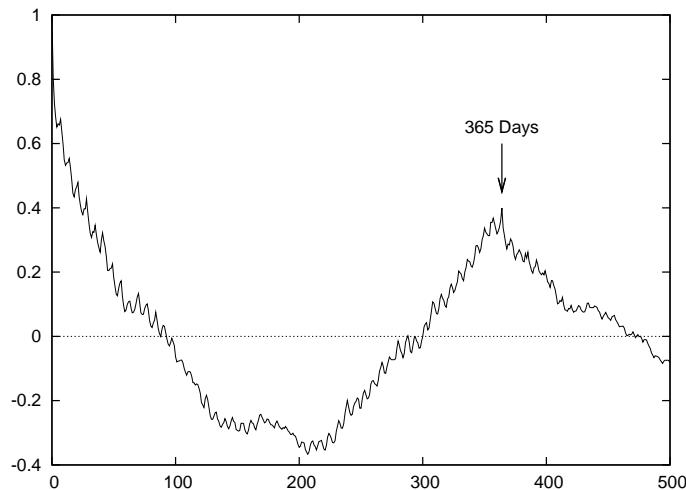


Figure 4.10: The correlation function for the callcenter data shown in figure 4.5. There is a secondary peak after exactly 365 days, and also a smaller weekly structure.

The auto-correlation function is intended for time series that do not exhibit a trend and have zero mean. Therefore, if the series we want to analyze does contain a trend, we need to remove it first. There are two basic ways we can do this: we can subtract the trend, or we can difference the series.

Subtracting the trend is straightforward — the only problem is that we need to determine the trend first! Sometimes we may have a “model” for the expected behavior and we can use this to construct an explicit expression for the trend. The airline passenger data that we considered in the previous section, for example, describes a growth process, and so we should suspect an exponential trend ($a \exp(x/b)$). We can now try guessing values for the two parameters and then subtract the exponential term from the data. For other data sets, we might try a linear or power law trend, depending on the data set and our understanding of the process generating the data. Alternatively, we might first apply a smoothing algorithm to the data and then subtract the result of the smoothing process from the raw data. The result will be the trend-free “noise” component of the time series.

A different approach consists of *differencing* the series: instead of dealing with the raw data, we instead work with the *changes* in the data from one time step to the next. Technically, this means replacing the original series x_i with one consisting of the differences of consecutive elements: $x_{i+1} - x_i$. If necessary, this process can be repeated, but in most cases single differencing is sufficient to remove the trend entirely.

Making sure that the time series has zero mean is simpler: simply calculate the mean of the (de-trended!) series and subtract it before calculating the correlation function. In the formula for the correlation function that I gave earlier this is done explicitly.

Another technical wrinkle concerns the way we implement the sum in the formula for the numerator. As written, this sum is slightly messy, with the upper limit of the sum depending on the term we are calculating. We can simplify it by *padding* one of the data sets with N zeroes on the right and letting the sum run from $i = 1$ to $i = N$ for all lags. In fact, many calculational packages that you may encounter will expect the data to have been prepared this way (see the Workshop section in this chapter).

The last issue you should be aware of is that there are two different normalization conventions for the auto-correlation function that are both widely used. In the first variant, numerator and denominator are not normalized separately — this is the scheme used in the formula above. In the second variant, both numerator and denominator are normalized by the number of non-zero terms in each sum. With this convention, the formula becomes instead:

$$c(k) = \frac{\frac{1}{N-k} \sum_{i=1}^{N-k} (x_i - \mu)(x_{i+k} - \mu)}{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2} \quad \text{with } \mu = \frac{1}{N} \sum_{i=1}^N x_i$$

Both conventions are fine, but if you want to compare results from different sources, or different software packages, you will have to make sure that you know which convention either of them is following!

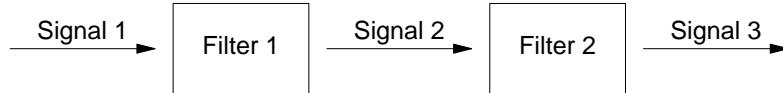


Figure 4.11: A filter chain: each filter applied to a signal yields another signal which itself can be filtered.

4.6 Filters and Convolutions

So far, we always talked about time series in a very direct fashion, but there is also a way of describing them and the operations performed on them on a much higher level of abstraction, by borrowing some concepts and terminology from electrical engineering, specifically from the field of *digital signal processing* (DSP).

In the lingo of DSP, we deal with *signals* (time series) and *filters* (operations). Applying a filter to a signal produces a new (filtered) signal. Since filters can be applied to any signal, we can apply another filter to the output of the first, and in this way end up chaining filters together (see figure 4.11). Signals can also be combined and subtracted from each other.

As it turns out, many of the operations we have seen so far (smoothing, differencing) can be expressed as filters. We can therefore use the convenient, high-level language of DSP when referring to the processes of time series analysis. To make this concrete, we need to understand how a filter is represented, and what it actually means to “apply” it to a signal.

Each digital filter is represented by a set of coefficients or weights. To apply the filter, we multiply the coefficients with a subset of the signal. The sum of the products is the value of the resulting (filtered) signal:

$$y_t = \sum_{i=-k}^k w_i x_{t+i}$$

This should look familiar! We used a very similar expression when talking about moving averages earlier in this chapter. A moving average is nothing but a time series run through an n -point filter, in which every coefficient is equal to $1/n$. A weighted moving average filter similarly consists of the weights used in the expression for the average.

But the filter concept is not limited to smoothing operations. The differencing step we talked about in the previous section can be thought of as the application of the filter $[1, -1]$. We can even shift an entire time series forward in time, by using the filter $[0, 1]$.

The last piece of terminology that we will need concerns the peculiar sum of a product that we have encountered several times by now. It’s called a *convolution*. A convolution is a way to combine two sequences, resulting in a third sequence, which you can think of as the “overlap” between the original sequences. The convolution operation is usually defined as follows:³

$$y_t = \sum_{i=-\infty}^{\infty} w_i x_{t-i}$$

Of course, if one or both of the sequences have only a finite number of elements, the sum also contains only a finite number of terms and therefore poses no difficulties. You

³Symbolically, the convolution operation is often expressed through an asterisk: $y = w * x$, where y , w , and x are sequences.

should be able to convince yourself easily that every application of a filter to a time series that we have done was in fact a convolution of the signal with the filter. This is true in general: applying a filter to a signal means forming the convolution of the two. You will find that many numerical software packages provide a convolution operation as a built-in function, making filter operations particularly convenient to use.

I have to warn you, however, that the entire machinery of digital signal processing is geared towards signals of infinite (or almost infinite) length, and if you think of typical electrical signals (such as the output from a microphone or a radio receiver) this makes good sense. For the rather short time series that we are likely to deal with, we need to pay close attention to a variety of *edge effects*. For example, if we apply a smoothing or differencing filter, the resulting series will be shorter by half the filter length than the original series. If we now want to subtract the smoothed from the original signal, the operation will fail because the two signals are not of equal length. We therefore have to either pad the smoothed signal or truncate the original one. The constant need to worry about padding and proper alignment detracts significantly from the conceptual beauty of the signal theoretic approach when used with time series of relatively short duration.

4.7 Workshop: `scipy.signal`

The `scipy.signal` package provides functions and operations for digital signal processing, which we can use to good effect to perform calculations for time series analysis. The `scipy.signal` package makes use of the signal processing terminology we just introduced in the previous section.

Listing 4.1 shows the complete commands used to create graphs like figures 4.5 and 4.10, including the commands to write the results out to file. The code is heavily commented and should be easy to understand. Here, I just want to draw attention to a few particular features.

The package provides the Gaussian, as well as many other filters. Unfortunately, the filters are not normalized, but this is easy enough to accomplish.

More attention needs to be paid to the appropriate padding and truncating. For example, to form the smoothed version of the data, I pad the data on both sides by half the filter length, to ensure that the smoothed data has the same length as the original set. The `mode` argument to the `convolve()` and `correlate` functions determines which pieces of the resulting vector to retain. Several modes are possible. Using `mode="same"`, the returned vectors has as many elements as the largest input vector (in our case: as the padded data vector), but the elements closest to the ends would be corrupted by the padded values. In the listing, I instead use `mode="valid"`, which keeps only those elements which have full overlap between the data and the filter — in effect, removing the elements added in the padding step again.

Notice how the signal processing machinery leads in this application to very compact code. Once you strip the comments and plotting commands out, there are only about 10 lines of code that perform actual operations and calculations. On the other hand, we had to pad all data carefully and to make sure to retain only those pieces of the result least contaminated by it!

4.8 Summary

Time series are both common and important. But because they arise in so many, and so many *different* contexts, little can be said about them without knowing the details. Studying and classifying the nature of each time series is therefore very important — blindly applying “cook book” methods, no matter how sophisticated, is never a good strategy.

The two most important tools for studying time series are the time plot and the plot of the auto-correlation function. Both together can reveal most of what can be found out about any particular time series. The signal theoretic methodology that we discussed briefly can provide a convenient framework to handle the calculations that are required.

Smoothing and forecasting are the two most often desired results of time series analysis and here I strongly recommend simple, yet robust methods, such as weighted averages (for smoothing) and exponential smoothing (for forecasting).

4.9 Further Reading

- *The Analysis of Time Series*. Chris Chatfield. 6th ed., Chapman & Hall. 2003.

My preferred text on time series analysis. It combines a thoroughly practical approach with mathematical depth and a healthy preference for the simple over the obscure. Highly recommended.

```

from scipy import *
from scipy.signal import *
from matplotlib.pyplot import *

filename = 'callcenter'

# Read data from a text file, retaining only the third column.
# (Column indexes start at 0.)
# The default delimiter is any whitespace.
data = loadtxt( filename, comments='#', delimiter=None, usecols=(2,) )

# The number of points in the time series. We will need it later.
n = data.shape[0]

# Finding a smoothed version of the time series:
# 1) Construct a 31-point Gaussian filter with standard deviation = 4
filt = gaussian( 31, 4 )
# 2) Normalize the filter through dividing by the sum of its elements
filt /= sum( filt )
# 3) Pad data on both sides with half the filter length of the last value
#     (The function ones(k) returns a vector of length k, with all elements 1.)
padded = concatenate( (data[0]*ones(31//2), data, data[n-1]*ones(31//2)) )
# 4) Convolve the data with the filter. See text for the meaning of "mode".
smooth = convolve( padded, filt, mode='valid' )

# Plot the raw together with the smoothed data:
# 1) Create a figure, sized to 7x5 inches
figure( 1, figsize=( 7, 5 ) )
# 2) Plot the raw data in red
plot( data, 'r' )
# 3) Plot the smoothed data in blue
plot( smooth, 'b' )
# 4) Save the figure to file
savefig( filename + "_smooth.png" )
# 5) Clear the figure
clf()

# Calculate the autocorrelation function:
# 1) Subtract the mean
tmp = data - mean(data)
# 2) Pad one copy of data on the right with zeros, then form correlation fct
#     The function zeros_like(v) creates a vector with the same dimensions
#     as the input vector v, but with all elements zero.
corr = correlate( tmp, concatenate( (tmp, zeros_like(tmp)) ), mode='valid' )
# 3) Retain only some of the elements
corr = corr[:500]
# 4) Normalize by dividing by the first element
corr /= corr[0]

# Plot the correlation function:
figure( 2, figsize=( 7, 5 ) )
plot( corr )

```


Chapter 5

More Than Two Variables: Graphical Multivariate Analysis

The moment we are dealing with more than two variables simultaneously, things become much more complicated — in particular graphical methods quickly become impractical. In this chapter I'll introduce a number of graphical methods that can be applied to multivariate problems. All of them work best if the number of variables is not *too* large (less than 20–25).

The border-line case of *three* variables can be handled through *false-color plots*, which we will discuss first.

If the number of variables is greater (but not much greater) than three, we can construct multi-plots out of a collection of individual, bivariate plots, by scanning through the various parameters in a systematic way. This gives raise to scatter-plot matrices and co-plots.

Depicting the way an overall entity is composed out of its constituent parts can be a rather nasty problem, in particular if the composition changes over time. Because this task is so common, I'll treat it separately in its own section.

Multidimensional visualization continues to be research topic, and in the last sections of this chapter we look at some more recent ideas in this space.

One recurring theme in this chapter will be the need for adequate tools: most multi-dimensional visualization techniques are either not practical with paper-and-pencil, or are outright impossible to do without a computer (in particular when it comes to animated techniques). Moreover, as the number of variables goes up, so does the need to look at a data set from different angles, which leads to the idea of using interactive graphics for exploration. In the last section, we look at some ideas in this area.

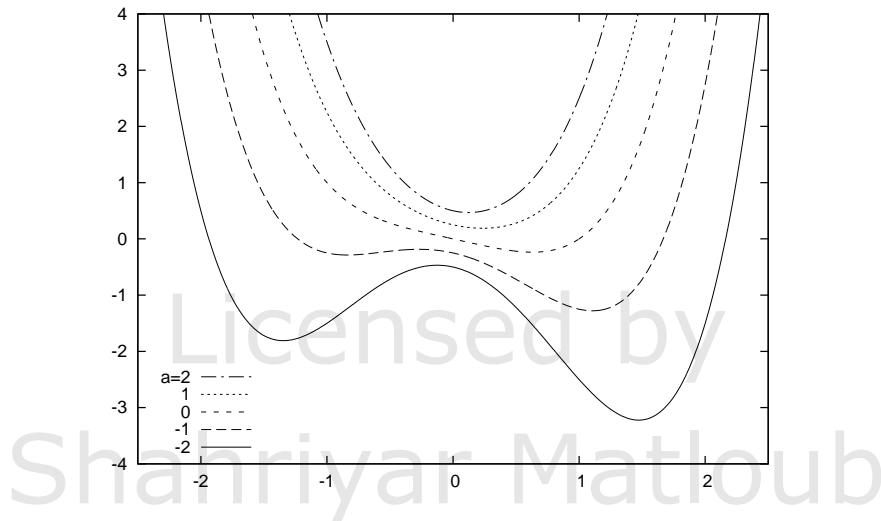


Figure 5.1: A simple, yet effective way to show three variables: treat one as parameter and draw a separate curve for several parameter values.

5.1 False-Color Plots

There are different ways to display information in three variables (typically, two independent and one dependent variable). Keep in mind that simple is sometimes best! Figure 5.1 for example shows the function $f(x,a) = x^4/2 + ax^2 - x/2 + a/4$ for various values of the parameter a in a simple, two-dimensional xy-plot. The shape of the function and the way it changes with a become perfectly clear from this graph, and is very difficult to display otherwise.

Another way to represent such tri-variate data is in the form of a *surface plot*, such as the one shown in figure 5.2. As a rule, surface plots are visually stunning, but are of very limited usefulness otherwise. Unless the data set is very smooth, and has a view point that allows us to look *down* onto the surface, they simply don't work! For example, it is pretty much impossible to get a good idea of the behavior of the function I showed in figure 5.1 from a surface plot. (Try it!). Surface plots give an idea for the overall structure of the data, but it is notoriously difficult to read off quantitative information from them.

In my opinion, surface plots have only two uses:

- to get an intuitive impression of the “lay of the land” for a complicated data set, and
- to dazzle the boss! (Not that this is not important at times.)

Another approach is to project the function into the base plane below the surface in figure 5.2. There are two ways in which we can represent values:

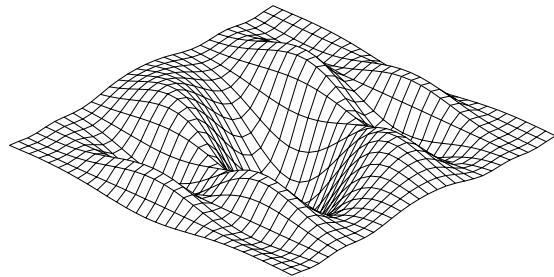


Figure 5.2: Surface plots can be visually very attractive, but generally don't represent quantitative information very well.

either by showing contours of constant alleviation in a *contour plot*, or by mapping the numerical values to a palette of colors in a *false-color plot*. Contour plots are familiar from topographic maps — they can work pretty well, in particular if the data is rather smooth, and one is primarily interested in local properties.

False-color plots are an alternative, and very flexible, technique, which can be used for different tasks and on a wide variety of data sets. To create a false-color plot, all values of the dependent variable (“ z ”) are mapped to a palette of colors. Each data point is then plotted as a region of the appropriate color. Figure 5.3 shows an example, where the “color” has been replaced by a gray scale.

I like false-color plots, because one can represent a lot of information in a them in a way that retains quantitative information. However, false-color plots depend crucially on the quality of the palette, that is the mapping that has been used to associate colors with numeric values.

Let's quickly recap some information on color and computer graphics. Colors for computer graphics are usually specified by a triple of numbers, giving the intensity of their red, green, and blue (RGB) components. Although RGB triples make good sense technically, they are not particularly intuitive. Instead, we tend to think of a color's hue, saturation, and value (that is: luminance or lightness). Conventionally, hue runs through the colors of the rainbow (from red over yellow, green, and blue to magenta). Curiously, the spectrum of hues seems close back onto itself, since magenta smoothly transforms back to red. (Here is the reason for this behavior: the hues in the rainbow spec-

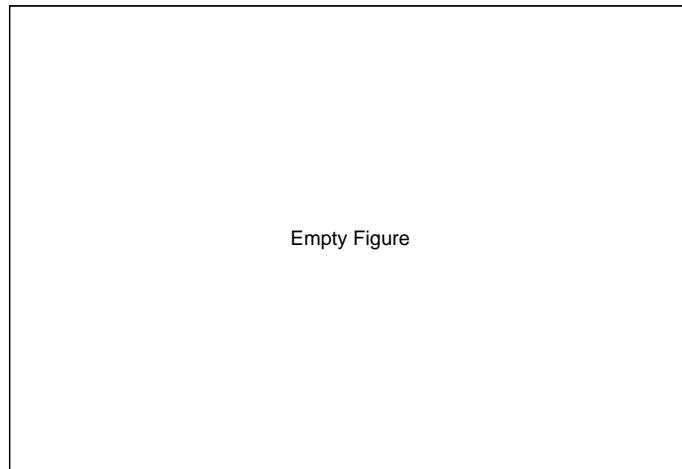


Figure 5.3: TBD

trum are arranged in order of their dominant electromagnetic frequency. For purple/magenta, no frequency dominates; instead, purple is a mixture of low-frequency reds and high-frequency blues.) Most computer graphics programs will be able to generate color graphics using a hue/saturation/value (HSV) triple.

It is surprisingly hard to find reliable recommendations on good palette design, which is even more unfortunate since convenience and what seems like common-sense often lead to particularly *bad* palettes. Here are some ideas and suggestions that you might want to contemplate:

Keep it simple! Very simple palettes using red, white, and blue often work surprisingly well. For continuous color changes, you can use a blue-white-red palette, for segmentation tasks you can use a white-blue-red-white palette with a sharp blue/red transition at the segmentation threshold.

Distinguish between segmentation tasks and the display of smooth changes. Segmentation tasks (such as finding all points that exceed a certain threshold, or finding the locations where the data crosses zero), call for palettes with sharp color transitions at the respective thresholds. On the other hand, if we want to represent smooth changes in a data set, we want to use continuous color gradients. Of course, both aspects can be combined in a single palette: gradients for part of the palette, sharp transitions elsewhere.

Try to maintain an intuitive sense of ordering. Map low values to “cold” colors and higher values to “hot” colors, to provide an intuitive sense of

ordering in your palette. Some examples include the simple blue/red palette and the so-called “heat-scale” (black-red-yellow-white — I’ll discuss in a moment why I don’t recommend the heat-scale for use). Other palettes that convey a sense of ordering (if only by convention) are the “improved rainbow” (blue-cyan-green-yellow-orange-red-magenta) and the “geo-scale” familiar from topographic maps (blue-cyan-green-brown-tan-white).

Place strong visual gradients in regions with important changes. Assume you have a data set with values that span the range from -100 to $+100$, but all the really interesting or important change occurs in the range -10 to $+10$. If you use a standard palette (such as the improved rainbow) for such a data set, the actual region of interest will appear to be all of the same color, with the remainder of the spectrum “wasted” on parts of the data range that are not that interesting. To avoid this, you have to compress the rainbow, such that it maps only to the region of interest. You might want to consider mapping the extreme values (from -100 to -10 and from 10 to 100) to some unobtrusive colors (possibly even a gray scale), and reserve the majority of hue changes for the most relevant part of the data range.

Prefer subtle changes... This is possibly the most surprising recommendation. When creating palettes, there is a natural tendency to “crank it up full”, and to use fully saturated colors at maximal brightness throughout. That’s not necessarily a good idea — the resulting effect is so harsh that details are easily getting lost. Instead, you might want to consider using soft, pastel colors; and even to experiment with mixed hues, instead of working with the pure primaries of the standard rainbow. (Recent versions of Microsoft Excel provide an interesting and easily accessible demonstration for this idea: all default colors offered for shading the background of cells are soft, mixed pastels — to good effect.) Furthermore, the eye is quite good at detecting even subtle variations. In particular when working with luminance-based palettes, small changes are often all that is required.

...but avoid changes that are hard to detect. Some visual changes are just particularly hard to perceive visually. For example, it is basically impossible to distinguish between different shades of yellow, and the transition from yellow to white is even worse! (This is why I don’t recommend the heat-scale, despite its nice ordering property: the bottom third consists of hard-to-distinguish dark reds and the entire upper third consists of very hard-to-distinguish shades of light yellow.)

Use hue- and luminance-dominated palettes for different purposes. In particular, consider a luminance-dominated palette to emphasize fine detail, and use hue- or saturation-based palettes for smooth, large-scale changes. There is some empirical evidence that luminance-based palettes are better suited for images that contain a lot of fine detail, whereas hue-based

palettes are more suitable to bring out smooth, global changes. A pretty striking demonstration of this observation can be found when looking at medical images (surely an application where details matter!): a simple gray-scale (pure luminance!) representation often seems much clearer than a multi-colored representation based on a hue-based rainbow palette. This rule is more relevant to image processing of photographs or similar images (such as the one in the medical example just given) than to the visualization of abstract information that we are considering here, but it is worth to keep in mind.

Don't forget to provide a color-box! No matter how intuitive you think your palette is, unless you provide a color-box, showing the values and the colors they are mapped to, nobody will know for sure what it is you are showing. Always, always, provide one.

One big problem not properly addressed by the recommendations above concerns *visual uniformity*. For example, when creating a palette based on the “improved rainbow” (blue-cyan-green-yellow-red-magenta), by distributing the six primaries at equal distances across the palette and interpolating linearly between them in color space, the fraction of the palette occupied by green appears to be much larger than the fraction occupied by either yellow or cyan. Or when placing a fully saturated yellow next to a fully saturated blue, the blue region will appear to be more intense (that is, saturated) than the yellow. Similarly, the browns that occur in a geo-scale easily appear darker than the other colors in the palette. This is a problem with our *perception* of color: simple interpolations in color space do not necessarily result in visually uniform gradients!

There is a variation of the HSV color space, called the *HCL* (for hue/chroma/luminance) space, which takes visual perception into account to generate visually uniform color maps and gradients. It is more complicated to use than the HSV color space, because not all combinations of hue, chroma, and luminance values exist. For instance, a fully saturated yellow appears lighter than a fully saturated blue, so a palette at full chroma and with high luminance will include the fully saturated yellow, but not the blue. As a result, HCL-based palettes that span the entire rainbow of hues tend naturally towards soft, pastel colors. On the down side, palettes in the HCL space often degrade particularly poorly when reproduced in black-and-white.¹

A special case of false-color plots are geographic *maps*, and cartographers have significant experience developing color schemes for various purposes. Their needs are a little different, and not all of their recommendations may work for general data analysis purposes, but it is worth becoming familiar with their learnings.²

¹An implementation of the transformation required to go between HCL and RGB is available in R and C, in the “colorspace” module available from CRAN.

²An interesting starting point is Cynthia Brewer’s online ColorBrewer at <http://colorbrewer2.org/>.

Finally, I'd like to point out two additional problems with all plots that depend on color to convey critical information:

- Color does not reproduce well. Once photo-copied or printed on a black-and-white laser printer, a false-color plot will become useless!
- Also keep in mind that about ten percent of all men are at least partially color blind and therefore won't be able to make much sense of most images that rely mostly or exclusively on color.

Both are potentially serious enough that you might want to reconsider before relying entirely on color for the display of some information.

In my experience, preparing good false-color plots is often a tedious and time-consuming task. This is one area where better tools would be highly desirable — an interactive tool that I could use to manipulate the palette directly and in real-time would be very nice to have. The same is true for a publicly available set of well-tested palettes.

5.2 A lot at a Glance: Multiplots

The primary concern in all multivariate visualizations is to find better ways to put more "stuff" on a graph. Besides color (see the previous section), there are basically two ways we can go about this: either we make the graph elements themselves richer, so that they can convey additional information beyond their position on the graph; or we put several similar graphs next to each other and vary those variables which are not explicitly displayed in a systematic fashion from one sub-graph to the next. The first idea leads to *glyphs*, which we will introduce later in this chapter, while the latter idea leads to scatter-plot matrices and co-plots.

5.2.1 The Scatter-Plot Matrix

For a *scatter-plot matrix* (occasionally abbreviated SPLOM), we construct all possible two-dimensional scatter plots from a multivariate data set, and plot them together in a matrix format (figure 5.4). We can now scan all of the graphs for interesting behavior, such as marked correlation between any two variables, and the like.

The data set shown in figure 5.4 consists of seven different properties of a sample of 250 wines.³ It is not at all clear, how these properties should relate to each other, but studying the scatter-plot matrix, we can make a few interesting observations. For example, we can see that sugar content and density are positively correlated: if the sugar content goes up, so does the density. The opposite is true for alcohol content and density: as the alcohol content goes up, density

³The data can be found in the "Wine Quality" data set, available at the UCI Machine Learning repository at <http://www.ics.uci.edu/~mlearn/MLRepository.html>.

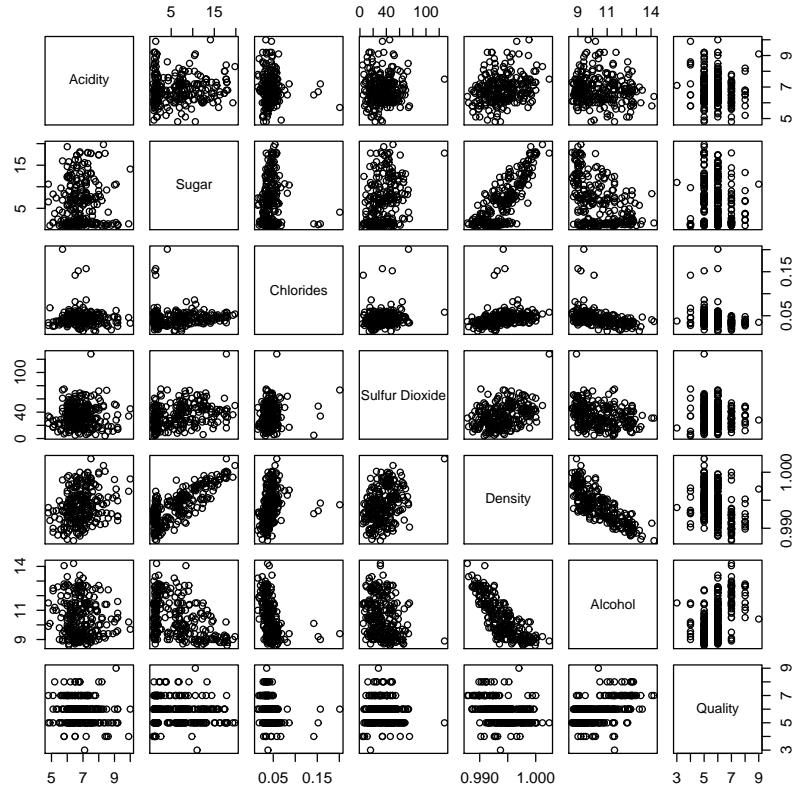


Figure 5.4: TBD

goes down. Neither of these observations should come as a surprise (sugar syrup has a higher density of water, alcohol a lower). What is potentially more interesting is that the wine quality seems to increase with increasing alcohol content: apparently, more potent wines are considered better!

One important detail that is easy to overlook is that all graphs in each row or column show the same plot range, in other words, they use *shared scales*. This makes it possible to compare graphs across the entire matrix.

The scatter plot matrix is symmetric across the diagonal: the sub-plots in the lower left are equal to the ones in the upper right, but rotated by ninety degrees. It is nevertheless customary to plot both versions, because this allows to scan a single row or column in its entirety to see how one quantity relates to every one of the other quantities.

Scatter plot matrices are easy to prepare and easy to understand, and that

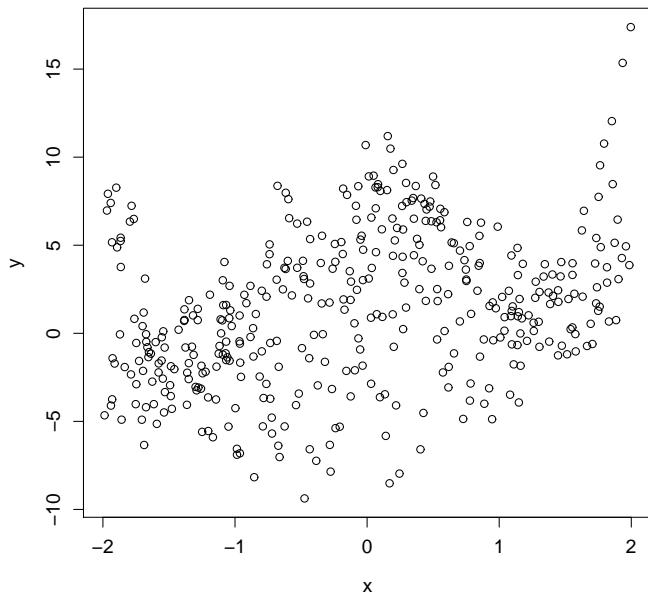


Figure 5.5: A projection of a tri-variate data set onto the xy -plane. How does the data vary with the third variable?

makes them very popular, but I think they can be overused. Once we have more than about half a dozen variables, the individual subplots become too small to recognize anything useful, in particular if the number of points is large (a few hundred points). Nevertheless, they are a convenient way to get a quick overview and to find view points (that is: variable pairings) that deserve a closer look.

5.2.2 The Co-Plot

In contrast to scatter plot matrices, which always show all data points, but *project* them onto different surfaces of the parameter space, *co-plots* (short for “conditional plots”) show various *slices* through the parameter space, such that each slice contains only a subset of the data points. The slices are taken in a systematic manner, and we can form an image of the entire parameter space by mentally glueing the slices back together again. (The salami principle.)

Figure 5.5 shows a tri-variate data set, projected onto the two-dimensional xy -plane. Although there is clearly structure in the data, no clear pattern emerges. In particular, the dependence on the third parameter is entirely obscured!

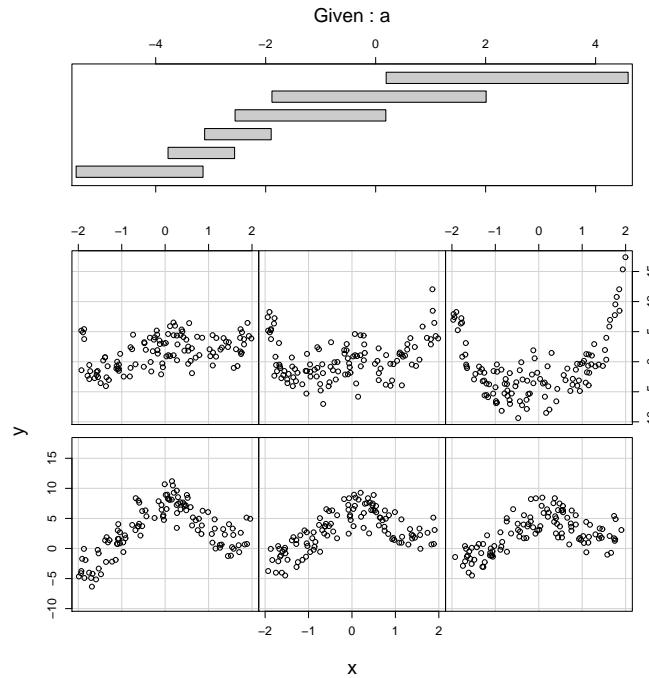


Figure 5.6: A co-plot of the same data as in figure 5.5. Each scatter plot shows only the data points for a certain range of a values; the top panel shows the corresponding values of a . (The scatter plot for the smallest value of a is in the bottom left corner, the one for the largest value of a is in the upper right.)

In figure 5.6 we see a co-plot of the same data set, sliced or *conditioned* on the third parameter (called a). The bottom part of the graph shows six slices through the data corresponding to different ranges of a — note that the slice for the *smallest* values of a is in the bottom left, and the one for the *largest* values of a is in the upper right hand corner. As we look at the slices, the structure in the data stands out very clearly, and we can easily follow the dependence on the third parameter a as well.

The top part of figure 5.6 shows the range of values that a takes on for each of the slices, and if you look closely, you will find that there are some subtle issues hidden (or rather: revealed!) by this panel, because it provides information on the details of the slicing operation!

Two decisions need to be made in regards to the slicing:

- By what method should the overall parameter range be cut into slices?
- Should slices overlap or not?

In many ways the most “natural” answer to these questions would be to cut the entire parameter range into a set of adjacent intervals of equal width. It is interesting to observe (by looking at the top panel in figure 5.6) that in the example graph a different decision has been made in regards to both questions! The slices are not of equal width in the range of parameter values that they span; instead, they have been made in such a way that each slice contains *the same number of points*. In addition, the slices are not adjacent, but partially overlap each other.

The first decision (to have each slice contain the same number of points, instead of spanning the same range of values) is particularly interesting, because it provides additional information on the way the values of the parameter a are distributed: for example, we can see that that large values of a (larger than about $a = -1$) are relatively rare, whereas values of a between -4 and -2 are much more frequent. This kind of behavior would be much harder to recognize precisely if we had chopped the interval for a into six slices of equal width. The other decision (to make the slices overlap partially) is more important for small data sets: in this case, each slice ends up containing so few points that the structure becomes hard to see. Having the slices overlap makes the data “go farther” than if the slices were entirely disjunct.

Co-plots are particularly useful if some of the variables in a data set are clearly “control” variables, because co-plots provide a systematic way to study the dependence of the remaining (“response”) variables on the controls.

5.2.3 Variations

The ideas behind scatter-plot matrices and co-plots are pretty generally applicable, and you can develop different variants, depending on your needs and tastes. Here are some ideas:

- In the standard-scatter plot matrix, half the individual graphs are redundant. You can remove the individual graphs from one half of the overall matrix and replace it with something different — the numerical value of the appropriate correlation coefficient, for example. Be aware, though, that you will be losing the ability to visually scan a full row or column to see how the corresponding quantity correlates with all other variables.
- Similarly, you can place a histogram showing the distribution of values for the quantity in question on the diagonal.
- The “slicing” technique used in co-plots can be used with other graphs besides scatter plots — for instance, you might want to use it with rank-order plots (see chapter 2), where the conditioning “parameter” is some quantity not explicitly shown in the rank-order plot itself.
- Finally, co-plots can be extended to *two* conditioning variables, leading to a matrix of individual slices.

By their very nature, all multiplots consist of many individual plot elements, sometimes with non-trivial interactions (such as the overlapping slicing for certain co-plots). Without a good tool, which handles most of these issues automatically, these plot types loose most of their appeal. For the plots in this section I used R (the statistical package), which contains support for both scatter plot matrices and co-plots as built-in functionality.

5.3 Composition Problems

Many data sets describe a *composition problem*, in other words, they describe how some overall quantity is composed of its parts. Composition problems pose some special challenges, because there are two *different* aspects of the data that we often want to visualize simultaneously: on the one hand, we are interested in the relative magnitude of the different components, on the other hand, we also care about their absolute size.

For one-dimensional problems, this is not too difficult (see chapter 2): we can use a histogram or a similar graph to display the absolute size for all components; to visualize the relative contribution that each component makes to the total, we can use a cumulative distribution plot or even the much-maligned pie chart.

But once we add additional variables into the mix, things can get pretty ugly. Two particular problems stand out: how to visualize *changes* to the composition over time, and how to depict the breakdown of an overall quantity along *multiple axes* at the same time.

5.3.1 Changes in Composition

To understand the difficulties in tracking compositional problems over time, imagine a company that makes five products, labeled A, B, C, D, and E. As we track the daily production numbers over time, there are two different questions that we are likely to be interested in: on the one hand, we'd like to know how many items were produced overall; on the other hand, we would like to understand how the item mix is changing over time.

Figures 5.7, 5.8, and 5.9 show three attempts to plot this kind of data. In the figure 5.7, I simply show the absolute numbers produced per day for each of the five product lines. That's not ideal — the graph looks messy, with some of the lines obscuring each other. Moreover, it is not possible to understand from this graph how the total number of items changes over time. Test yourself: what would you say, does the total number of items go up over time, or does it go down, or do you think it stays about even?

In figure 5.8, I show a *stacked plot* of the same data: the daily numbers for each product are added to the numbers for the products that appear lower down in the diagram — in other words, the line labeled B gives the number of items produced in product lines A *and* B. The top-most line in this diagram shows the total number of items produced per day (and answers the question

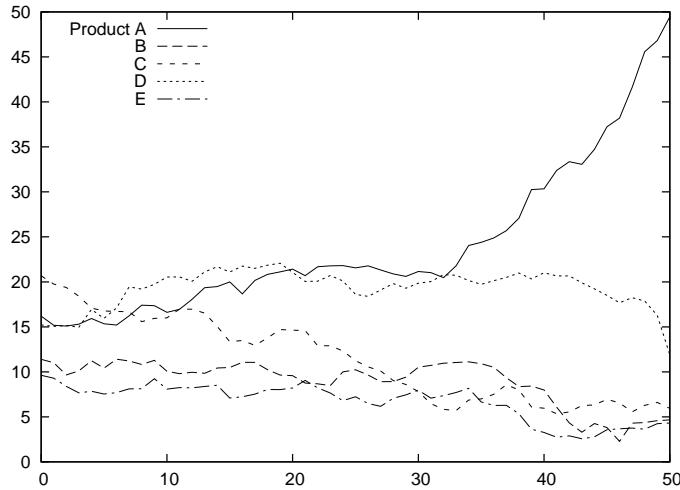


Figure 5.7: Absolute number of items produced per product line and day.

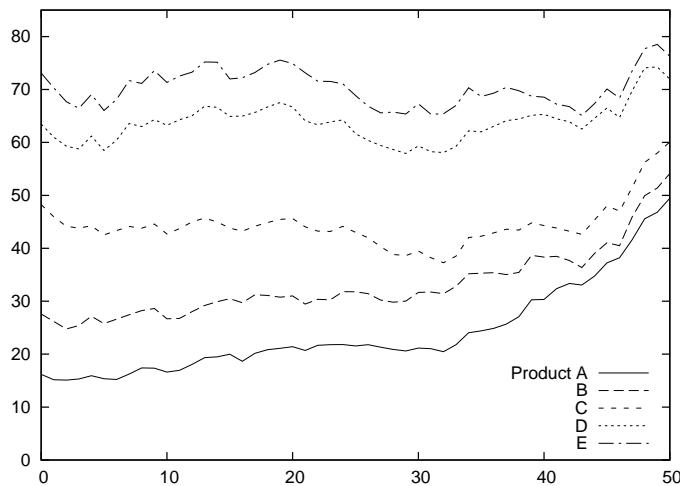


Figure 5.8: Number of items produced per product line and day — stacked graph.

posed in the previous paragraph: the total number of items does *not* change appreciably over the long run — a possibly surprising observation, given the appearance of figure 5.7).

Such a stacked plot can be very compelling, because it has intuitive appeal

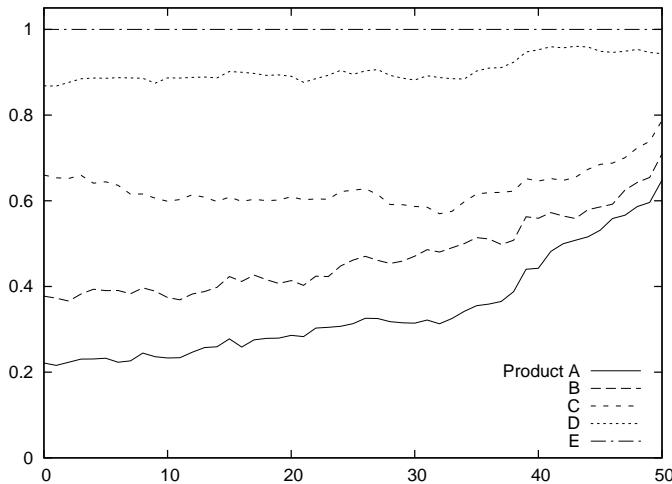


Figure 5.9: Relative contribution that each product line makes to the total — stacked graph.

and appears to be clear and uncluttered. But in reality it tends to hide the details in the development of the individual components, because the changing baseline makes comparison difficult, if not impossible. For example, from figure 5.7 it was pretty clear that production of item D increased for a while, but then dropped rapidly over the last 5 to 10 days. We would never guess this from figure 5.8: the strong growth of product line A masks the smaller changes in the other components. (This is why you should order the components in a stacked graph in ascending order of variation — which quite intentionally was *not* done in figure 5.8.)

Figure 5.9 shows yet a different attempt to visualize this data: this figure again is a stacked graph, but now we are not looking at the absolute numbers of items produced, but at the relative fraction that each product line contributes to the daily total. Because the change in the total number of items produced has been eliminated, this graph can help us understand how the item mix varies over time (although we still have the changing baseline problem common to all stacked graphs). On the other hand, information about the total number of items produced has been lost.

In the end, I don't think any one of these graphs succeeds particularly well. No single graph can capture both of our conflicting goals (namely to monitor both absolute numbers and relative contributions) and be clear and visually attractive at the same time.

I think an acceptable solution for this sort of problems will always have to involve a combination of graphs — for example, one for the total number of items produced, and another one for the relative item mix. Furthermore,

despite their aesthetic appeal, stacked graphs should be avoided, because they make it too difficult to recognize relevant information in the graph. A plot such as figure 5.7 may appear messy, but it can nevertheless be read accurately and reliably.

5.3.2 Multidimensional Composition: Tree- and Mosaic-Plots

Coming soon

5.4 Novel Plot Types

Most of the graph types I have described so far (with the exception of mosaic plots) can be described as “classical”: they have been around for years. In this section, I am going to discuss a few techniques which are much more recent — or at least, which have only recently received greater mindshare.

5.4.1 Glyphs

We can include additional information in any simple plot (such as a scatter plot) if we replace the simple symbols used for individual data points with *glyphs*, that is, more complicated symbols that are able to express additional bits of information by themselves.

An almost trivial application of this idea occurs if we put two data sets on a single scatter plot and use different symbols (such as squares and crosses) to mark the data points from each data set. The symbols themselves carry meaning, but only a very simple, categorical one (namely whether the point belongs to the one data set or the other).

But if we make the symbols more complicated, they can express more data. Textual labels (letters and digits) are often surprisingly effective when it comes to conveying more information — although distinctly low-tech, this is a technique to keep in mind!

The next step up in sophistication are arrows, representing both a direction and a magnitude, but we need not stop there. Each symbol can be a fully formed graph (such as a pie chart or a histogram) all by itself. And even that is not the end — probably the craziest idea in this realm are “Chernoff Faces”, where different quantities are encoded as *facial features* (such as: size of the mouth, or distance between the eyes), and these faces are now used as symbols on a plot!

As you can see, the problem lies not so much in putting more information on a graph, but in being able to interpret the result in a useful manner! And that seems to depend mostly on the *data*, in particular on the presence of large-scale, regular structure in it. If such structure is missing, plots using glyphs can be very hard to decode, and quite possibly useless.

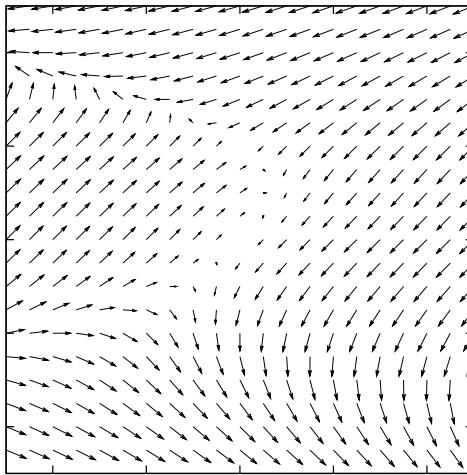


Figure 5.10: Simple glyphs: using arrows to indicate both direction and magnitude of a field. Notice that the variation in the data is smooth, and the data itself is taken on a regular grid.

Figures 5.10 and 5.11 show two extreme examples. In figure 5.10 we visualize a four-dimensional data set using arrows (each point of the two-dimensional plot area has both a direction and a magnitude, hence the total number of dimensions is four). You can think of the system as flow in a liquid, or as electrical or magnetic field lines, or as deformations in an elastic medium — it does not matter, the overall nature of the data becomes quite clear. Figure 5.11 on the other hand, is an entirely different matter! Here, we are dealing with a data set in seven dimensions: the first two are given by the position of the symbol on the plot, the remaining five are represented through the distortions of a five-edged polygon. Although we can make out some regularities (for example, the shapes of the symbols in the lower left-hand corner are all quite similar and different from the shapes elsewhere), this graph is very hard to read, and I don't think the overall structure of the data is revealed very well by it. Also keep in mind that the appearance of the graph will change if we map a different pair of variables to the main axes of the plot, and even if we change the order of variables in the polygons.

5.4.2 Parallel Coordinate Plots

As we have seen, a scatter plot can show two variables; if we use glyphs, we can show more, but not all variables are treated equally (some are encoded in the glyphs, some are encoded by the position of the symbol on the plot). Using *parallel coordinate plots*, we show *all* variables of a multivariate data set

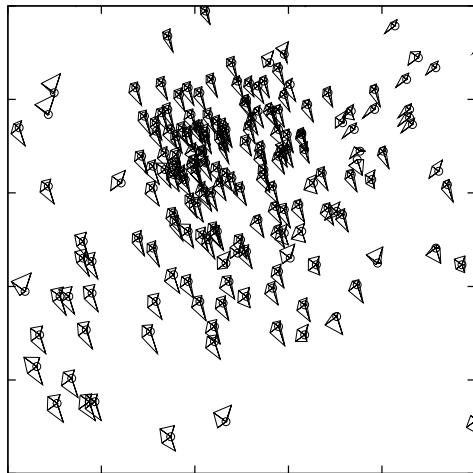


Figure 5.11: More complicated glyphs: each polygon encodes five different variables, its position on the plot adds another two.

on equal footing! The price we pay is that we end up with a graph that is neither pretty nor particularly intuitive, but that can be useful for exploratory work nevertheless.

In a regular scatter plot in two (or even three) dimensions, the coordinate axes are at right angles to each other. In a parallel coordinate plot, the coordinate axes instead are *parallel* to each other. For every data point, its value for each of the variables is marked on the corresponding axis, and finally, all these points are connected with lines. Because the axes are parallel to each other, we don't run out of spatial dimensions, and we can therefore have as many of them as we need. Figure 5.12 shows what a single record looks like in such a plot, and figure 5.13 shows the entire data set. Each record consists of nine different quantities (labeled A through J).

The main use of parallel coordinate plots is to find clusters in high-dimensional data sets. For example in figure 5.13, we can see that the data forms two clusters for the quantity labeled B: one around 0.8, and one around 0. Furthermore, we can see that most records for which B is zero, tend to have higher values of C than those that have a B near 0.8. And so on.

A couple of technical points should be noted about parallel coordinate plots:

- First of all, you usually want to rescale the values in each coordinate to the unit interval, using the linear transformation (also see B):

$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

This is not mandatory however — there may be situations where you

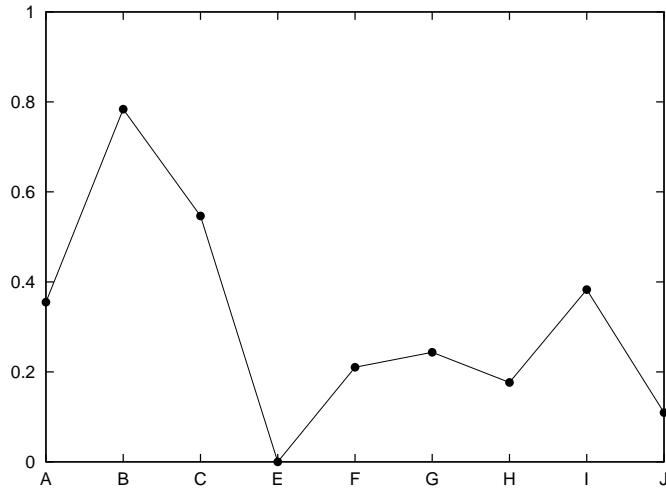


Figure 5.12: A single record (that is: a single data point) from a multivariate data set shown in a parallel coordinate plot.

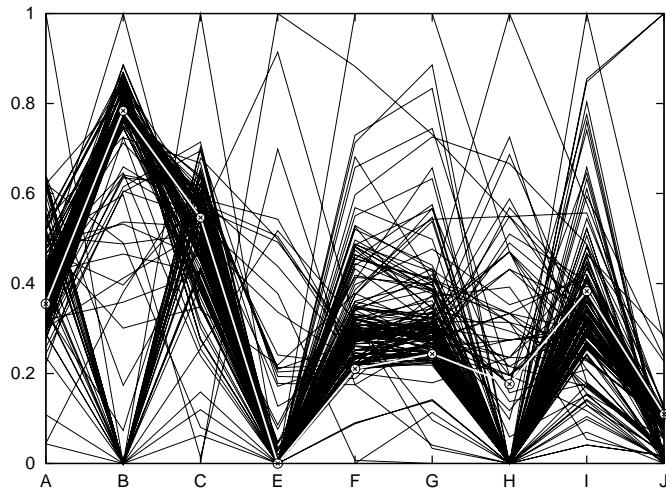


Figure 5.13: All records from the data set shown in a parallel coordinate plot. The record shown in figure 5.12 is highlighted.

care about the absolute positions of the points along the coordinate axis; or about scaling to a different interval.

- The appearance of parallel coordinate plots depends strongly on the or-

der in which the coordinate lines are drawn: rearranging them can hide or reveal structure. Ideally, you are using a tool that lets you reshuffle the coordinate axis interactively.

- In particular for larger data sets (several hundreds of points or more), overplotting of lines becomes a problem. One way to deal with this is through alpha blending: lines are shown as semi-transparent and their visual effects are combined where they overlap each other.
- Similarly, it is often highly desirable to be able to select a set of lines and highlight them throughout the entire graph, for example to see how data points that are clustered in one dimension are distributed in the other dimensions.
- Instead of combining points on adjacent coordinate axes with straight lines that have sharp kinks at the coordinate axes, one can use smooth lines that pass the coordinate axes without kinks.

All of these issues really are *tool* issues, and in fact, parallel coordinates don't make sense without a tool that supports them natively and includes good implementations of the features mentioned above. The consequence here is that the purpose of parallel coordinate plots is less a finished, static graph but instead the interactive process of exploration of a data set.

Parallel coordinate plots still seem pretty novel. The idea itself has been around for about 25 years, but even today, good (!) tool support for parallel coordinates is still far from common-place.

What is not clear yet is how useful parallel coordinate plots really are. On the one hand, the concept seems very straightforward and easy enough to use. On the other hand, I have found the experience of actually trying to apply them frustrating and not very fruitful. It is easy to get bogged down in technicalities of the plot (ordering and scaling of coordinate axes), with little real, concrete insight resulting in the end. The erratic tool situation of course does not help. I wonder whether more computationally intensive methods (such as the Principal Component Analysis — see chapter 14) do not give a better return on investment overall. But the jury is still out.

5.5 Interactive Explorations

Coming soon

5.6 Toolbox

Multivariate graphs tend to be complicated and therefore require good tool support even more strongly than other forms of graphs. In addition, some

multivariate graphics are highly specialized (think mosaic plots!), and can not easily be prepared using general-purpose plotting tool.

That being said, the tool situation is questionable, at best. Here are three different starting points for exploration — each with its own set of difficulties.

5.6.1 R

R is not a plotting tool per se, it is a statistical analysis package and a full development environment as well. However, R has always included pretty extensive graphing capabilities. R is particularly strong at “scientific” graphs — rather straightforward-looking, but very accurate line diagrams.

Because R is not just a plotting tool, but instead a full data manipulation and programming environment, its learning curve is pretty steep: you have to know a lot of different things to be able to do anything. But once you are up and running, the large number of advanced functions that are already built-in can make working with R very productive. For example, the scatter-plot matrix in figure 5.4 was generated using the following commands:

```
d <- read.delim( "wines" , header=T )

pairs(d)

dev.copy2eps( file="splom.eps" )
```

(the R command `pairs()` generates a plot of all pairs — in other words, a scatter-plot matrix), and the scatter plot figure 5.5 and the co-plot in figure 5.6 were generated using:

```
d <- read.delim( "data" )

plot( y ~ x, data=d )
dev.copy2eps( file='coplot1.eps' )

coplot( y ~ x | a, data=d )
dev.copy2eps( file='coplot2.eps' )
```

Note that these are the *entire* command sequences, including reading the data from file and writing the graph back to disk! We'll have a bit more to say about R in the Workshop sections of chapter 10 and chapter 14.

5.6.2 ggobi and Mondrian

If you want to explore some of the more “novel” graph ideas, such as parallel coordinate and mosaic plots, or want to try out interactive ideas such “brushing” or grand tours, you might want to check out the open source tools ggobi and Mondrian (www.ggobi.org/ and www.rosuda.org/mondrian/).

Both tools are academic research projects, and it shows: they are technology demonstrators, intended to try out and experiment with new graph ideas, but neither is anywhere near production strength. (Both are rather fussy about the required data input format, their graphical user interfaces are clumsy, neither of them includes a proper way to export graphs to file: if you want to save a plot, you have to take a screenshot. The interactive brushing features in ggobi are slower than paper tape, making it in practice unusable for realistically sized data sets. There are some lessons here, besides the intended ones, to be learned about the design of tools for statistical graphics.)

5.6.3 Python Chaco Library

The Chaco library (<http://code.enthought.com/projects/chaco/>) is a Python library for two-dimensional plotting. In addition to the usual line symbol drawing capabilities, it includes easy support for color and color manipulation, and — more importantly — for real-time user interaction.

Chaco is part of the Enthought Tool Suite, which is developed by Enthought, Inc. and is available under a BSD-style license.

5.7 Further Reading

- *Graphics of Large Datasets: Visualizing a Million.* Antony Unwin, Martin Theus, Heike Hofmann. Springer. 2006.

This is a very modern book, describing in many ways the “state of the art” in statistical data visualization. Mosaic plots, glyph plots, parallel coordinate plots, grand tours — you will find it discussed here. Unfortunately, over all the progress, the basics are being neglected: standard tools like logarithmic plots are never even mentioned, and simple things like labels are frequently messed up. Nevertheless, there is a lot to learn here: both how to do it, and how *not* to do it.

- *The Elements of Graphing Data.* William S. Cleveland. 2nd ed., Hobart Press. 1994.

This book provides an interesting counter-point to the book by Unwin and co-workers. Cleveland’s graphs often look pedestrian, but he thinks more deeply about ways to incorporate more and more quantitative information in a graph than almost anybody else. What stands out in his works is that he explicitly takes human perception into account as a guiding principle when developing new graphs. My treatment of scatter plot matrices and co-plots is heavily influenced by his careful treatment.

- *Gnuplot in Action — Understanding Data with Graphs.* Philipp K. Janert. Manning Publications. 2010.

Chapter 9 of this book contains additional details and examples for the

use of color to prepare false-color plots, including explicit recipes to create such plots using gnuplot. But the principles are valid more generally, even if you use different tools.

- *Why Should Engineers and Scientists Be Worried About Color?*. B. E. Rogowitz, L. A. Treinish. <http://www.research.ibm.com/people/l/lloyd/color/color.HTM>. 1995.
This paper contains important lessons for false-color plots, including the distinction between segmentation and smooth variation, as well as the difference between hue- and luminance based palettes. The examples were prepared using IBM's (now open sourced) OpenDX graphical Data Explorer.
- *Escaping RGBland: Selecting colors for statistical graphics*. A. Zeileis, K. Hornik, P. Murrell. <http://statmath.wu.ac.at/~zeileis/papers/Zeileis+Hornik+Murrell-2009.pdf>. 2009.
A more recent paper on the use of color in graphics, which emphasizes the importance of perception-based color spaces, such as the HCL model.

1969616

Chapter 6

Intermezzo: A Data Analysis Session

Occasionally I get the question: “How do you actually work?” or “How do you come up with this stuff?”, and as an answer, I wanted to take you on a tour through a new data set. I will use gnuplot, which is my preferred tool for this kind of interactive data analysis — you will see why. And I will share my observations and thoughts as we go along.

6.1 A Data Analysis Session

The data set is a classic: the CO₂ measurements above Mauna Loa on Hawaii. The inspiration for this section comes from W. Cleveland’s “Elements of Graphical Analysis”¹, but the approach is entirely mine.

First question: what’s in the data set? I see that the first column represents the date (month and year), the second contains the measured CO₂ concentration in parts per million. Here are the first few lines:

Jan-1959	315.42
Feb-1959	316.32
Mar-1959	316.49
Apr-1959	317.56
...	

The measurements are regularly spaced (in fact, monthly), so I don’t need to parse the date in the first column. I simply plot the second column by itself. (In the figure, I have drawn tic labels on the horizontal axis for clarity, but I am omitting the commands required here — they are not essential).

¹*The Elements of Graphing Data*. William S. Cleveland. Hobart Press. 1994.
The data itself (in a slightly different format) is available from StatLib: <http://lib.stat.cmu.edu/datasets/visualizing.data.zip> and from many other places around the Web.

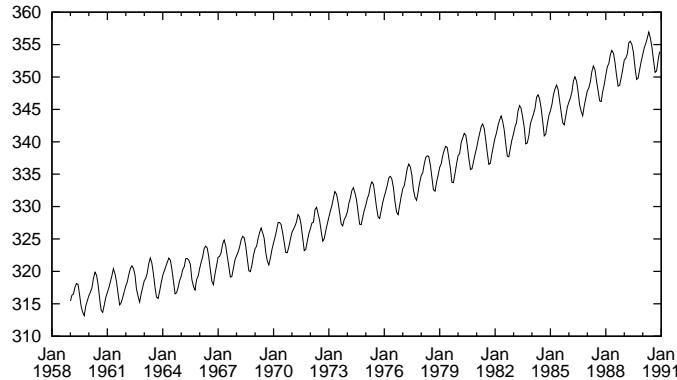


Figure 6.1: The first look at the data: plot "data" u 1 w 1

```
plot "data" u 1 w 1
```

I see a rather regular short-term variation, overlaid on non-linear upwards trend. (See figure 6.1.)

The coordinate system is not convenient for mathematical modeling: the x-axis is not numeric, and for modeling purposes it is usually helpful if the graph goes through the origin. So, let's make it so, by subtracting the vertical offset from the data and expressing the horizontal position as the number of months since the first measurement. (This corresponds to the line number in the data file, which is accessible in a gnuplot session through the pseudo-column with column number 0.)

```
plot "data" u 0:($2-315) w 1
```

A brief note on the command: the specification after the u (short for `using`) gives the columns to be used for the *x* and the *y* coordinates, separated by a colon. Here, we use the line number (which is in the pseudo-column zero) for the *x* coordinate. Also, we subtract the constant offset 315 from the values in the second column and use the result as the *y* value. Finally, we plot the result with lines (short: `w 1`, instead of using points or other symbols. (See figure 6.2.)

The most predominant feature is the trend. What can we say about it? First of all, the trend is nonlinear: if we ignore the short term variation, the curve is convex downwards. That suggests a power law with an as-yet unknown exponent: x^k . All power law functions go through the origin $(0, 0)$ and through the point $(1, 1)$. We already made sure that the data passes through the origin, but to fix the upper-right-hand-corner, we need to rescale both axes: if x^k goes through $(1, 1)$, then $b \left(\frac{x}{a}\right)^k$ goes through (a, b) .

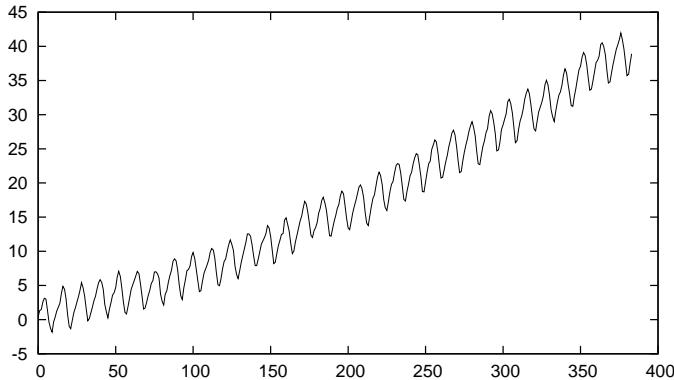


Figure 6.2: Making the x values numeric and subtracting the constant vertical offset: `plot "data" u 0:(\$2-315) w 1`

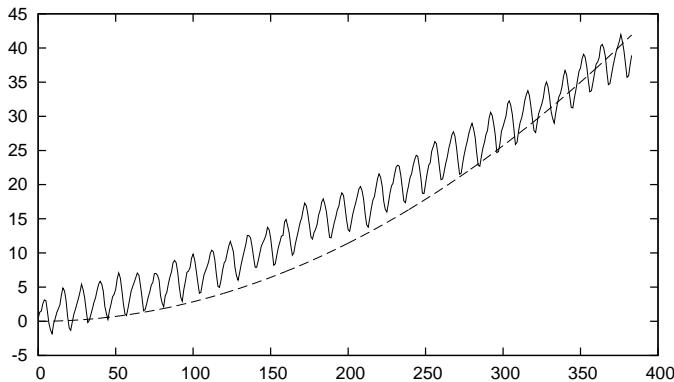


Figure 6.3: Adding a function: `plot "data" u 0:(\$2-315) w 1, 35*(x/350)**2`

What's the value for the exponent k ? All I know about it right now is that it must be greater than 1 (because the function is convex). Let's try: $k = 2$. (See figure 6.3.)

```
plot "data" u 0:(\$2-315) w 1, 35*(x/350)**2
```

Not bad at all! The exponent is a bit too large — some fiddling suggests that $k = 1.35$ would be a good value. (See figure 6.4.)

```
plot "data" u 0:(\$2-315) w 1, 35*(x/350)**1.35
```

To verify that, let's plot the residual; that is, we subtract the trend from the data and plot what's left. If our guess for the trend is correct, the residual

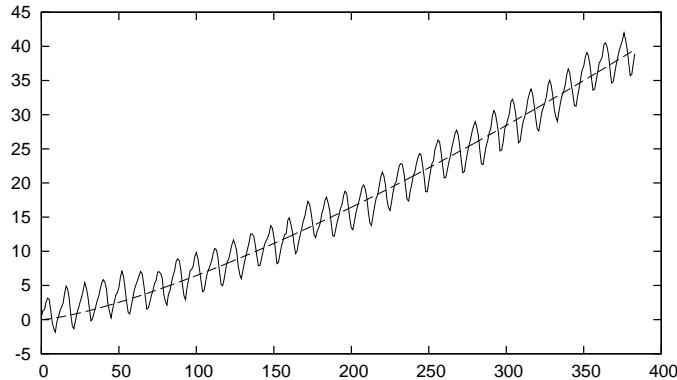


Figure 6.4: Getting the exponent right: $f(x) = 35 \left(\frac{x}{350}\right)^{1.35}$

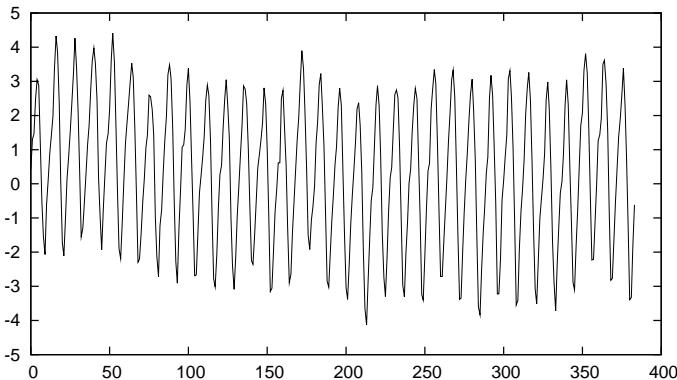


Figure 6.5: The residual, after subtracting the function from the data.

should not exhibit any trend itself anymore — it should just straddle $y = 0$ in a balanced fashion. (See figure 6.5.)

```
plot "data" u 0:(\$2-315 - 35*(\$0/350)**1.35) w 1
```

It can be hard to see the longer-term trend in this data, therefore we may want to approximate it by a “smoother” curve. We can use the weighted-spline approximation built into gnuplot for that purpose. It takes a third parameter, which is a measure for the smoothness: the smaller the third parameter, the smoother the resulting curve is; the larger the third parameter, the more closely does the spline follow the original data.

```
plot "data" u 0:(\$2-315 - 35*(\$0/350)**1.35) w 1,
      "" u 0:(\$2-315 - 35*(\$0/350)**1.35):(0.001) s acs w 1
```

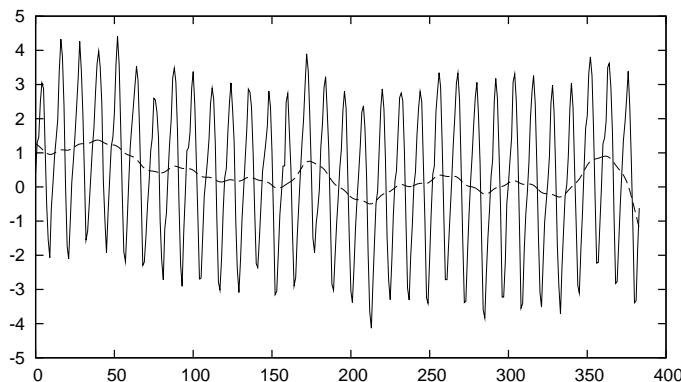


Figure 6.6: Plotting a smoothed version of the residual together with the unsmoothed residual to determine whether there is any systematic trend remaining in the residual.

At this point, the expression for the function that we use to approximate the data has become unwieldy, and it now makes sense to define it as a separate function:

```
f(x) = 315 + 35*(x/350)**1.35
plot "data" u 0:(\$2-f(\$0)) w 1, "" u 0:(\$2-f(\$0)):(0.001) s acs w 1
```

From the smoothed line, we can see that the overall residual is pretty much flat and straddles zero. Apparently we have captured the overall trend quite well: there is little evidence of a systematic drift remaining in the residuals.

With the trend taken care of, the next feature to tackle is the seasonality. The seasonality seems to consist of rather regular oscillations, so we should try some combination of sines and cosines. The data pretty much starts out at $y = 0$ for $x = 0$, so we can try a sine by itself. To make a guess for its wavelength, we realize that the data is meteorological and has been taken on a monthly basis — maybe there is a year-over-year periodicity? This would imply that the data is the same every 12 data points. In other words, a full period of the sine, which corresponds to 2π , should equal a horizontal distance of 12 points. For the amplitude, the graph suggests a value close to 3. (See figure 6.7.)

```
plot "data" u 0:(\$2-f(\$0)) w 1, 3*sin(2*pi*x/12) w 1
```

Right on! In particular the guess for the wave length worked out really well. Which makes sense, given the origin of the data.

Let's take residuals again, employing splines to see the bigger picture as well (see figure 6.8):

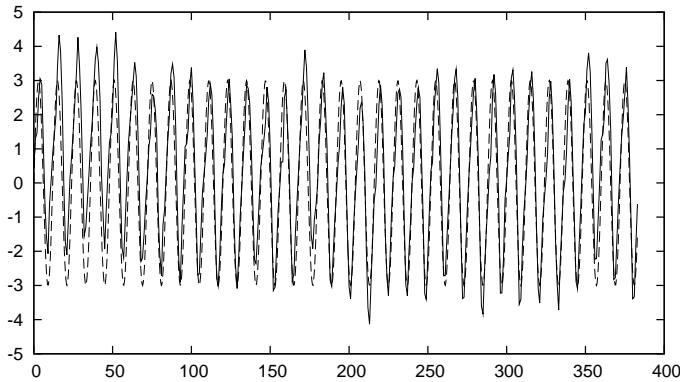


Figure 6.7: Fitting the seasonality with a sine wave: $3 \sin(2\pi \frac{x}{12})$

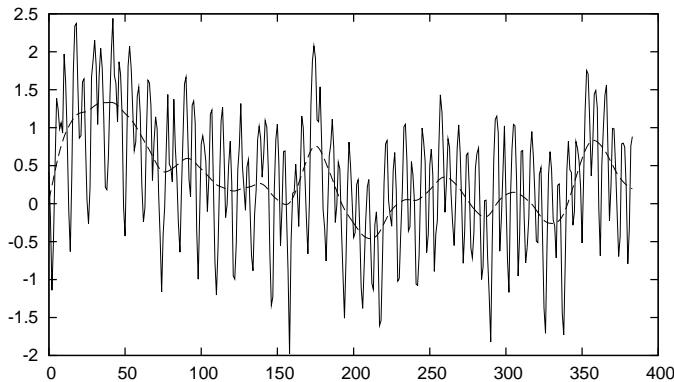


Figure 6.8: Residuals after subtracting both trend and seasonality.

```
f(x) = 315 + 35*(x/350)**1.35 + 3*sin(2*pi*x/12)
plot "data" u 0:(\$2-f(\$0)) w 1, "" u 0:(\$2-f(\$0)):(0.001) s acs w 1
```

Pretty good, but not good enough. There is clearly some regularity still left in the data, but at a higher frequency than the main seasonality. Let's zoom in on a smaller interval of the data to take a closer look. The data in the interval [60:120] appears particularly regular, so let's look there (see figure 6.9):

```
plot [60:120] "data" u 0:(\$2-f(\$0)) w 1p, "" u 0:(\$2-f(\$0)):(0.001) s acs w 1
```

I have indicated the individual data points, using gnuplot's `linespoints` (`1p`) style. We can now count the number of data points between the main valleys in the data: 12 points. So that is the main seasonality. But it appears as

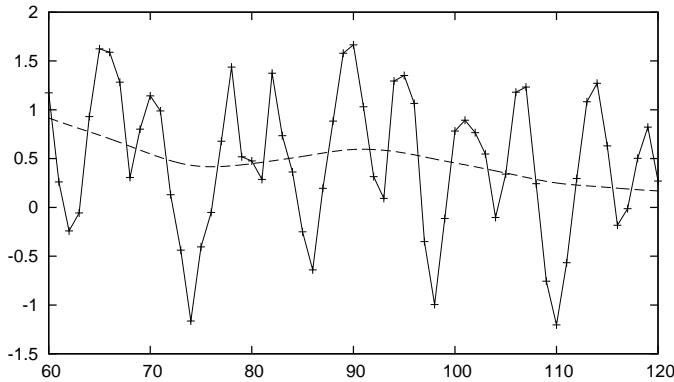


Figure 6.9: Zooming in for a closer look. Individual data points are marked by symbols.

if between any two primary valleys, there now is exactly one secondary valley. Of course: higher harmonics! The original seasonality had a period of exactly 12 months, but its shape was not entirely symmetric: it raising flank comprised 7 months, but the falling flank only 5 (as you can see if you zoom in on the original data with only the trend removed). This kind of asymmetry implies that the seasonality cannot be represented through a simple sine wave alone, but that we have to take into account higher harmonics, that is sine functions with frequencies which are integer multiples of the primary seasonality. So, let's try the first higher harmonic, again punting a little on the amplitude. (See figure 6.10.)

```
f(x) = 315 + 35*(x/350)**1.35 + 3*sin(2*pi*x/12) - 0.75*sin(2*pi*$0/6)
plot "data" u 0:(\$2-f($0)) w 1, "" u 0:(\$2-f($0)):(0.001) s acs w 1
```

Now we are really pretty close. Look at the residual, in particular for values of x greater than about 150. The data starts to look quite “random”, although there is some systematic behavior for x in the range [0:70], which we don't really capture. Let's put on some constant ranges for comparison. (See figure 6.11.)

```
plot "data" u 0:(\$2-f($0)) w 1, "" u 0:(\$2-f($0)):(0.001) s acs w 1, 0, 1, -1
```

It seems to me that the residual is skewed towards positive values, so let's adjust the vertical offset by 0.1. (See figure 6.12.)

```
f(x) = 315 + 35*(x/350)**1.35 + 3*sin(2*pi*x/12) - 0.75*sin(2*pi*$0/6) + 0.1
plot "data" u 0:(\$2-f($0)) w 1, "" u 0:(\$2-f($0)):(0.001) s acs w 1, 0, 1, -1
```

That's now really pretty close. You should notice how small the last adjustment was — we started out with data ranging from 300 to 350, and now we are

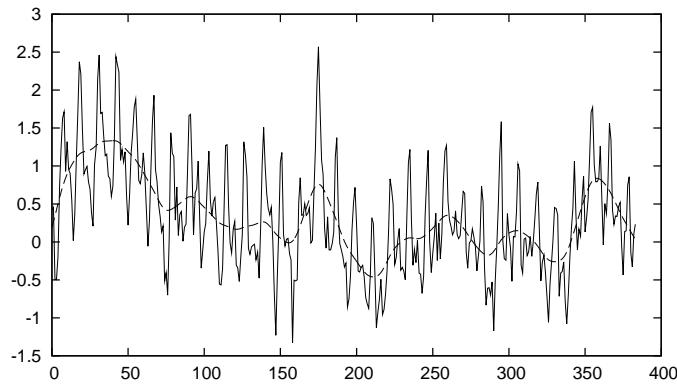


Figure 6.10: Residual after removing trend, and the first and second harmonic of the seasonality.

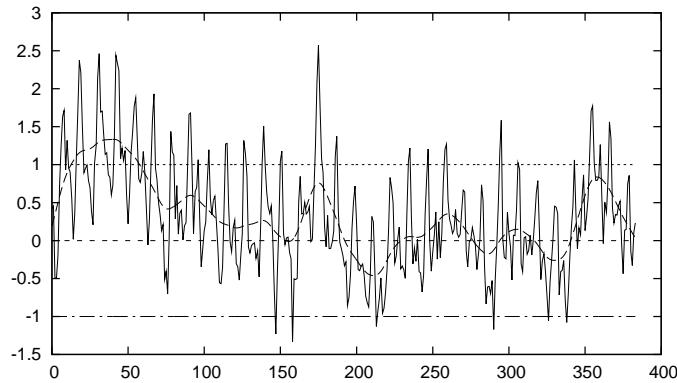


Figure 6.11: Adding some grid lines for comparison.

at the point to make adjustments to the parameters on the order of 0.1. Also note how small the residual is by now: mostly in the range from -0.7 to 0.7. That's only about 3% of the total variation in the data.

Finally, let's look at the original data again, this time together with our analytical model. (See figure 6.13.)

```
f(x) = 315 + 35*(x/350)**1.35 + 3*sin(2*pi*x/12) - 0.75*sin(2*pi*$0/6) + 0.1
plot "data" u 0:2 w 1, f(x)
```

All in all, pretty good.

So, what's the point here? The point is that we started out with nothing — no idea at all what the data looked like. And then, layer by layer, we peeled off

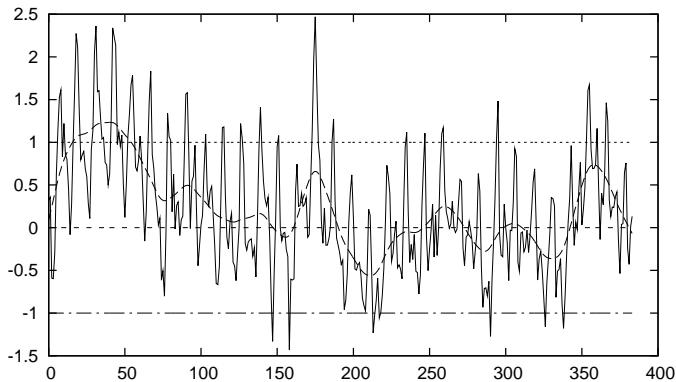


Figure 6.12: The final residual.

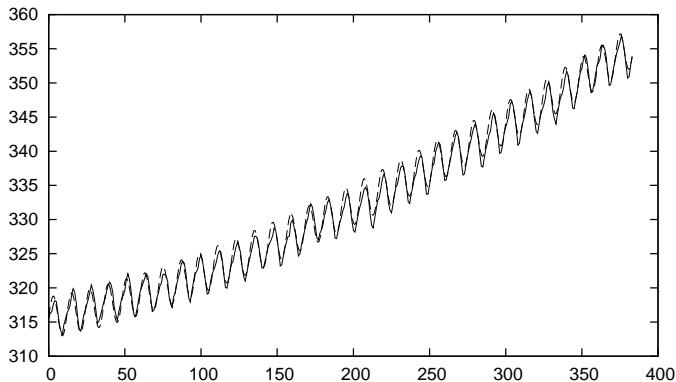


Figure 6.13: The raw data with the final fit.

the components that the data consists of, until only random noise remained. We ended up with an explicit, analytical formula, which describes the data remarkably well.

But there is something more. We did so entirely “manually”: by plotting the data, trying out some approximations, and wiggling the numbers until the agreed reasonably with the data. At no point did we resort to a “black-box” fitting routine: because we didn’t have to! We did just fine. (In fact, after everything was done I tried to perform a nonlinear fit using the functional form of the analytical model as we have worked it out — only to have it explode terribly! The model depends on 7 parameters, which means that convergence of a nonlinear fit can be a bit precarious. In fact, it took me *longer* to try to make the fit work, than it took me to work the parameters out manually, as I have

demonstrated.)

I'd go even further. We learned *more* by doing this work manually, than if we had used a fitting routine. Some of the observations (such as the idea to include higher harmonics) only arose through direct interaction with the data. And it's not even true that the parameters coming out of a fitting routine would be more accurate. Sure, they would contain 16 digits, but not more information: through the manual wiggling of the parameters, we can see really quickly and directly at what point changes to the parameters are so small that they no longer influence the agreement between the data and the model. That's when we have extracted all the information from the data — any further "precision" of the parameters is just insignificant noise.

You might want to try your hand at it yourself. For example, you may question the choice of the power law behavior for the long-term trend. Does an exponential function ($\sim \exp(x)$) give a better fit? It is not easy to tell from the data, but makes a huge difference if we want to project our findings significantly (ten years or more) into the future. Or you could take a closer look at the seasonality. Because it is so regular, and in particular because its period is known exactly, you should be able to isolate just the periodic part of the data in a separate model by averaging corresponding months for all years. And finally, there is much more data available than in the "classic" version of the data set that I used in my original exploration².

6.2 Workshop: gnuplot

While the example commands in this chapter should have given you an idea what working with gnuplot looks like, let's take a quick look at some of the basics.

Gnuplot is command-line oriented: when you start it, it presents you with a text prompt at which to enter commands; the resulting graphs are shown in a separate window. Creating plots is simple — the command:

```
plot sin(x) with lines, cos(x) with linespoints
```

will generate a plot of (you guessed it), a sine and a cosine. The sine will be drawn with plain lines, whereas the cosine will be drawn with symbols ("points"), connected by lines.

(Many gnuplot keywords can be abbreviated: instead of `with lines`, I would usually say: `w l`, or `w lp` instead of `with linespoints`. These short forms are a major convenience, but of course a bit cryptic in the beginning. In this little introductory section, I will make sure to only use the full forms of all commands.)

To plot data from a file, you also use the `plot` command, for instance like so:

²You can get it from the observatory's official web site at <http://www.esrl.noaa.gov/gmd/ccgg/trends/>. Also check out the narrative (with photos of the apparatus!) at <http://celebrating200years.noaa.gov/datasets/mauna/welcome.html>.

```
plot "data" using 1:2 with lines
```

When plotting data from a file, we use the `using` keyword to specify which columns from the file we want to plot — in the example above, we use the values in the first column as x values, and values from the second column for y .

One of the nice features of gnuplot is that you can apply arbitrary transformations to the data as it is being plotted. To do so, you enclose each entry in the column specification that you want to apply a transform to in parentheses. Within these parentheses, you can have any mathematical expression. The data from each column is available by prefixing the column index by the dollar sign. An example will make this more clear:

```
plot "data" using (1/$1):($2+$3) with lines
```

This command plots the sum of the second and third column (that is: $\$2+\3) as a function of one over the value in the first column ($1/\$1$).

It is also possible to mix data and functions in a single plot command (as we have seen in the examples in this chapter):

```
plot "data" using 1:2 with lines, cos(x) with lines
```

This is different from the Matlab-style of plotting, where a function has to be explicitly *evaluated* on a set of points, and only then the resulting set of values can be plotted.

We can now proceed to add decorations (such as labels and arrows) to the plot. Gnuplot also has all kinds of options to customize virtually every aspect of the plot's appearance: tic marks, legend, aspect ratio, you name it. When we are done with a plot, we can save all the commands used to create it, including all decorations, using the `save` command:

```
save "plot.gp"
```

Now we can use `load "plot.gp"` to recreate the graph.

As you can see, gnuplot is extremely straightforward to use. The one area that is often regarded as somewhat clumsy is the creation of graphs in common graphics file formats. The reason for this is historical: the first version of gnuplot was written in 1985, when one could not expect every computer to be connected to a graphics-capable terminal, and when many of our current file formats did not even exist yet! The gnuplot designers dealt with this situation by creating the so-called “terminal” abstraction. All hardware-specific capabilities were encapsulated by this abstraction, so that the rest of gnuplot could be as portable as possible. Over time, this “terminal” has started to include different graphics *file formats* as well (not just graphics hardware terminals), and this particular usage continues to this day.

To export a graph to a common file format (such as GIF, PNG, PostScript, or PDF), requires a five-step process:

```
set terminal png
set output "plot.png"
replot
set terminal wxt
set output
```

In the first step, we choose the output device or “terminal”: here, a PNG file. In the second step, we choose the file name. In the third step, we explicitly request that the graph be re-generated for this newly chosen device. The remaining commands restore the interactive session, by selecting the interactive wxt terminal (built on top of the wxWidgets widget set) and redirecting output back to the interactive terminal. If you find this process clumsy and error-prone, you are not alone; but rest assured: gnuplot allows you to write macros, which can reduce those five steps to one!

I should mention one further aspect of gnuplot: because it has been around for 25 years, it is by now very mature and robust when it comes to dealing with typical day-to-day problems. For example, gnuplot is refreshingly unpicky in regards to parsing input files. Many other data analysis or plotting programs I have seen are pretty rigid in this regard, and will bail when encountering unexpected data in an input file. In theory, that is the right thing to do, but in practice, data files are often not clean, with ad-hoc formats and missing or corrupted data points. Having your plotting program bail over whitespace (instead of tabs!), is a *major* nuisance when doing *real* work. In contrast, gnuplot usually does an amazingly good job at making sense of almost any input file you might throw at it, and that is indeed a great help! Similarly, gnuplot recognizes undefined mathematical expressions (such as $1/0$, $\log(0)$, and so on) and discards them. Again, this is a great help, because it means that you don’t have to worry about the domains over which functions are properly defined while you are in the thick of things. Since the output is graphical, there is usually very little risk that this silent discarding of undefined values will lead you to miss essential behavior. (This is different in a computer program, where silently ignoring error conditions usually only compounds the problem.)

6.3 Further Reading

- *Gnuplot in Action — Understanding Data with Graphs*. Philipp K. Janert. Manning Publications. 2010.

If you want to know more about gnuplot, then you might find this book interesting. It not only includes explanations of all sorts of advanced options, but also helpful hints for working with gnuplot.

Part II

Analytics: Modeling Data

Chapter 7

Guesstimation and the Back of the Envelope

Look around the room you are sitting in as you read this. Now answer the following question: How many Ping-Pong balls would it take to fill this room?

Yes, I know, it's lame to make the reader do jot'em-dot'em exercises, and the question is old anyway, but please make the effort to come up with a number. I am trying to make a point here.

Done? Good — then, tell me, what is the margin of error in your result? How many balls, plus or minus, do you think the room might accommodate as well? Again, numbers, please! Look at the margin of error: can you justify it or did you just pull some numbers out of thin air to get me off your back? And if you came up with your estimate using some argument: does the result seem right to you? Too large, too small?

Finally, can you state the assumptions you made when answering the first two questions? What did or did you not take into account? Did you take the furniture out or not? Did you look up the size of a Ping-Pong ball or did you guess it? Did you take into account different ways to pack spheres? Which of these assumptions has the largest effect on the result? Continue on a second sheet of paper if you need more space for your answer.

The game we just played is sometimes called *guesstimation*, and is a close relative to the *back-of-the-envelope calculation*. The difference is minor — the way I see it, in guesstimation, we worry more about finding suitable input values, whereas in a typical back-of-the-envelope calculation, the inputs are reasonably well known, and the challenge is to simplify the actual calculation to the point that it can be done on the back of the proverbial envelope. (Some people seem to prefer napkins to envelopes — that's the more sociable crowd.)

Let me be clear about this: I consider proficiency at guesstimation and similar techniques the absolute hallmark of the practical data analyst — the person who goes out and solves *real* problems in the *real* world. It is so powerful, because it connects a conceptual understanding (no matter how rough) with the

concrete reality of the problem domain — it leaves no place to hide. Guesstimation also generates *numbers* (not theories or models), with their wonderful ability to cut through vague generalities and opinion-based discussions.

For all these reasons, guesstimation is a crucial skill. It is where the rubber meets the road.

The whole point of guesstimation is to come up with an approximate answer — quickly and easily. The flip side of this is that it forces us to think about the accuracy of the result: first how to estimate the accuracy and then how to quote it. That will be the program for this chapter.

7.1 Principles of Guesstimation

Let's step through the Ping-Pong ball example from this chapter's introduction together. This will give me an opportunity to point out a few techniques that are generally useful.

First: the room. Basically rectangular in shape. I have bookshelves along several walls and that helps me estimate the length of each wall, since I know that shelves are 90cm (3ft) wide — that's a pretty universal standard. I also know that I am 1.80m (6ft) tall, which helps me estimate the height of the room. All told, this comes to 5m by 3.5m by 2.5m or about 50m³.

Now, the Ping-Pong ball. I haven't had one in my hands for a long time, but I seem to remember that they are about 2.5cm (1in) in diameter. That means I can line up 40 of them in a meter, which means I have 40³ in a cubic meter. The way I calculate this is: $40^3 = 4^3 \cdot 10^3 = 2^6 \cdot 1000 = 64,000$. That's how many Ping-Pong balls fit into a cubic meter.

Taking things together, I can fit $50 \cdot 64000$ or approximately 3,000,000 Ping-Pong balls into this room. That's a large number. If each ball costs me a dollar at a sporting goods store, then the value of all the balls required to fill this room would be many times higher than the value of the entire house!

Next, the margins of error. The uncertainty in each dimension is at least ten percent. Relative errors add to each other in a multiplication (we will discuss error propagation later in this chapter), so that the total error turns out to be $3 \cdot 10$ percent = 30 percent! That's pretty large — the number of balls required might be as low as two million or as high as four million. It is uncomfortable to see how the rather harmless-looking 10 percent error in each individual dimension has compounded to lead to a 30 percent uncertainty.

The same problem applies to the diameter of the Ping-Pong balls. Maybe 2.5cm are a little low — maybe 3cm is more like it. Now, that's a 20 percent increase, meaning that the number of balls that fit into a one cubic meter cube goes down by 60 percent (three times the relative error, again): now we can fit only about 30,000 of them into a cubic meter. The same goes for the overall estimate: a drop by half if balls are 5mm larger than initially assumed. Now the range is something between one and two millions.

Finally, the assumptions. Yes, I took the furniture out. Given the uncertainty in the total volume of the room, the space taken up by the furniture does

not matter much. I also assumed that balls would stack like cubes, when in reality they pack much tighter if we arrange them in the way oranges (or cannonballs) are stacked. It's a slightly non-trivial exercise in geometry to work out the factor, but it comes to about 15 percent more balls in the same space.

So, what can we now say with certainty? We will need a few millions of Ping-Pong balls — probably not less than one million and certainly not more than five millions. The biggest uncertainty is the size of the balls themselves, but if we need a more accurate estimate then the one we have obtained so far, we can look up their exact dimensions and adjust the result accordingly.

(After I wrote this paragraph, I finally looked up the size of a regulation Ping-Pong ball: 38-40mm. Oops. That means only about 15,000 balls fit into a cubic meter and I have to adjust all my estimates down by a factor of four.)

This example demonstrates all important aspects of guesstimation:

- Estimate sizes of things by comparing them to something you know.
- Establish functional relationships using simplifying assumptions.
- Originally innocuous errors can compound dramatically, hence tracking the accuracy of an estimate is crucial.
- And finally, a few bad guesses on things that are not very familiar can have a devastating effect (I really haven't played Ping-Pong in a long time), but can be corrected easily when better input is available.

Yet on the other hand, we did find the order of magnitude, one way or the other: a few million.

7.1.1 Estimating Sizes

The best way to estimate the size of an object is to compare it to something you know. The shelves played this kind of role in the previous example, although sometimes you have to work a little harder to find a familiar object to use as reference in any given situation.

Obviously, this is easier to do the more you know, and it can be very frustrating to find yourself in a situation where you don't know anything to use as reference. On the other hand, it is usually possible to go quite far with just a few data points to use as reference values.

(There are stories from the middle ages, how soldiers used to count how many rows of stone blocks were used in the walls of a fortress before mounting an attack, to estimate the height of the walls. Obtaining an accurate value was important, to prepare scaling ladders of the appropriate length: if the ladders were too short, the top of the wall could not be reached, but if they were too long, the defenders could get a hold of the overhanging tops and keel the ladders over. Bottom line: you got to find your reference objects where you can.)

Knowing the sizes of things is therefore the first order of business — not only because the more you know, the easier it is to form an estimate, but also because the more you know, the more you develop a feeling for the correct answer. That is an important step when operating with guesstimates: to do an independent sanity check at the end to make sure we did not make some horrible mistake along the way. (In fact, the general advice is that “two (independent) estimates are better than one”, which is certainly true, but not always possible — at least I can’t think of an independent way to do the Ping-Pong ball example that we started with.)

Knowing the sizes of things can be *learned*. All it takes is a healthy interest in the world around you — please don’t go through the dictionary, memorizing data points in the order of the alphabet. This is not about beating your buddies at a game of Trivial Pursuit! Instead, this is about becoming familiar (I’d almost say: intimate) with the world you live in. Feynman once wrote about Hans A. Bethe that “every number was near something he knew”. That is the ideal.

The next step is to *look things up*, and in situations where one frequently needs relatively good approximations to problems coming from a comparably small problem domain, special-purpose lookup tables can be a great help. I vividly remember a situation in a senior Physics lab, where we were working on an experiment (I believe, to measure the muon lifetime), when the instructor came by and asked us some guesstimation problem — I forget what it was, but it was non-trivial. None of us had a clue, and so he whipped out from his back-pocket a little booklet the size of a playing card, with the physical properties of all kinds of sub-nuclear particles: for almost any situation that could arise in the lab, he had an approximate answer right there.

Specialized lookup tables exist in all kinds of disciplines, and you might want to make your own as necessary for whatever it is you are working on. The funniest I have seen gave typical sizes (and costs) for all elements of a manufacturing plant or warehouse: so many square-feet for the office of the general manager, so many square-feet for his assistant (half the size of the boss’s), down to the number of square-feet per toilet stall, and — not to forget — how many toilets to budget for every 20 workers per 8-hour shift.

Finally, if we don’t know anything close, and we aren’t able to look anything up, we can try to estimate from the ground up, so to speak: starting just with what we know, and then piling up arguments to come up with an estimate. The problem with this approach is that the result may be *way off*: as we have seen earlier, errors compound, and the more steps we have in our line of arguments, the larger the final error is going to be, possibly becoming so large that the result is going to be useless. If that’s the case, we can still try and find a cleverer argument that makes do with fewer arguments steps. But I also have to acknowledge that occasionally we will find ourselves simply stuck: not able to make an adequate estimate with the information we have.

The art is to make sure this happens only rarely.

7.1.2 Establishing Relationships

Establishing relationships that get us from what we know to what we want to find is usually not that hard, in particular in common business scenarios, where the questions often revolve around rather simple relationships (how something fits into the other, how many items of a kind there are, and the like). In scientific applications, this kind of argument can be harder. But for most situations we are likely to encounter outside the science lab, simple geometric and counting arguments will suffice.

In the next chapter (chapter 8), I will talk more about the kinds of arguments you can use to establish relationships. For now, just one recommendation: *make it simple!* Not: *keep it simple*, because more likely than not, initially the problem is *not simple*, so you have to make it so, in order to make it tractable.

Simplifying assumptions let you cut through the fog and get to the essentials of a situation. You may incur an error as you simplify the problem, and you want to estimate its effect, but at least you are moving towards a result.

An anecdote may illustrate what I mean: When I was working for Amazon.com, I once asked a rather sophisticated mathematician how many packages Amazon can typically fit onto a tractor-trailer truck, and he started to work out the different ways you can *stack* rectangular boxes into the back of the truck! This is entirely missing the point, since for a rough calculation, we can make the simplifying assumption that the packages can take on any shape at all (that is, they behave like a liquid) and simply divide the total volume of the truck by the typical volume of a package. Since the individual package is tiny compared to the size of the truck, the specific shapes and arrangements of individual packages are entirely irrelevant and their effect is much smaller than the errors in our estimates for the size of the truck, for instance. (We'll discuss in more detail in chapter 8, when we talk about the mean-field approximation.)

The point of back-of-the-envelope estimates is to retain only the core of the problem, stripping away as much non-essential detail as possible. Be careful that your sophistication does not get in the way of finding simple answers.

7.1.3 Working With Numbers

When working with numbers, don't automatically reach for a calculator! I know that I am now running the risk of sounding ridiculous, praising the virtues of old-fashioned reading, 'riting, and 'rithmetic. But that's not my point. My point is that it is *alright* to work with numbers — there is no reason to avoid them.

I have seen the following scenario happen countless times: a discussion is going on, everybody is involved, ideas are flying, concentration is intense, when all of a sudden we need a few numbers to proceed. Immediately, *everything* comes to a screeching halt, while several people grope for their calculators, others fire up their computers, followed by hasty attempts to get the required answer, which invariably (given the haste) leads to numerous keying

errors and false starts, followed by arguments about the best calculator software to use. In any case, the whole creative process just died. It is a shame.

Besides the need for context switching, calculators remove you one step further from the nature of the problem. When you do a problem in your head, you get a feeling for the significant digits in the result: for which digits does the result change as the inputs take on any value from their permissible range? The surest sign that somebody has no clue is when they quote the results from a calculation based on order-of-magnitude inputs to sixteen digits!

The whole point here is not to be religious about it — either way. If it actually becomes more complicated to work out a numerical approximation in one's head, by all means let's get a calculator. But the compulsive habit to avoid working with numbers at all cost is not helpful, either.

There are a few good techniques, which help with the kinds of calculations required for back-of-the-envelope estimates, and which are simple enough that they still (even today) hold their own against uncritical calculator use. Only the first is a must-have, the other two are optional.

Powers of Ten

The most important technique for deriving order-of-magnitude estimates is to work with orders of magnitudes directly: that is, with powers of ten.

It quickly gets confusing to multiply 9,000 by 17 and then to divide by 400, and so on. Instead of trying to work with the numbers directly, split each number into the most significant digit (or digits) and the respective power of ten. The multiplications now take place among the digits only, while the powers of ten are summed up separately. In the example I just gave, we split $9000 = 9 \cdot 1000$, $17 = 1.7 \cdot 10 \approx 2 \cdot 10$, and $400 = 4 \cdot 100$. From the leading digits, we therefore have 9 times 2 divided by 4 equal to 4.5; and from the powers of ten we have 3 plus 1 minus 2 equal to 2 — in other words $4.5 \cdot 10^2 = 450$. That wasn't so hard, was it? (I have replaced 17 with $2 \cdot 10$ in this approximation, so the result is a bit on the high side, by about 15%, and I might want to correct for that in the end — a better approximation would be closer to 390. The exact value is 382.5.)

More systematically, any number can be split into a decimal fraction and a power of ten. It will be most convenient to require the fraction to have exactly one digit before the decimal point, like so:

$$123.45 = 1.2345 \cdot 10^2$$

$$1,000,000 = 1.0 \cdot 10^6$$

$$0.00321 = 3.21 \cdot 10^{-3}$$

The fraction is commonly known as the *mantissa* (or the *significand* in the most recent usage), whereas the power of ten is always referred to as the *exponent*.

This notation significantly simplifies multiplications and divisions between numbers of very different magnitude: the mantissas multiply (involving only *single digit* multiplications, if we restrict ourselves to the most significant digit) and the exponents add. The biggest challenge is to keep the two different tallies simultaneously in one's head.

Small Perturbations

The techniques in this section are part of a much larger family of methods known as *perturbation theory*, which play a huge role in applied mathematics and related fields. The idea is always the same — we split the original problem into two parts: one that is easy to solve, and one that is somehow “small” compared to the first. If we do it right, the effect of the latter part is only a “small perturbation” to the first, easy part of the problem. (You may want to review the appendix B if some of this material is unfamiliar to you.)

The easiest application of this idea is in the calculation of simple powers, such as (for instance) 12^3 . Here is how we would proceed:

$$\begin{aligned} 12^3 &= (10 + 2)^3 = 10^3 + 3 \cdot 10^2 \cdot 2 + 3 \cdot 10 \cdot 2^2 + 2^3 \\ &= 1000 + 600 + \dots \\ &= 1600 + \dots \end{aligned}$$

In the first step, we split 12 into $10 + 2$ — here 10 is the easy part (because we know how to raise 10 to an integer power) and 2 is the perturbation (because $2 \ll 10$). In the next step, we make use of the binomial formula (see appendix B), ignoring anything except for the linear term in the “perturbation”. The final result is pretty close to the exact value.

The same principle can be applied to many other situations. In the context of this chapter, I am interested in it because it gives us a way to estimate and correct for the error introduced by ignoring anything except the first digit in the powers-of-ten calculations. Let's look at an example, again:

$$32 \cdot 430$$

Using only the most significant digits, this is $(3 \cdot 10^1) \cdot (4 \cdot 10^2) = (3 \cdot 4) \cdot 10^{1+2} = 12000$. But this is clearly not correct, because we dropped some digits from the factors.

We can consider the non-leading digits as *small perturbations* to the result and treat them separately. In other words, the calculation becomes $(3 + 0.2) \cdot (4 + 0.3) \approx 3(1 + 0.1\dots) \cdot 4(1 + 0.1\dots)$, where I have *factored out* the largest factor in each term. On the right hand side, I did not write out the correction terms in full — it's enough to know for our purposes that they are about 0.1.

Now we can make use of the binomial formula:

$$(1 + \epsilon)^2 = 1 + 2\epsilon + \epsilon^2$$

We drop the last term (since it will be very small compared to the other two), but the second term gives us the size of the correction: $+2\epsilon$, which in our case is about 20 percent, since ϵ is one-tenth.

I will admit that this technique seems somewhat out of place today, although I do use it for real calculations when I don't have a calculator on me. But the true value of this method is that it gives me a way to estimate and reason about the effect that changes to my input variables are going to have on the overall outcome. In other words, this is method is a first step towards *sensitivity analysis*.

Logarithms

This is the method by which generations before us did numerical calculations. The crucial insight is that we can use logarithms for products (and exponentiation) by making use of the functional equation for logarithms:

$$\log(xy) = \log(x) + \log(y)$$

In other words, instead of *multiplying* two numbers, we can *add* their logarithms. The slide-rule was a mechanical calculator based on this idea.

Amazingly, using logarithms for multiplication is *still* relevant, but in a slightly different context. For many statistical applications (in particular when using Bayesian methods), we need to multiply the probabilities of individual events to arrive at the probability for the combination of these events. Since probabilities are by construction $0 \leq p \leq 1$, the product of any two probabilities is always smaller than the individual factors. It does not take many probability factors to underflow the floating-point precision on almost any standard computer. Logarithms to the rescue! Take logarithms of the individual probabilities and add the logarithms rather than multiply the probabilities. (The logarithm of a number that is less than one is negative, therefore one usually works with $-\log(p)$.) The resulting numbers, although mathematically entirely equivalent, have much better numerical properties. Finally, since in many applications we mostly care which of a selection of different events has the maximum probability, we don't even need to convert back to probabilities: the event with maximum probability will also be the one with the maximum (negative) logarithm.

7.1.4 More Examples

We have all seen this scene in many a Hollywood movie: the gangster comes in to pay off the hitman (or pay for the drug deal, or whatever it is). Invariably, he hands over an elegant briefcase with the money — cash obviously. Question: how much was in the case?

Well, a briefcase is usually sized to hold two letter-size papers next to each other so it is about 17-by-11 inches wide, and maybe 3 inches tall (or 40-by-30-by-7 centimeters). A bank note is about 4 inches wide and 2 tall, which means

that we can fit about 10 per sheet of paper. And a 500-page ream of printer paper is about 2 inches thick. All told, we end up with $2 \cdot 10 \cdot 750 = 15,000$ banknotes. The highest dollar denomination in circulation is the one-hundred dollar bill (larger denominations exist, but — although legal tender — are not officially in circulation, and apparently fetch far more than their face value among collectors), hence the maximum value of that pay-off was about 1.5 million dollars, and certainly not more than 5 million dollars.

Conclusion: for the really big jobs, you need to pay by check. Or use direct transfer.

For a completely different example, consider the following question: what's the typical take-off weight of a large, intercontinental jet airplane? It turns out that you can come up with an approximate answer even if you don't know *anything* about planes.

A plane is basically an aluminum tube with wings. Ignore the wings for now; let's concentrate on the tube. How big is it? One way to find out is to check your boarding pass: it will display your row number. Unless you are much more classy than your author, chances are that it shows a row number in the range of 40–50. You can estimate that the distance between seats is a bit over 50 cm — although it feels closer. (When you stand in the aisle, facing sideways, you can place both hands comfortably on the tops of two consecutive seats. Your shoulders are about 30 cm apart, hence the distance between seats must be a tad larger than that.) So, there we have the length: $50 \cdot 0.5\text{m}$. We double this, to make up for first and business class, and because seats are *further* apart than 50 cm. Therefore, the length of the tube is about 50m. How about its diameter? Back in economy, rows are about 9 seats abreast, plus two aisles. Each seat being just a bit wider than your shoulders (hopefully), we end up with a diameter of about 5m. So, we are dealing with a tube, 50m long and 5m in diameter.

As you walked through the door, you could have noticed the strength or thickness of the tube: it's about 5mm. Let's make that 10mm or 1cm to account for "stuff", such as wiring, seats, all kinds of other hardware that's in the plane. So, if you imagine that you unroll the entire plane (the way you unroll aluminum foil), you end up with a sheet that is $50 \cdot \pi \cdot 5 \cdot 0.01\text{m}^3$. The density of aluminum is a little higher than water (if you have ever been to a country that uses aluminum coins, you know that you can make them float), so let's say it's 3g/cm^3 .

It is at this point that we need to employ the proverbial back of the envelope (or the cocktail napkin they gave you with your peanuts), to work out the numbers. It will help to realize that there are $100^3 = 10^6$ cubic centimeters in a cubic meter, and that therefore the density of aluminum can be written as 3 tons per cubic meter. The final mass of the "tube" comes out to about 25 tonnes. Let's double this to take into account the wings (wings are about as long as the fuselage is wide — if you look at the silhouette of a plane in the sky it forms an approximate square), hence we have 50 tonnes just for the "shell" of the airplane. This does not take into account the engines and most of the other equipment inside the plane.

	Length	Width	Diameter	Weight (Empty)	Weight (Full)	Passengers
B767	50m	50m	5m	90t	150t	200
B747	70m	60m	6.5m	175t	350t	400
A380	75m	80m	7m	275t	550t	500

Table 7.1: Approximate measurements for some common intercontinental jets.

Let's compare this number with the load — we have 50 rows, half of them with nine passengers, and for the other half, we will assume five, for an average of seven passengers per row, or a total of 350 passengers per plane. Assuming that each passenger contributes 100kg (body weight and baggage), the load amounts to 35 tonnes: comparable with the weight of the plane itself. (The weight-to-load ratio is actually not that different than for a car, fully occupied with four people. Of course, if you are driving alone, the car is *much* worse.)

How good are we doing? Actually not bad at all: table 7.1 gives typical values for three planes that are common on the transatlantic routes: the mid-size Boeing 767, the large Boeing 747 (the "Jumbo"), and the extra-large Airbus 380. That's enough to check our calculations. We are doing pretty good.

(What we totally *missed* is that planes don't fly on air and inflight peanuts alone: in fact, the greatest single contribution to the weight of a fully loaded and fuelled airplane comes from the weight of the *fuel*. You can estimate that as well, but to do so, you will need one additional bit of information: namely that the fuel consumption of a modern jet airplane per passenger and mile travelled is better than for a normal economy class car with only a single passenger.)

That was a long and involved estimation, and I won't blame you if you skipped some of the intermediate steps. In case you are just joining us again, I'd like to emphasize just one point: notice how we came up with a reasonable estimate without having to resort to any "bottom of the pants" estimates — despite the fact that we had no prior knowledge! Everything that we used we could either observe directly (such as the number of rows in the plane or the thickness of the fuselage walls) or could relate to something that was familiar to us (such as a distance between seats). That's an important take-away!

But not all calculations have to be complicated. Sometimes, all you have to do is put "two-and-two" together. A friend told me recently that his company had to cut their budget by one million dollars. We knew that the overall budget for this company was about five million dollars annually. I also knew that, since it was mostly a service company, basically all its budget was spent on payroll (there was no inventory or rent to speak of). I could therefore tell my friend that lay-offs were around the corner — even with a salary reduction program, the company would have to cut *at least* 15 percent of their staff. The response was: "Oh, no, our management would *never* do that." Two weeks later, the company eliminated one third of all positions...

7.1.5 Things I Know

Table 7.2 is a collection of things that I know and which I frequently use to make estimates. Of course, this list appears a bit whimsical, but it is actually pretty serious. For instance, note the *range* of areas from which these items are drawn! What domains can you reason about, given the information in this table?

Also notice the absence of systematic “scales”. That is no accident. I don’t need to memorize the weights of a mouse, a cat, and a horse — because I know (or can guess) that a mouse is 1000 times smaller than a human, a cat 10 times smaller, and a horse 10 times larger. The items in this table are *not* intended to be comprehensive. They are the bare minimum. Knowing how things relate to each other lets me take it from there.

Of course this table reflects my personal history and interests. Yours will look different.

7.2 How good are those numbers?

Remember the Ping-Pong ball question that started out this chapter? I once posted that question on a homework set in a class I was teaching, and one of the answers I got back was: 1,020,408.16327 or something of that sort. (Did you catch *both* mistakes? Not only does the result of a rough estimate pretend to be accurate to within a single ball; but the answer also includes a fractional part — which is meaningless, given the context.) This kind of oversight or confusion is incredibly common: we focus so much on the calculation (any calculation) that we forget to interpret the result.

This story serves as a reminder that there are two questions that we should ask *before* any calculation, and one to ask *afterwards*. The two questions to ask before are:

- What level of correctness do I *need*?
- What level of correctness can I *afford*?

and the question to ask afterwards is:

- What level of correctness did I *achieve*?

I use the term “correctness” here a bit loosely to refer to the quality of the result. There are actually two different concepts involved: *accuracy* and *precision*.

Accuracy Accuracy expresses how close the result of a calculation or measurement comes to the “true” value. Low accuracy is due to systematic error.

Precision Precision refers to the “margin of error” in the calculation or the experiment. In experimental situations, precision tells us how far the results will stray when the experiment is repeated several times. Low precision is due to random noise.

Said another way: accuracy is a measure for the correctness of the result, and precision is a measure of the results uncertainty.

7.2.1 Before You Get Started: Feasibility and Cost

The first question (what level of correctness is needed) will define the overall approach — if I need an order of magnitude approximation, the proverbial back of the envelope will do; if I need better results, I might need to work harder. The second question is the necessary corollary, by asking whether I will be able to achieve my goal given the available resources. In other words, these two questions pose a classic engineering trade-off (cost/benefit analysis).

This obviously does not matter much for a throw-away calculation, but it matters a lot for bigger projects. I once witnessed a huge (multi-man-year) project to build a computation engine, which had failed to come clear on both points. The project was eventually canceled when it turned out that it would cost *more* to achieve the accuracy required than the project was supposed to gain the company in increased revenue! (Don't laugh — it could happen to you. Or at least: in your company.)

This story points to an important fact which is quite generally true: correctness is usually expensive, and high correctness is often *disproportionally* more expensive. In other words, a 20 percent approximation can be done on the back of an envelope, a 5 percent solution can be done in a couple of months, but the cost for a 1 percent solution may be astronomical. It is also not uncommon that there is no middle ground (such as an affordable 10 percent solution).

I have also seen the opposite problem: projects chasing correctness that is not really needed, or not achievable because the required input data is not available or of poor quality itself. This is a particular danger if there is the opportunity to play with some attractive new technology.

Finding out the true cost or benefit of higher quality results can often be tricky. I was once working on a project to forecast the number of visitors viewing the company's website every day, and I was told that "we must have absolute forecast accuracy, nothing else matters". I suggested that if this was so, then we should take the entire site *down*, since this would guarantee a perfect forecast (zero page views). Because it would also imply zero revenue from display advertising, this suggestion focused the minds wonderfully to define more clearly what "else" mattered.

7.2.2 After You Finish: Quoting and Displaying Numbers

It is obviously pointless to quote results to more digits than is warranted. In fact, it is not only pointless, but misleading — at the very least, it is unhelpful, because it fails to communicate to the reader another important aspect of the result, namely its reliability!

A good rule (sometimes known as *Ehrenberg's Rule*) is to quote all digits up to and including the *first two variable digits*. Starting from the left, you keep all digits that do not change over the entire range of numbers from one data point

to the next, and then you keep the first two digits which vary over the *entire range* from 0 to 9 as you scan over all data points. An example will make this clear. Consider the following data set:

```
121.733
122.129
121.492
119.782
120.890
123.129
```

Here, the first digit (from the left) is always 1, the second one takes on only two values (1 and 2), so we retain them both. All further digits can take on any value between 0 and 9, and we retain the first two of them — meaning that we retain a total of *four* digits from the left. The two right-most digits therefore carry no significance, and we can drop them when quoting results. The mean (for instance) should be reported as:

121.5

Displaying further digits adds no value.

This rule (to retain the first two digits which vary over the entire range of values, and all digits to the left of them) works well with the methods described in this chapter: if you are working with numbers as I suggested earlier, you also develop a sense for the digits that are largely unaffected by reasonable variations in the input parameters, and the position in the result after which the uncertainties in the input parameters corrupt the outcome.

Finally, a word of warning: the accuracy level of a numerical result should be established from the outset; doing so later will trigger resistance. I once worked on a system that reported projected sales numbers (which were typically in the hundreds of thousands) to 6 “significant” digits (like “324,592” or so). Since these were forecasts, which were *at best* accurate to within 30 percent, *all* digits beyond the first were absolute junk! (Thirty percent of 300,000 is 100,000, meaning that the confidence band for this result was 200,000–400,000.) But a later release of the same software, which tried to report only the actually significant digits, was met by violent opposition from the user community, because it was “so much less precise”!

7.3 A Closer Look at Perturbation Theory and Error Propagation

I have mentioned the notion of “small perturbations” earlier. It is one of the great ideas of applied mathematics, so it is worth a closer look.

Whenever we can split a problem into an “easy” part and a part that is “small”, the problem lends itself to a perturbative solution. The “easy” part we can solve directly

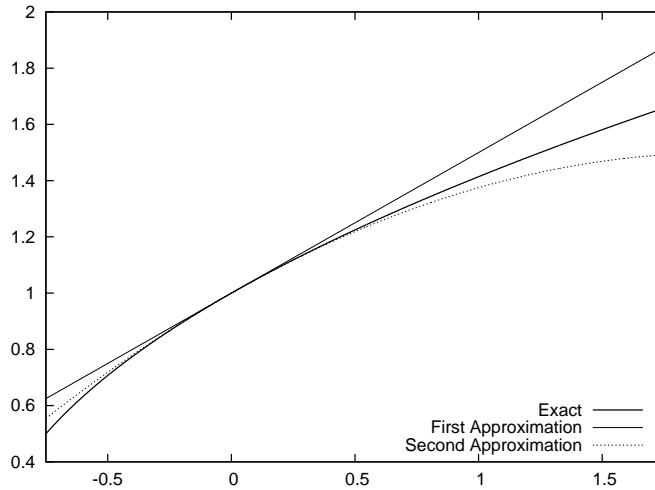


Figure 7.1: The square root function $\sqrt{1+x}$ and the first two approximations around $x = 0$.

(that's what we mean by "easy"), and the part that is "small" we solve in an approximative fashion. By far the most common source of approximations in this area is based on the observation that every function (every curve) is linear (a straight line) in a sufficiently small neighborhood: we can therefore replace the full problem by its linear approximation when dealing with the "small" part. And linear problems are always solvable.

Let me demonstrate how this works using two simple examples. In the first, we calculate a number, but in the second, we actually solve an equation.

As a first example, let's calculate $\sqrt{17}$. Can we split this into a "simple" and a "small" problem? Well, we know that $16 = 4^2$, hence $\sqrt{16} = 4$. That's the simple part, and we therefore now write $\sqrt{17} = \sqrt{16+1}$. Obviously $1 \ll 16$, so there's the "small" part of the problem. We can now rewrite our problem as follows:

$$\begin{aligned}\sqrt{17} &= \sqrt{16+1} \\ &= \sqrt{16(1+\epsilon)} \\ &= \sqrt{16}\sqrt{1+\epsilon} \\ &= 4\sqrt{1+\epsilon}\end{aligned}$$

It is often convenient to factor out everything, so that we are left with $1 + \text{small stuff}$, as we have done in the second line. At this point, we also replaced the small part with ϵ — we will put the numeric value back in at the end.

Up to this point, everything has been exact. To make progress, we need to make an approximation. In this case, we replace the square root by a local approximation around 1. (Remember: ϵ is small, and $\sqrt{1}$ is easy.) Every smooth function can be replaced by a

straight line locally, and if we don't go too far, that approximation turns out to be quite good (see figure 7.1). These approximations can be derived in a systematic fashion by a process known as *Taylor expansion*. The figure shows both the simplest approximation, which is just a straight line, and also the next higher (second order) approximation, which is even better.

Taylor expansions are so fundamental that they are almost considered a *fifth* basic operation (after addition, subtraction, multiplication, and division). See appendix B for a little more information on them.

With the linear approximation in place, our problem now become quite tractable:

$$\begin{aligned}\sqrt{17} &\approx 4\left(1 + \frac{\epsilon}{2} + \dots\right) \\ &= 4 + 2\epsilon\end{aligned}$$

We can now plug the numeric value $\epsilon = 1/16$ back in: $\sqrt{17} \approx 4 + 2/16 = 4.125$. The exact value is $\sqrt{17} = 4.12310\dots$. We are doing good.

Coming soon:
Second Example

7.3.1 Error Propagation

Error propagation considers situations where we have some quantity x and an associated uncertainty δx — we write $x \pm \delta x$ to indicate that we expect the true value to lie anywhere in the range from $x - \delta x$ to $x + \delta x$. In other words, we don't just have a single value for the quantity x , but instead a whole range of possible values.

Now imagine we have several quantities, each with its own error term, and we need to combine them in some fashion. We probably know how to work with the quantities themselves — but what about the uncertainties? For example, we know both the height and width of a rectangle to within some range: $h + \delta h$ and $w + \delta w$. We also know that the area is $A = hw$ (from basic geometry). But what can we say about the uncertainty in the area?

This kind of scenario is ideal for the kind of “perturbative” methods discussed earlier: the uncertainties are “small”, and we can therefore use simplifying approximations to deduce their behavior.

Let's work through the area example:

$$\begin{aligned}A &= (h \pm \delta h)(w \pm \delta w) \\ &= hw \left(1 \pm \frac{\delta h}{h}\right) \left(1 \pm \frac{\delta w}{w}\right) \\ &= hw \left(1 \pm \frac{\delta h}{h} \pm \frac{\delta w}{w} + \frac{\delta h}{h} \frac{\delta w}{w}\right)\end{aligned}$$

Here again, we have factored the primary terms out, to end up with terms of the form $1 + \text{small stuff}$, because that makes life easier. This also means that instead of expressing the uncertainty through the *absolute error* δh or δw , we instead express them through the *relative error* $\delta h/h$ or $\delta w/w$. (Note that if $\delta h \ll h$, then $\delta h/h \ll 1$.)

So far, everything has been exact. Now comes the approximation: the error terms are small (in fact: smaller than 1), hence their product is extra-small, and we can drop it.

Our final result is therefore $A = hw \left(1 \pm \left(\frac{\delta h}{h} + \frac{\delta w}{w}\right)\right)$, or in words: “When multiplying two quantities, their relative errors add.” So, if I know both the width and the height to within 10% each, then my uncertainty in the area will be 20%.

Here are a few more results of this form, which are useful whenever you work with quantities that have associated uncertainties (you might want to try and derive some of these):

$$\begin{aligned} (x \pm \delta x) + (y \pm \delta y) &= x + y \pm (\delta x + \delta y) && \text{Sum} \\ (x \pm \delta x) \cdot (y \pm \delta y) &= xy \left(1 \pm \left(\frac{\delta x}{x} + \frac{\delta y}{y}\right)\right) && \text{Product} \\ \frac{x \pm \delta x}{y \pm \delta y} &= \frac{x}{y} \left(1 \pm \left(\frac{\delta x}{x} + \frac{\delta y}{y}\right)\right) && \text{Fraction} \\ \sqrt{x + \delta x} &= \sqrt{x} \sqrt{1 + \frac{\delta x}{x}} \approx \sqrt{x} \left(1 + \frac{1}{2} \frac{\delta x}{x}\right) && \text{Square Root} \\ \log(x + \delta x) &= \log \left(x \left(1 + \frac{\delta x}{x}\right)\right) \approx \log x + \frac{\delta x}{x} && \text{Logarithm} \end{aligned}$$

The most important ones are the first two: when adding (or subtracting) two quantities, their absolute errors add, and when multiplying (or dividing) two quantities, their relative errors add. This implies that if of two quantities one has a significantly larger error than the other, the larger error dominates the final uncertainty.

Finally, you may have seen a different way to calculate errors, which gives slightly tighter bounds, but it is only appropriate if the errors have been determined by calculating the variances in *repeated measurements* of the same quantity: only in that case are the statistical assumptions valid on which this alternative calculation is based. For guesstimation, the simple, but more pessimistic approach described here is more appropriate.

7.4 Workshop: The Gnu Scientific Library (GSL)

What do you do when a calculation becomes too involved that you can do it in your head or (even) on the back of an envelope? In particular, what to do if you *need* the extra precision that a simple order-of-magnitude estimation as we have practiced them in this chapter will not provide? Absolutely: you reach for a numerical library!

The Gnu Scientific Library (GSL) is the best currently available open-source library for numerical and scientific calculations that I am aware of. The list of included feature is comprehensive, and the implementations are of high quality. Thanks to some unifying conventions, the API, though at first sight forbidding, is actually quite easy to learn and comfortable to use. Most of all, the library is mature, well-documented, and reliable.

Let’s use it to solve two rather different problems, which will give us an opportunity to highlight some of the design choices incorporated into the GSL. The first example will involve matrix and vector handling: we will calculate the singular value decomposition (SVD) of a matrix. The second example demon-

strates how the GSL handles non-linear, iterative problems in numerical analysis: we will find the minimum of a non-linear function.

The listing 7.1 should give you a flavor what vector and matrix operations look like when using the GSL. First, we allocate a couple of (two-dimensional) vectors and assign values to their elements. We then perform some basic vector operations: adding one vector to another, and performing a dot product. (The result of a dot product is a scalar, not another vector.) Finally, we allocate and initialize a matrix and calculate its singular value decomposition (SVD). (See chapter 14 for a bit more information regarding vector and matrix operations.)

It is becoming immediately (and a little painfully) clear that we are dealing with plain C, not C++ or any other more modern, object-oriented language! There is no operator-overloading; we must use regular functions to access individual vector and matrix elements. There are no namespaces, so function names tend to be lengthy. And of course there is no garbage collection!

What is *not* so obvious is that element access is actually boundary checked: if you try to access a vector element that does not exists, for instance like so: `gsl_vector_set(a, 4, 1.0);`, the GSL internal error handler will be invoked. By default, it will halt the program and print a message to the screen. This is quite generally true: if the library detects an error, including bad inputs, failure to converge numerically, or an out-of-memory situation, it will invoke its error handler to notify you. You can provide your own error handler to respond to errors in a more flexible fashion. For a fully tested program, you can also turn range checking on vector and matrix elements *off* completely, to achieve the best possible runtime performance.

Two more implementation details before leaving the linear algebra example: although the matrix and vector elements are of type `double` in this example, versions of all routines exist for integer and complex data types as well. Furthermore, the GSL will use an optimized implementation of the BLAS (Basic Linear Algebra Subprograms) API if one is available; if not, it comes with its own, basic implementation.

Now let's take a look at the second example, listing 7.2. Here, we use the GSL to find the minimum of a one-dimensional function. The function to minimize is defined at the top of the listing: $x^2 \log(x)$. In general, nonlinear problems such as this must be solved iteratively: we start with a guess, then calculate a new trial solution based on that guess, and so on, until the result meets whatever acceptance criteria we care to define.

At least that's what introductory text books tell you...

In the main part of the program, we instantiate a “minimizer”, which is an encapsulation of a specific minimization algorithm (Golden Section Search, in this case — others are available, too), and initialize it with the function to minimize, as well as our initial guess for the interval containing the minimum.

Now comes the surprising part: an explicit loop! In this loop, the “minimizer” takes a single step in the iteration (that is: calculates a new, tighter interval bounding the minimum), but then essentially hands control back to us. Why so complicated? Why can't we just specify the desired accuracy of

the interval and let the library handle the entire iteration for us? Because real problems more often than not don't converge as obediently as the books tell us! Instead they can (and do) fail in a variety of ways: they converge to the wrong solution, they attempt to access values for which the function is not defined, they attempt to make steps which (for reasons of the larger system of which the routine is only a small part) are either too large or too small, or they diverge entirely. Based on my experience, I have come to the conclusion that *every nonlinear problem is different* (whereas every linear problem is the same), and therefore generic black-box routines don't work!

Which brings us back to the way this minimization routine is implemented: rather than as a black-box, the entire iteration is open and accessible to us. We can merely monitor its progress (as we do in this example, by printing every iteration step to the screen), but we could also interfere with it, for instance to enforce some invariant that is specific to our problem. The "minimizer" does as much as it can, by calculating and proposing a new interval, but ultimately, we are in control over the way the iteration progresses. (For the text book example we use here it does not matter, but when you are doing serious numerical analysis on real problems, it makes all the difference!)

Obviously, we have only touched on the GSL. My primary intention in this section was to give you a flavor for the way the GSL is designed, and what kinds of considerations you will find incorporated in it. The list of features is very large — consult the documentation for more information.

7.5 Further Reading

- *Guesstimation: Solving the World's Problems on the Back of a Cocktail Napkin*. Lawrence Weinstein and John A. Adam. Princeton University Press. 2008.
This little book contains about a hundred guesstimation problems (with solutions!) from all walks of life. If you are looking for ideas to get you started, look no further.
- *Programming Pearls*. Jon Bentley. 2nd ed., Addison-Wesley. 1999.
and More Programming Pearls: Confessions of a Coder. Jon Bentley. Addison-Wesley. 1989.
These two volumes of reprinted magazine columns are delightful to read, although (or because) they breath the somewhat dated air of the old Bell Labs atmosphere. Both contain chapters on guesstimation problems in a programming context.
- *Back-of-the-Envelope Physics*. Clifford E. Swartz. The Johns Hopkins University Press. 2003.
Physicists regard themselves as the inventors of back-of-the-envelope calculations. This book contains a set of examples from introductory physics. With solutions.

- *The Flying Circus of Physics*. Jearl Walker. 2nd ed., Wiley. 2006.

If you'd like some hints on how to take an interest in the world around you, try this book. It contains hundreds of everyday observations and challenges you to provide an explanation (such as: why are dried coffee stains always darker around the rim? why are shower curtains pulled inwards?). Remarkably, many of these observations are still not fully understood! (You might also want to check out the rather different 1st edition.)

- *Pocket Ref*. Thomas J. Glover. 3rd ed., Sequoia Publishing. 2009.

This little book is an extreme example of the "look-up" model. It seems to contain almost everything: from the strength of wood beams and electrical wiring charts, over properties of materials and planetary data, all the way to first aid, military insignia, and sizing charts for clothing. It also shows the limitations of an overcomplete collection of trivia: I simply don't find it all that useful, but it is interesting for the breadth of topics covered.

Size of an atomic data type	10 bytes
A page of text	55 lines of 80 characters: 4500 characters total
A record (of anything)	100 to 1000bytes
A car	4m long, 1ton weight
A person	2m tall, 100kg weight
A shelf	1m wide, 2m tall
Swimming pool	25 · 12.5 meters (not Olympic — Olympic pools are 50m long)
A storey in a commercial building	4m high
Passengers on a large airplane	350
Speed of a jet liner	1000km/hr
Flight time from NY	6hrs (to LA or Frankfurt, DE)
Human, Walking	1m/s (5km/hr)
Human, maximum power output	200W (not sustainable)
Power consumption of a water kettle	2kW
Electricity grid	100–200V (US versus Europe)
Household fuse	16A
$3 \cdot 3$	10 (minus 10%)
π	3
Large City	1M
Population, Germany or Japan	100M
Population, USA	250M
Population, China or India	1B
Population, Earth	6B
US Median annual income	USD 60k
US Federal Income tax rate	25% (but also as low as 0% and as high as 40%)
Minimum hourly wage	10USD/hr
Billable hours in a year	2000 (50weeks at 40hours/wk)
Low inflation	2%/yr
High inflation	8%/yr
Price of a B2 bomber	USD2billion a piece
American Civil War; Franco-Prussian War	1860s, 1870s
French Revolution	1789
Reformation	1517
Charlemagne	800
Great Pyramids	-3000
Hot Day	35 Celsius
Very Hot Kitchen Oven	250 Celsius
Steel Melts	1200 Celsius
Density of Water	1g/cm ³
Density of Aluminum	3g/cm ³
Density of Lead	13g/cm ³
Density of Gold	20g/cm ³

```

#include <stdio.h>
#include <gsl/gsl_vector.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_blas.h>
#include <gsl/gsl_linalg.h>

int main() {
    double r;

    gsl_vector *a, *b, *s, *t;
    gsl_matrix *m, *v;

    /* --- Vectors --- */
    a = gsl_vector_alloc( 2 );
    b = gsl_vector_alloc( 2 );

    /* a = [ 1.0, 2.0 ] */
    gsl_vector_set( a, 0, 1.0 );
    gsl_vector_set( a, 1, 2.0 );

    /* a = [ 3.0, 6.0 ] */
    gsl_vector_set( b, 0, 3.0 );
    gsl_vector_set( b, 1, 6.0 );

    /* a += b (so that now a = [ 4.0, 8.0 ] ) */
    gsl_vector_add( a, b );
    gsl_vector_fprintf( stdout, a, "%f" );

    /* a . b (dot product) */
    gsl_blas_ddot( a, b, &r );
    fprintf( stdout, "%f\n", r );

    /* --- Matrices --- */
    s = gsl_vector_alloc( 2 );
    t = gsl_vector_alloc( 2 );

    m = gsl_matrix_alloc( 2, 2 );
    v = gsl_matrix_alloc( 2, 2 );

    /* m = [ [1, 2],
           [0, 3] ] */
    gsl_matrix_set( m, 0, 0, 1.0 );
    gsl_matrix_set( m, 0, 1, 2.0 );
    gsl_matrix_set( m, 1, 0, 0.0 );
    gsl_matrix_set( m, 1, 1, 3.0 );

    /* m = U s V^T (SVD : singular values are in vector s) */
    gsl_linalg_SV_decomp( m, v, s, t );
    gsl_vector_fprintf( stdout, s, "%f" );

    /* --- Cleanup --- */
    gsl_vector_free( a );
    gsl_vector_free( b );

```

```

#include <stdio.h>
#include <gsl/gsl_min.h>

double fct( double x, void *params ) {
    return x*x*log(x);
}

int main() {
    double a = 0.1, b = 1; /* interval which bounds the minimum */

    gsl_function f;          /* pointer to the function to minimize */
    gsl_min_fminimizer *s; /* pointer to the minimizer instance */

    f.function = &fct;        /* the function to minimize */
    f.params = NULL;         /* no additional parameters needed */

    /* allocate the minimizer, choosing a particular algorithm */
    s = gsl_min_fminimizer_alloc( gsl_min_fminimizer_goldensection );

    /* initialize the minimizer with a function and an initial interval */
    gsl_min_fminimizer_set( s, &f, (a+b)/2.0, a, b );

    while ( b-a > 1.e-6 ) {
        /* perform one minimization step */
        gsl_min_fminimizer_iterate( s );

        /* obtain the new bounding interval */
        a = gsl_min_fminimizer_x_lower( s );
        b = gsl_min_fminimizer_x_upper( s );

        printf( "%f\t%f\n", a, b );
    }

    printf( "Minimum Position: %f\tValue: %f\n",
            gsl_min_fminimizer_x_minimum(s), gsl_min_fminimizer_f_minimum(s) );

    gsl_min_fminimizer_free( s );
}

return 0;
}

```

Example 7.2: Finding the minimum of a one-dimensional function with the GSL.

Chapter 8

Arguments from Scale

8.1 Models

The next step after familiarizing yourself with the data through plots and graphs is to start building a model for the data. The meaning of the word “model” is quite hazy and I don’t want to spend much time and effort in an attempt to define the abstract concept — for our purposes, a *model* is a mathematical description of the data, which is ideally guided by our understanding of the system under consideration, and which relates the various variables of the system to each other: a “formula”.

Models of this form are incredibly important. It is at this point that we go from the merely *descriptive* (plots and graphs) to the *prescriptive*: having a model allows us to predict what the system will do under a certain set of conditions. And a good, or truly useful, model will allow us to do so without having to resort to the model itself or to evaluate any particular formula explicitly, because it helps us to *understand* how the system works. A good model ties the various variables that control the system together in such a way that we can see how varying any one of them will influence the outcome. It is this use of models, as an aide (or expression) for understanding, that is the most important. (To obtain actual numbers for a specific prediction, we of course still need to evaluate the model formulas explicitly.)

I should point out that this view of models and what they can do is not universal, and you will find the term used quite differently elsewhere. Statistical models, for instance (and this includes machine-learning models as well), are much more descriptive: they do not purport to *explain* the observed behavior in the way just described; their purpose is to predict expected outcomes with the greatest level of accuracy possible (numbers in, numbers out). By contrast, my training is in theoretical physics, where the development of *conceptual understanding* of the observed behavior is the ultimate goal. Because of this I will use all available information about the system and how it works (or even: how I suspect it to work!) wherever I can; I won’t let myself be restricted to using

only the information contained in the data itself. (This is a practice that statisticians are traditionally very wary of, because it constitutes a form of “pollution” of the data. They may very well be right, but my purpose is different: I don’t want to understand the *data*, I want to understand the *system!*) At the same time, absolute accuracy of my models is not paramount to me: a model that gives me order-of-magnitude accuracy, but helps me understand the behavior of the system (so that I can, for instance, make informed trade-off decisions) is much more valuable to me than a one-percent accuracy black-box.

To be clear: there are situations when achieving the best possible accuracy is all that matters, and conceptual understandings are of little interest. (Often this will involve repeatable processes in well-understood systems.) If that describes your situation, then you need to use different methods, appropriate to your problem scenario.

8.1.1 Modeling

As should be clear from the preceding description, building models is basically a creative process, and as such is difficult, if not impossible to teach: there are no established “techniques” or “processes” to arrive at a useful model in any given scenario. One approach to teaching this material therefore is to present a large number of case studies, describing the problem situations and attempts at modeling them. I have not found this style very effective: first of all, every (non-trivial) problem is different, and tricks and fortuitous insights that worked well for one example rarely carry over to a different problem. Moreover, to build effective models often requires pretty deep insight into the particulars of the problem space, and so you end up having to describe lots of tedious detail of the *problem*, when in reality you wanted to talk about the *model* (or rather: the modeling).

Instead, in this chapter, I’ll try a different approach. Effective modeling is often an exercise in determining “what to leave out”: good models should be simple (so that they are workable), but retain the essential features of the system (certainly those that we are interested in!).

As it turns out there are a few essential arguments and approximations that prove helpful again and again to make a complex problem tractable and to identify the dominant behavior. That’s what I want to talk about.

8.1.2 Using and Misusing Models

Just a reminder: models are not reality. They are descriptions or approximations to reality — often quite coarse ones! We need to make sure that we only place as much confidence into a model as is warranted.

How much confidence is warranted? That depends on how well-tested the model is. If a model is based on a good theory, agrees well with a wide range of data sets, and has proven to be able to predict observations correctly, we may have much stronger confidence.

On the other extreme are what one might call “pie-in-the-sky” models: ad-hoc models, involving half a dozen (or so) parameters, all of which have been estimated independently, and not verified against real data. The reliability of such a model is highly dubious: each of the parameters introduces a certain degree of uncertainty, which in combination can make the results of the model meaningless. Remember the discussion in chapter 7: three parameters known to within 10 percent produce an uncertainty in the final result of 30 percent — if the parameters are known to within 10 percent! With four to six parameters, possibly known only much less precisely than 10 percent, the situation is correspondingly worse. (Many business models fall into this category.)

Also keep in mind that basically all models have only a limited region of validity. If you try to apply an existing model to a drastically different situation, or to inputs which are very different than the ones that you used to build the model with, you may find that the model gives poor predictions. Check that the assumptions on which the model is based are in fact fulfilled for each application that you have in mind!

8.2 Arguments from Scale

One office I worked in overlooked the local stadium and the adjacent parking lot. During game days, the parking lot would fill up with cars, and — for obvious reasons — a line of portable toilets would be set up all along one of the edges of the parking lot. I couldn’t help wondering about the balancing problem posed by this setup. Was this particular arrangement going to work for all situations, no matter how large the parking lot in question?

The answer is: No. The number of people in the parking lot grows with the area of the parking lot, which grows with the square of the edge length $\sim L^2$, but the number of toilets is proportional to the edge length itself $\sim L$. Therefore, as we make the parking lot bigger and bigger, there comes a point where the number of people overwhelms the number of available facilities. Guaranteed.

8.2.1 Scaling Arguments

This kind of reasoning is an example of a *scaling argument*. Scaling arguments try to capture how some quantity of interest depends on a control parameter — in particular, a scaling argument describes how the output quantity will *change* as the control parameter changes. Scaling arguments are a particularly fruitful way to arrive at symbolic expressions for phenomena (“formulas”), which can be manipulated analytically.

You should have noted that the expressions I gave earlier were not “dimensionally consistent”. We had people expressed as the square of a length, and toilets expressed as length — what is going on here? Nothing, I merely omitted some detail that was not relevant for the argument I tried to make: a car takes up some amount of space on a parking lot, hence given the size of the

parking lot (its area), we can figure out how many cars it will fit. Each car seats on average 2 people (on a game day), and so we can figure out the number of people as well. Each person has a certain probability of needing to go to a bathroom during the duration of a game, and is going to spend however many minutes there — given all these parameters, we can figure out the required “toilet availability minutes”. (And I can make a similar argument to find the “availability minutes” provided by the installed toilets.) But note that none of these parameters depend on the size of the parking lot: they are constants. Therefore, I don’t need to worry about them if all I want to find out whether this particular arrangement (with toilets all along one edge, but nowhere else) is going to work for all parking lot sizes. (It is a widely followed convention to use the *tilde* $A \sim B$ to express that A “scales as” B , where A and B do not necessarily have the same dimensions.)

On the other hand, if I actually want to know the exact number of toilets I will need for a specific parking lot, then I do need to worry about these factors and try to get the best possible estimates for them.

It is this property that makes scaling arguments such a particularly convenient and powerful way to begin the modeling process: at the beginning, when things are most uncertain and our understanding of the system is least developed, they free us from worrying too much about pesky numerical factors (such as the question: how long does the average person spend on the toilet?), and instead help us concentrate on the overall behavior. Once the big picture has become clearer (and the model still seems worth pursuing), we may want to derive some actual numbers from it as well. Only at this point do we need to worry about numerical constants, which we must estimate or in fact derive from available data.

The previous example (the parking lot) demonstrated one typical application of high-level scaling arguments: what I would call a “no-go argument”: even without any specific numbers, the scaling behavior alone was enough to determine that this particular arrangement of toilets to visitors was going to break down at some point.

Another application of scaling arguments is to cast a question as an optimization problem. Consider a group of people scheduled to perform some task (say, a programming team). Obviously, the amount of work that this group can perform in a fixed amount of time (its “throughput”) is proportional to the number of people on the team: $\sim n$. On the other hand, the members of the team will have to coordinate with each other. Let’s assume each member of the team needs to talk to each other member of the team at least once a day. This implies that we have a communication overhead which scales with the *square* of the number of people $\sim -n^2$. (I have included a minus sign, to indicate that the communication overhead results in a loss in throughput.) This argument alone is enough to show that there will be an optimal number of people for this task, that will have the highest realized productivity. (Also see figure 8.1.)

To find the optimal staffing level, we want to maximize the productivity P

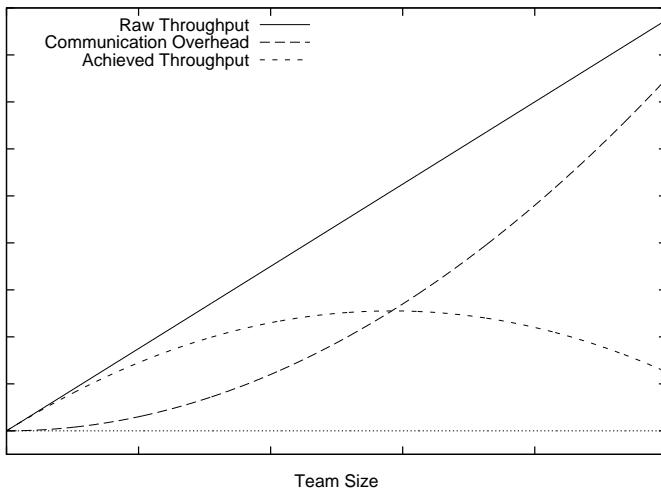


Figure 8.1: TBD

with respect to the number of workers on the team n :

$$P(n) = cn - dn^2$$

where c is the number of minutes each person can contribute during a regular work day and d is the *effective* number of minutes consumed by each communication event. (I'll come back to the cautious "effective" modifier in a moment.)

To find the maximum, we take the derivative of $P(n)$ with respect to n , set it equal to zero, and solve for n (see appendix B). The result is:

$$n_{\text{optimal}} = \frac{c}{2d}$$

Clearly, as the time consumed by each communication event d grows larger, the optimal team size shrinks.

If we now want to find an actual number for the optimal staffing level, then we need to worry about the numerical factors, and this is where the "effective" comes in. The total number of hours people can put in during a regular work day is pretty easy to estimate (eight hours at sixty minutes, less time for diversions), but the amount of time spent in a single communication event is harder to determine. There are also additional effects, which I would choose to lump into the "effective" parameter: for example, not everybody on the team needs to talk to everybody else — maybe only half the people need to talk to somebody else. Adjustments like this can be lumped into the parameter d , making it more and more a synthetic parameter and less and less one that can be measured directly.

8.2.2 Case Study: A Simple Cost Model

Models don't have to be particularly complicated to provide important insight. In one place I worked at, we were trying to improve the operation of what was essentially a manufacturing environment. There was a job that was done on a special machine, which had to be retooled for each different item type to be produced. So, first the machine would be set-up (which took about five to ten minutes), and then a worker would operate the machine to produce a batch of 150 to 200 identical items. The whole cycle lasted a bit longer than one-and-a-half hours before the batch would be completed, and the machine was retooled for the next batch.

The retooling part of the cycle was a constant source of management frustration: for ten minutes (while the machine was being set up), nothing seemed to be happening. Wasted time! (In manufacturing, productivity, defined as "units per hour" is the most closely watched metric.) Consequently, there had been a long string of process improvement projects, with the purpose of making the retooling part more efficient and thereby faster. By the time I arrived, it had been streamlined very well. Nevertheless, there were constant efforts underway to reduce the time it took — after all, the sight of the machine sitting idle for ten minutes seemed to be all the proof that was needed.

It is interesting to set up a minimal cost model for this process. The relevant quantity to study is "minutes per unit". This is essentially the inverse of the productivity, but I find it easier to think in the time it takes to produce a single unit than the other way around. Also note that "time per unit" equates to "cost per unit", once we take the hourly wage into account. So, the time per unit is the time T it took to produce the entire batch, divided by the number of items n in the batch. The total processing time itself is composed of the setup time T_1 and n times the amount of time t required to produce a single item:

$$\begin{aligned}\frac{T}{n} &= (T_1 + nt) / n \\ &= \frac{T_1}{n} + t\end{aligned}$$

The first term on the right-hand side is the amount of the setup time that can be attributed to a single item, and the second term, or course, is the time to actually produce the item. The larger the batch size, the smaller the contribution of the setup time to the cost of each item — the setup time is "amortized" over more units.

This is one of the situations where the numerical factors actually matter. We know that T_1 is in the range of 300–600 seconds, and that n is between 150 and 200, so that T_1/n varies between 1–4 seconds per item. By contrast, we can find the time t to actually produce a single item if we recall that the cycle time for the entire batch was about 90 minutes, therefore $t = 90 \cdot 60 / n$, which comes to about 30 seconds per item. In other words, the setup time that caused management so much grief, accounted for less than 10 percent of the total time to produce an item!

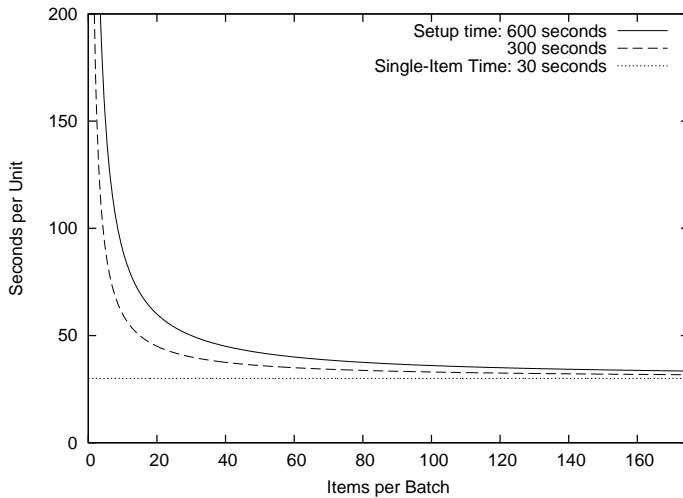


Figure 8.2: TBD

But we aren't finished yet! Let's assume that through some strenuous effort we were able to reduce the setup time by 10 percent. (Not very likely, because this was the part of the process that had already been receiving a lot of attention, but let's assume — best case!) This would mean that we can reduce the setup time *per item* to 1–3.5 seconds — but this means that the *total* time per item is reduced by only 1 or 2 percent! This is the kind of efficiency gain that only makes sense in very, very controlled situations, where *everything* else is completely optimized. In contrast, a 10 percent reduction in t , the actual work time per item, would result in (basically) a 10 percent improvement in overall productivity (because the amount of time that it takes to produce an item is so much larger than the fraction of the setup time attributable to a single item).

We can see this in figure 8.2, showing the "loaded" time per unit (including the setup time) for two typical values of the setup time as a function of the number of items produced in a single batch. Although the setup time contributes significantly to the per-item time if there are less than about 50 items per batch, its effect is very small for batch sizes of 150 and above: for batches of this size, the time it takes to actually *make* an item dominates over the time to retool the machine.

The story is still not finished. I tried to get a project sponsored to look at ways to reduce t for a change, and eventually got it launched — but it was never really adopted and shut down at the earliest possible moment by plant management, in favor of a project to look at, you guessed it, the setup time! The sight of the machine sitting idle for ten minutes was more than any self-respecting plant manager could bear.

The very important take-away message here is this: facts and numbers

carry very little credibility, in the face of apparent (!) hands-on, immediate, visual evidence to the contrary. And in the face of long-standing habits. (We will come back to this in the epilogue, 19.)

8.2.3 Typical Scales and Collapsing Curves

Coming soon

8.2.4 Finding the Right Scales

Coming soon

8.2.5 Scaling Arguments versus Dimensional Analysis

Scaling arguments may seem similar to another concept you may have heard of: *dimensional analysis*, but — although related — they are really quite different. Scaling concepts, as I have presented them, are based on our intuition of the way the system behaves, and are a way to capture this intuition in a mathematical expression.

By contrast, dimensional analysis applies to physical systems, which are described by a number of quantities, having different physical *dimensions*, such as length, mass, time, temperature. Because we are talking about physical systems, equations describing them must be dimensionally consistent, and we can try to deduce the form of these equations by forming dimensionally consistent combinations of the relevant variables.

Let's look at an example. Everybody is familiar with the phenomenon of air resistance or drag: there is a force F that acts to slow a moving body down. It seems reasonable to assume that this force depends on the cross-sectional area of the body A and the speed v . But it must also depend on some property of the medium (air, in this case) that the body moves through — the most basic property is the density ρ , which is the mass (in grams or kilograms) per volume (in cubic centimeters or meters):

$$F = f(A, v, \rho)$$

Here, $f(x, y, z)$ is an as-yet unknown function.

Force has units $mass \cdot length^2 / time^2$, area has units $length^2$, velocity $length / time$, and finally density $mass / length^3$. We can now try to combine A , v , and ρ to form a combination that has the same dimensions as force. A little experimentation leads us to:

$$F = c\rho Av^2$$

where c is a pure (dimensionless) number. This equation expresses the well-known result that air resistance increases with the square of the speed. Note that we arrived at it without any insight into the physical mechanisms at work, using purely dimensional arguments.

This form of reasoning has a certain kind of magic to it: why did we choose these specific quantities? Why did we not include the viscosity of air, or the ambient air pressure, or the temperature, or the length of the body? The answer is (mostly): physical intuition. The viscosity of air is small (viscosity measures the resistance to shear stress, that is the force transmitted by a fluid captured between parallel plates moving parallel

to each other but in opposite directions — clearly, not a large effect for air at macroscopic length scales). The pressure enters through the density (at constant temperature, according to the ideal gas law). And the length of the body is hidden in the numerical factor c , which depends on the shape of the body and therefore on the ratio of the cross-sectional radius \sqrt{A} and the length. In summary: it is quite impressive that we got as far as we did using only very simple arguments, but it is hard to overcome a certain level of discomfort entirely!

Methods of dimensional analysis appear less arbitrary when the governing equations are known. In this case, we can use dimensional arguments to reduce the number of independently variable quantities. For example, *assume* that we already know that the drag force is described by $F = c\rho Av^2$, and that we want to perform experiments to determine c for various bodies, by measuring the drag force on them under various conditions. Naively, it might appear that we have to map out the full three-dimensional parameter space by making measurements for all combinations of (ρ, A, v) . But realizing that these three parameters only occur in the combination $\gamma = \rho Av^2$, it is sufficient to run a single series of tests that varies γ over the range of values we are interested in. This constitutes a significant simplification!

Dimensional analysis relies on dimensional consistency, and therefore works best for physical and engineering systems, which are described by independently measurable, dimensional quantities. It is particularly prevalent in areas such as fluid dynamics, where the number of variables is particularly high and the physical laws complicated and often not well understood. It is much less applicable in economic or social settings, where there are fewer (if any) rigorously established dimensionally consistent relationships.

8.2.6 Other Arguments

There are other arguments that can be very powerful when attempting to formulate models. They come from the physical sciences, and (as with dimensional analysis) may not work as well in social and economic settings, which are not governed by strict physical laws.

- Conservation Laws. Conservation laws tell us that some quantity does not change over time. The best-known example is the law of conservation of energy. Conservation laws can be very powerful (in particular if they are exact), but may not be available: after all, the entire idea of economic growth and (up to a point) manufacturing itself rest on the assumption that more comes out than is being put in!
- Symmetries. Symmetries, too, can be very helpful in reducing complexity: if I know that an apparently two-dimensional system exhibits the symmetry of a circle, I know that I am only dealing with a one-dimensional problem, because any variation can only occur in the *radial* direction, since a circle looks the same in all directions. When looking for symmetries, don't restrict yourself to geometric considerations — for example, if you know that items are entering and leaving a buffer at the same rate, then this is a form of symmetry. In this case, you might only have to

solve one of the two processes explicitly, and treat the other one as mirror image of the first.

- Extreme Value Considerations. How does the system behave at the extremes? If there are no customers, messages, orders, items? How if there are infinitely many? What if the items are very large or vanishingly small, or if we wait an infinite amount of time? Considerations such as these can be helpful to sanity check an existing model, but can also provide inspiration when first establishing a model: limiting cases are often easier to treat, because only one effect dominates, so that the complexities which arise out of the interplay of different factors disappear.

8.3 Mean-Field Approximations

The term *mean-field approximation* comes from statistical physics, but I use it only as a convenient and intuitive expression for a much more general approximation scheme.

Statistical physics deals with large systems of interacting particles, such as gas molecules in a piston, or atoms on a crystal lattice. These systems are extraordinarily complicated, because every particle interacts with every other particle. If you now move one of the particles, this will affect all the other particles, and so they will move, too...but their movement, in turn, will influence the first particle that we started with! Finding exact solutions for such large, coupled systems is often impossible. To make progress, we ignore the individual interactions between explicit pairs of particles. Instead, we assume that the test particle experiences a field, the “mean-field”, which captures the “average” effect of all the other particles.

For example, consider N gas atoms in a bottle of volume V . We may be interested to understand how often two gas atoms collide with each other. Of course to do that, we would have to follow every single atom over time, to see whether it bumps into any of the other atoms. This is obviously very difficult, and it certainly seems as if it forces us to keep track of a whole lot of detail that should be unnecessary, if we are only interested in macroscopic properties.

Realizing this, we can consider this gas in a mean-field approximation: the probability that our test particles collides with another particle should be proportional to the average density of particles in that bottle $\rho = N/V$. Since there are N particles in the bottle, we expect the number of collisions (over some timeframe) to be proportional to $N\rho$. This is good enough to start making some predictions — for example, note that this expression is proportional to N^2 . Doubling the number of particles in the bottle therefore means that the number of collisions will grow by a factor of 4. On the other hand, reducing the volume of the container by half will drive the number of collisions up by only a factor of 2.

You will have noticed that in the previous argument I have omitted lots of detail — for example, any reference to the time frame over which I intend to

Exact	Mean-Field
$E[x] = \sum_{\text{all outcomes } x} F(x)p(x)$	$E_{\text{MF}}[x] = F\left(\sum_{\text{all outcomes } x} p(x)\right)$

Table 8.1: Mean-field approximations replace an average over functions with functions of averages.

count collisions. There is also a constant of proportionality missing: $N\rho$ is not really the number of collisions, it is merely proportional to it. But if all I care about is to understand how the number of collisions depends on the two variables I consider explicitly (namely N and V), then I don't need to worry about any of these details. The argument given above has been entirely sufficient to work out how the number of collisions scales with both N and V .

You can see how mean-field approximations and scaling arguments enhance and support each other!

After this introductory example, let's step back and look at the concept behind mean-field approximations more closely.

8.3.1 Background and Further Examples

If mean-field approximations were limited to systems of interacting particles, they would not be of much interest in this book. However, the concept behind them is much more general and very widely applicable.

Whenever we want to calculate with a quantity that is distributed according to some probability distribution, we face the challenge that this quantity does not have a fixed value: it has a whole spectrum of possible values, each being more or less likely according to the probability distribution. Operating with such a quantity is difficult, because we have to perform all calculation (at least in principle) for each possible outcome and then weight the result of our calculation by the appropriate probability. At the very end of the calculation, we eventually form the average (properly weighted according to the probability factors), to arrive at a unique numerical value.

Attempting to do this kind of calculation exactly, with its concomitant combinatorial explosion of possible outcomes, invariably starts to feel like wading in a quagmire — and that assumes that the calculation can actually be carried out exactly at all!

The mean-field approach cuts through this difficulty by performing the average *before* embarking on the actual calculation. Rather than working with all possible outcomes (and averaging them at the end), we work out what the average outcome is at the beginning, and then only work with that value alone. Table 8.1 summarizes the differences.

That sounds formidable, but is actually something we do all the time. Do you ever try to estimate how high the bill is going to be when you are waiting in line at the supermarket? You can do this explicitly, by going through all the items individually, adding up their prices (approximately) in your head

— or you can apply a mean-field approximation, by realizing that the items in your cart represent a sample, drawn “at random”, from the selection of the supermarket. In the mean-field approximation, you would now estimate the average single-item price for that supermarket (probably something around \$5–\$7) and multiply that value with the number of items in your cart. Note that it should be much easier to simply count the items in your cart, as opposed to doing the explicit sum.

This example also highlights the potential pitfalls with mean-field arguments: it will only be reliable if the average item price is a good estimator! If your cart contains two bottles of champagne and a rib-roast for a party of eight people, an estimate based on a typical item price of \$7 is going to be *way* off.

To get a grip on the expected accuracy of the mean-field approximation, we can try to find a measure for the width of the original distribution (such as its standard deviation or inter-quartile range) and then repeat our calculations after adding (and subtracting) the width from the mean value. (We may also treat the width as a small perturbation to the average value and use the perturbation methods from chapter 7.)

Another example: how many packages does UPS (or any comparable freight carrier) fit onto a truck (to be clear: here I don’t mean a delivery truck, but one of these 53 feet tractor-trailer long-hauls)? Well, we can estimate the “typical” size of a package as about a cubic foot (0.3^3m^3), but it might also be as small as half that, or as large as twice that size. To get an estimate for the number of packages that will fit, we divide the volume of the truck (17 meters long, 2 meters wide, 2.5 meters high — you can stand upright in these things) by the typical size of a package: $(17 \cdot 2 \cdot 2.5 / 0.3^3) \approx 3000$ packages. Since we said that the volume (not the length!) each package might vary by as much as a factor of two, we end up with lower and upper bounds of 1500 to 6000 packages.

This calculation makes use of the mean-field idea twice: first, we work with the “average” package size; second, we don’t worry about the actual spatial packing of boxes inside the truck, instead, we pretend that we can reshape them like putty. This also is a form of “mean-field” approximation.

I hope you appreciate how the mean-field idea has turned this problem from an almost impossibly difficult one into a trivial one — and I don’t just mean in regards to the actual computation and the eventual numerical result, but more importantly in the way we think about it. Rather than getting stuck in the enormous technical difficulties of working out different stacking orders for packages of different sizes, the mean-field notion reduced the problem description to the most fundamental question: into how many small pieces can we divide a large volume? (And if you think that all of this is rather trivial, I fully agree with you — but the “trivial” is easily overlooked when presented with a complex problem in all of its ugly detail. Trying to find mean-field descriptions helps to strip away non-essential detail and to reveal the fundamental questions at stake.)

One common feature of mean-field solutions is that they frequently violate some of the properties of the system. For example, when I worked for Amazon, we would often consider the typical order to contain 1.7 items, of which 0.9

were books, 0.3 were CDs, and the remaining 0.5 items were other stuff (or whatever the numbers were). This is obviously nonsense, but don't let this disturb you! Just carry on as if nothing happened, and work out the correct break-down of things at the end. This does not always work (if you try to schedule people for jobs, and it turns out that one job only requires one-tenth of a person, you may still have to assign a full-time worker to that task!), but this kind of argument is often sufficient to work out the general behavior of things.

There is a story involving Richard Feynman working on the Connection Machine, one of the earliest massively parallel super-computers. All the other people on the team were computer scientists, and when a certain problem came up, they tried to solve it using discrete methods and exact enumerations, and got stuck with it. By contrast, he worked with quantities such as "the average number of 1 bits in a message address" (clearly a mean-field approach), which allowed him to cast the problem in terms of partial differential equations, which were easier to solve...¹

8.4 Common Time-Evolution Scenarios

Sometimes we can propose a model based on the way the system under consideration evolves. The "proper" way to do this is to write down a differential equation that describes the system (and, in fact, that is exactly what the term "modeling" often means), and then to proceed and solve it, but that would take us too far afield. (Differential equations relate the change in some quantity, expressed through its derivative, to the quantity itself. These equations can be solved to yield the quantity for all times.)

However, there are a few scenarios which are so fundamental and so common, that we can go ahead and simply write down the solution in its final form. (I'll give a few notes on the derivation as well, but it is the solutions to these differential equations that should be committed to memory.)

8.4.1 Unconstrained Growth and Decay Phenomena

The simplest case concerns pure growth (or death) processes. If the *rate* of change of some quantity is constant in time, the quantity will follow an *exponential* growth (or decay). Consider a cell culture. At every time step, a certain fraction of all cells in existence at that time step will split (that is: generate offspring). Here, the *fraction* of cells that participate in the population growth at every time step is constant in time, but because the population itself grows, the total number of new cells at each time step is larger than at the previous time step. Many pure growth processes exhibit this behavior — compound interest on a monetary amount is another example (see chapter 17).

¹This story can be found in the paper "Richard Feynman and The Connection Machine" by Daniel Hillis, which can be found on the web; it appeared in print in Physics Today, Volume 42, February 1989).

Pure death processes work similarly, only that in this case a constant fraction of the population dies or disappears at each time step. Radioactive decay is probably the best known example; but another one is the attenuation of light in a transparent medium (such as water). For every unit of length that light penetrates into the medium, its intensity is reduced by a constant fraction, giving rise to the same exponential behavior. In this case, the independent variable is not time, but space; however, the argument is exactly the same.

Mathematically, we can express the behavior of a cell culture as follows: if $N(t)$ is the number of cells alive at time t , and a fraction f of these cells split into new cells, then the number of cells at the next time step $t + 1$ will be:

$$N(t+1) = N(t) + fN(t)$$

Here, the first term on the right-hand side comes from the cells which were already alive at time t , whereas the second term on the right comes from the “new” cells created at t . We can now rewrite this equation as follows:

$$N(t+1) - N(t) = fN(t)$$

This is a *difference equation*. If we can assume that the time “step” is very small, we can replace the left-hand side with the derivative of N (this process is not always quite as simple as in this example — you may also check appendix B for more details on difference and differential quotients):

$$\frac{d}{dt}N = \frac{1}{T}N(t)$$

This equation is true for growth processes; for pure death processes instead we have an additional minus sign on the right-hand side.

These equations can be solved or integrated explicitly, and their solutions are:

$N(t) = N_0 e^{t/T}$	Pure Birth Process
$N(t) = N_0 e^{-t/T}$	Pure Death Process

Instead of using the “fraction” f of new or dying cells that we used in the difference equation, here we employ a characteristic *time scale* T , which is the time over which the number of cells changes by a factor e or $1/e$, where $e = 2.71828 \dots$. The value for this time scale will depend on the actual system — for cells that multiply rapidly, T will be smaller than for another species that grows more slowly. Notice that such a scale factor *must* be there, to make the argument of the exponential function dimensionally consistent! Furthermore, the parameter N_0 is the number of cells in existence at the beginning $t = 0$.

Exponential process (either birth or death) are very important, but they never last very long. In a pure death process, the population dwindles very quickly to practically nothing. At $t = 3T$, only five percent of the original population are left; at $t = 10T$, less than one in ten-thousand of the original cells

has survived; at $t = 20T$, we are down to one in a billion. In other words, after a time which is a small multiple of T , the population will have all but disappeared.

Pure birth processes face the opposite problem: the population grows so quickly, that after a very short while it will exceed the capacity of its environment. This is so generally true, that it is worth emphasizing: exponential growth is not sustainable over extended time periods. A process may start out exponential, but before long, it must and will saturate. That brings us to the next scenario...

8.4.2 Constrained Growth: The Logistic Equation

Pure birth processes never continue for very long: the population grows very quickly to a size that is unsustainable, and the growth slows. A very common model to take this behavior into account assumes that the members of the population start to “crowd” each other, possibly competing for some shared resource, such as food or territory. Mathematically, this can be expressed as:

$$\frac{d}{dt}N = \lambda N(K - N) \quad \lambda, K > 0 \text{ fixed}$$

The first term on the right-hand side (λKN) is the same as in the exponential growth equation: by itself, it would lead to an exponentially growing population $N(t) = C \exp(\lambda Kt)$. But the second term $-\lambda N^2$ counteracts this: it is negative, so its effect is to *reduce* the population, and it is proportional to N^2 , so that it grows more strongly as N becomes large. (You can motivate the form of this term by observing that it measures the number of collisions between members of the population, and therefore expresses the “crowding” effect mentioned earlier.)

This equation is known as the *logistic differential equation* and it is solved by the *logistic function*:

$$N(t) = \frac{K}{1 + \left(\frac{K}{N_0} - 1 \right) e^{-\lambda Kt}}$$

This is a complicated function, depending on three parameters:

- λ the characteristic growth rate
- K the carrying capacity: $K = N(t \rightarrow \infty)$
- N_0 the initial number of cells: $N_0 = N(t = 0)$

Compared to a pure (exponential) growth process, the appearance of the parameter K is new. It stands for the “carrying capacity” of the system, that is the maximum number of cells that the environment can support. You should convince yourself that the logistic function indeed tends to K as t becomes

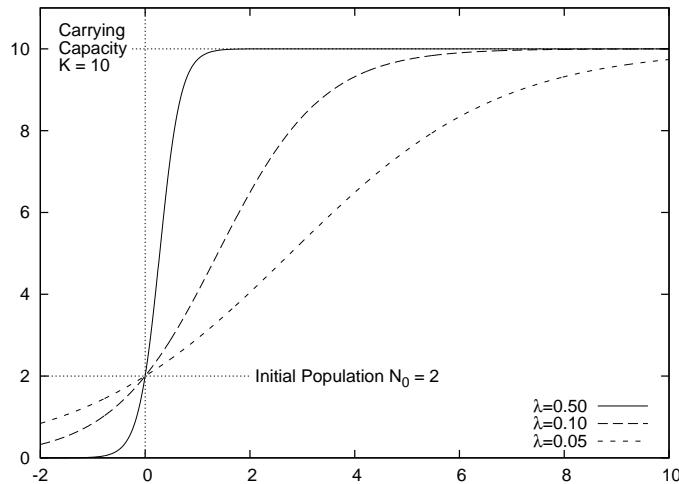


Figure 8.3: TBD

large. (You will find different forms of this function in the literature, with different parameters; but the form given here is the most useful one.) Figure 8.3 shows the logistic function for a selection of parameter values.

I should point out that determining values for the three parameters from data can be extraordinarily difficult, in particular if only data points to the left of the inflection point (the point with maximum slope, about halfway between N_0 and K) are available: many different combinations of λ , K , and N_0 may seem to fit the data about equally well. In particular, it is difficult to assess K from early-stage data alone — you may want to try and obtain an independent estimate for the carrying capacity (even a very rough one) and use that when determining the remaining parameters from the data.

The logistic function is the most common model for all growth processes that exhibit some form of saturation. Infection rates for contagious diseases, for example, can also be modeled using the logistic equations, or even the approach to equilibrium for cache hit rates.

8.4.3 Oscillations

The last of the common dynamical behavior concerns systems in which some quantity has an equilibrium value and responds to excursions from that equilibrium position with a restoring effect, which drives the system back to the equilibrium position. If the system does not come to rest in the equilibrium position, but overshoots, the process will continue, going back and forth across the neutral position — in other words, the system undergoes *oscillation*. Oscillations of course occur in physical systems (from tides over grand-father clocks

to molecular bonds), but the “restore-and-overshoot” phenomenon is much more general, so that oscillations can be found almost everywhere: the pendulum that has “swung the other way” is proverbial, from the political scene to personal relationships.

Oscillations are periodic: the system undergoes the same motion again and again. The simplest functions that exhibit this kind of behavior are the trigonometric functions $\sin(x)$ and $\cos(x)$ (also see appendix B), therefore we can express any periodic behavior (at least approximately) in terms of sines or cosines. (Sine and cosine are periodic with period 2π . To express an oscillation with period D we therefore need to rescale x by $2\pi/D$. It may also be necessary to shift x by a phase factor ϕ : an expression like $\sin(2\pi(x - \phi)/D)$ will be an at least approximate description for any periodic data set.)

But it gets better: a powerful theorem states that *every* periodic function, no matter how crazy, can be written as a (possibly infinite) combination of trigonometric functions called a *Fourier Series*. A Fourier Series looks like this:

$$f(x) = \sum_{n=1}^{\infty} a_n \sin\left(2\pi n \frac{x}{D}\right)$$

where I have assumed that $\phi = 0$. The important point is that only integer multiples of $2\pi/D$ are being used in the argument of the sine: the so-called “higher harmonics” of $\sin(2\pi x/D)$. We need to adjust the coefficients a_n to describe a data set. Although the series is in principle infinite, we can usually get pretty good results by truncating after only a few terms. (We have already seen an example for this in chapter 6, when we used the first two terms to describe the variation of the CO₂ concentration over Mauna Loa on Hawaii.)

If the function is known exactly, the coefficients a_n can be worked out. For the sawtooth function (see figure 8.4), the coefficients are simply $1, 1/2, 1/3, 1/4, \dots$ with alternating signs:

$$f(x) = \frac{\sin x}{1} - \frac{\sin 2x}{2} + \frac{\sin 3x}{3} - \dots$$

You can see that the series converges quite rapidly, even for such a crazy, discontinuous function as the sawtooth.

8.5 Case Study: How Many Servers Are Best?

To close out this chapter, let’s discuss an additional simple case study in model building.

Imagine you are deciding how many servers to purchase to power your ecommerce site. Each server costs you a fixed amount E per day — this includes both the operational cost for power and colocation, as well as the amortized acquisition cost (that is the purchase price divided by the number of days until the server is obsolete and will be replaced). The total cost for n servers is therefore nE .

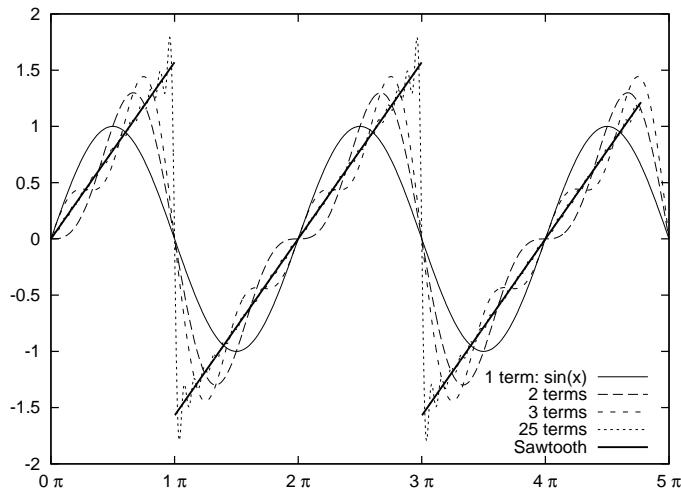


Figure 8.4: The sawtooth function can be composed out of sine functions and their higher harmonics.

Given the expected traffic, one server should be sufficient to handle the load. However, each server has a finite probability p to fail on any given day. If your site goes down, you expect to lose B in profit before a new server can be provisioned and brought back online. Therefore, the expected loss when using a single server is pB .

Of course, you can improve the reliability of your site by using multiple servers. If you have n servers, your site will only be down if *all* of them fail simultaneously. The probability for this event is p^n . (Note that $p^n < p$, since p is a probability and therefore $p < 1$.)

The total daily cost C that you incur can now be written as the combination of the fixed cost nE and the expected loss due to server downtime p^nB (also see figure 8.5):

$$C = p^nB + nE$$

Given p , B , and E , you would like to minimize this cost with respect to the number of servers n . We can do this either analytically (by taking the derivative of C with respect to n) or numerically.

But wait, there is more! Let's say that we also have an alternative proposal in front of us, to provision our data center with servers from a different vendor. We know that their reliability q is worse (so that $q > p$), but their price F is significantly lower ($F \ll E$). How does this variant compare to the previous one?

The answer depends on the values for p , B , and E . To make a decision, we will not only have to evaluate the *location* of the minimum in the total cost (that is: the number of servers required), but also the actual *value* of the total cost at

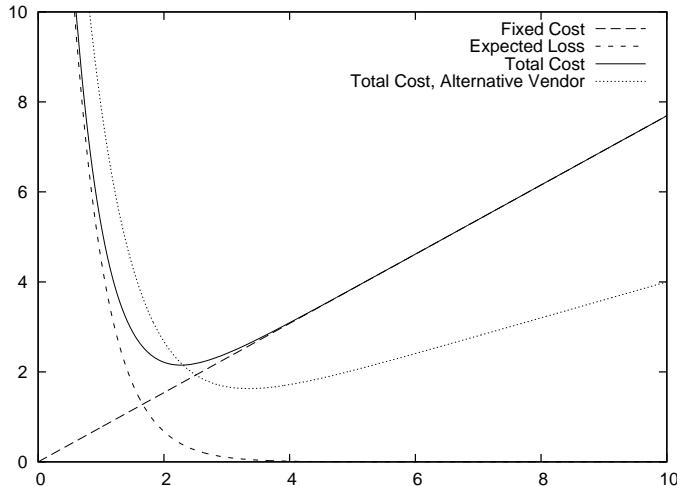


Figure 8.5: Costs associated with provisioning a data center, as a function of the number of servers.

the minimum position. Figure 8.5 includes the total cost for an alternative proposal, using less reliable, but much cheaper servers: although we need more servers if we decide to go with this solution, the total cost is nevertheless lower, than in the first proposal.

(We can go even further: how about a mix of different servers? This scenario, too, we can model in a similar fashion and evaluate it against its alternatives...)

8.6 Why Modeling?

Why worry about modeling in a book on data *analysis*? We rarely seem to have touched actual data in all of the examples in this chapter.

It all depends on your goals when working with data. If all you want to do is to describe it, extract some features, or even decompose it fully into its constituent parts, then the “analytic” methods of graphical and data analysis will suffice. However, if you intend to use the data to develop an understanding of the *system* that produced the data, then looking at the data itself will only be a first (although very important!) step.

I consider conceptual modeling extremely important — as I said earlier, it is at this point that we go from the descriptive to the prescriptive. A conceptual model may be the most valuable outcome of an analysis, but at the very least it will also enhance the purely analytical part of our work: more likely than not, a conceptual model will lead to additional hypothesis and thereby suggest

additional ways to look at and study the data in an iterative process.

The methods described in this chapter and in the next (chapter 9 I have found to be the most practically useful when thinking about data and the processes that generated it. Whenever I am looking at data, I always try understand the system behind it, and I always use some, if not all, of the methods from these two chapters.

8.7 Workshop: SAGE

Most of the tools that we have (and will) introduce in this book work with *numbers*, and since we are mostly interested in understanding data, that makes sense. However, a different kind of tool exists, which works with formulas instead: so-called *computer algebra systems*. The big (commercial) brand names for computer algebra systems have been Maple and Mathematica; in the open-source world, the SAGE project (www.sagemath.org) has become somewhat of a front runner.

SAGE is an “umbrella” project, which attempts to combine various already existing open-source projects (SymPy, Maxima, and others), together with some added functionality, into a single coherent, Python-like environment. SAGE places heavy emphasis on features for number theory and abstract algebra (not exactly everybody’s cup of tea), and also includes support for numerical calculations and graphics, but in this section we will limit ourselves to basic calculus and a little linear algebra — mostly because that is the part that is unique to SAGE, whereas we have seen many other tools for visualization and numerics. (A word of warning: if you are not really comfortable with calculus, you probably want to skip the rest of this section. Don’t worry — it won’t be needed in the rest of the book.)

Once you start SAGE, it drops you into a text-based command interpreter (a “REPL”: a read-eval-print loop). SAGE makes it very easy to perform some simple calculations. For example, let’s define a function and take its derivative:

```
sage: a, x = var( 'a x' )
sage: f(x) = cos(a*x)
sage: diff( f, x )
x |--> -a*sin(a*x)
```

In the first line, we declare *a* and *x* as symbolic variables, so that we can refer to them later, and SAGE knows how to handle them. We then define a function, using the “mathematical” notation $f(x) = \dots$. Only functions defined in this way can be used in symbolic calculations. (It is also possible to define Python functions, using regular Python syntax: `def f(x, a): return cos(a*x)`, but such functions can only be evaluated numerically.) Finally, we calculate the first derivative of the function just defined.

All the standard calculus operations are available. We can combine functions to obtain more complex ones, we can find integrals (both definite and indefinite), we can even evaluate limits:

```
sage: # Indefinite integral:
sage: integrate( f(x,a) + a*x^2, x )
1/3*a*x^3 + sin(a*x)/a
sage:
sage: # Definite integral on [0,1]:
sage: integrate( f(x,a) + a*x^2, x, 0, 1 )
1/3*(a^2 + 3*sin(a))/a
sage:
sage: # Definite integral on [0,pi], assigned to function:
sage: g(x,a) = integrate( f(x,a) + a*x^2, x, 0, pi )
sage:
sage: # Evaluate g(x,a) for different a:
sage: g(x,1)
1/3*pi^3
sage: g(x,1/2)
1/6*pi^3 + 2
sage: g(x,0)
-----
RuntimeError
(some output omitted...)
RuntimeError: power::eval(): division by zero
sage: limit( g(x,a), a=0 )
pi
```

In the second to last line, we tried to evaluate an expression that is mathematically not well defined: the function $g(x,a)$ includes a term of the form $\sin(\pi a)/a$, which we can't evaluate for $a = 0$, because we can't divide by zero. However, the limit $\lim_{a \rightarrow 0} \sin(\pi a)/a = \pi$ exists and is found by the `limit()` function.

As a final example from calculus, let's evaluate some Taylor series (the arguments are: the function to expand, the variable to expand in, the point around which to expand, and finally the degree of the desired expansion).

```
sage: taylor( f(x,a), x, 0, 5 )
1/24*a^4*x^4 - 1/2*a^2*x^2 + 1
sage: taylor( sqrt(1+x), x, 0, 3 )
1/16*x^3 - 1/8*x^2 + 1/2*x + 1
```

So much about basic calculus. Let's also visit an example from linear algebra. Imagine we have the linear system of equations:

$$\begin{array}{rcl} ax & + & by \\ 2x & + & ay + 3z = 2 \\ b^2x & & - z = a \end{array}$$

and we would like to find those values of (x, y, z) which solve this system. If all the coefficients were numbers, then we could use a numeric routine to obtain the solution, but in this case, some of the coefficients are only known symbolically (as a and b) and we would like to express the solution in terms of these variables.

SAGE can do this for us quite easily:

```
sage: a, b, x, y, z = var('a b x y z')
sage:
sage: eq1 = a*x + b*y == 1
sage: eq2 = 2*x + a*y + 3*z == 2
sage: eq3 = b^2 - z == a
sage:
sage: solve([eq1,eq2,eq3], x,y,z)
[[x == (3*b^3 - (3*a + 2)*b + a)/(a^2 - 2*b),
y == -(3*a*b^2 - 3*a^2 - 2*a + 2)/(a^2 - 2*b),
z == b^2 - a]]
```

As a last example, let's demonstrate how to calculate the eigenvalues of the matrix:

$$M = \begin{pmatrix} a & b & a \\ b & c & b \\ a & b & 0 \end{pmatrix}$$

Again, if the matrix was given numerically, we could use a numeric algorithm, but here we would like to obtain a symbolic solution.

Again, SAGE can do this quite easily:

```
sage: m = matrix([[a,b,a],[b,c,b],[a,b,0]])
sage: m.eigenvalues()
[-1/18*(-I*sqrt(3) + 1)*(4*a^2 - a*c + 6*b^2 + c^2)/(11/54*a^3 - 7/18*a^2*c + 1/3*b^2*c + 1/27*c^3 + 1/18*(15*b^2 - c^2)*a + 1/18*sqrt(-5*a^6 - 6*a^4*b^2 + 11*a^2*b^4 - 5*a^2*c^4 - 32*b^6 + 2*(5*a^3 + 4*a*b^2)*c^3 + (5*a^4 - 62*a^2*b^2 - 4*b^4)*c^2 - 2*(5*a^5 + 17*a^3*b^2 - 38*a*b^4)*c)*sqrt(3))^(1/3) - 1/2*(I*sqrt(3) + 1)*(11/54*a^3 - 7/18*a^2*c + 1/3*b^2*c + 1/27*c^3 + 1/18*(15*b^2 - c^2)*a + 1/18*sqrt(-5*a^6 - 6*a^4*b^2 + 11*a^2*b^4 - 5*a^2*c^4 - 32*b^6 + 2*(5*a^3 + 4*a*b^2)*c^3 + (5*a^4 - 62*a^2*b^2 - 4*b^4)*c^2 - 2*(5*a^5 + 17*a^3*b^2 - 38*a*b^4)*c)*sqrt(3))^(1/3) + 1/3*a + 1/3*c, -1/18*(I*sqrt(3) + 1)*(4*a^2 - a*c + 6*b^2 + c^2)/(11/54*a^3 - 7/18*a^2*c + 1/3*b^2*c + 1/27*c^3 + 1/18*(15*b^2 - c^2)*a + 1/18*sqrt(-5*a^6 - 6*a^4*b^2 + 11*a^2*b^4 - 5*a^2*c^4 - 32*b^6 + 2*(5*a^3 + 4*a*b^2)*c^3 + (5*a^4 - 62*a^2*b^2 - 4*b^4)*c^2 - 2*(5*a^5 + 17*a^3*b^2 - 38*a*b^4)*c)*sqrt(3))^(1/3) - 1/2*(-I*sqrt(3) + 1)*(11/54*a^3 - 7/18*a^2*c + 1/3*b^2*c + 1/27*c^3 + 1/18*(15*b^2 - c^2)*a + 1/18*sqrt(-5*a^6 - 6*a^4*b^2 + 11*a^2*b^4 - 5*a^2*c^4 - 32*b^6 + 2*(5*a^3 + 4*a*b^2)*c^3 + (5*a^4 - 62*a^2*b^2 - 4*b^4)*c^2 - 2*(5*a^5 + 17*a^3*b^2 - 38*a*b^4)*c)*sqrt(3))^(1/3) + 1/3*a + 1/3*c, 1/3*a + 1/3*c + 1/9*(4*a^2 - a*c + 6*b^2 + c^2)/(11/54*a^3 - 7/18*a^2*c + 1/3*b^2*c + 1/27*c^3 + 1/18*(15*b^2 - c^2)*a + 1/18*sqrt(-5*a^6 - 6*a^4*b^2 + 11*a^2*b^4 - 5*a^2*c^4 - 32*b^6 + 2*(5*a^3 + 4*a*b^2)*c^3 + (5*a^4 - 62*a^2*b^2 - 4*b^4)*c^2 - 2*(5*a^5 + 17*a^3*b^2 - 38*a*b^4)*c)*sqrt(3))^(1/3) + 1/3*a + 1/3*c]
```

```

$$\begin{aligned} & ^3 + 4*a*b^2)*c^3 + (5*a^4 - 62*a^2*b^2 - 4*b^4)*c^2 - 2*(5*a^5 + 17*a^3*b^2 - 38 \\ & *a*b^4)*c)*sqrt(3))^{(1/3)} + (11/54*a^3 - 7/18*a^2*c + 1/3*b^2*c + 1/27*c^3 + 1/18 \\ & *(15*b^2 - c^2)*a + 1/18*sqrt(-5*a^6 - 6*a^4*b^2 + 11*a^2*b^4 - 5*a^2*c^4 - 32*b^6 + 2*(5*a^3 + 4*a*b^2)*c^3 + (5*a^4 - 62*a^2*b^2 - 4*b^4)*c^2 - 2*(5*a^5 + 17*a^3*b^2 - 38*a*b^4)*c)*sqrt(3))^{(1/3)} \end{aligned}$$

```

Whether those results are useful for us is a different question!

This last example demonstrates something I have found to be true quite generally when working with computer algebra systems: it can be difficult to find the right kind of problem for them. Initially, computer algebra systems seem like pure magic: so effortlessly do they perform tasks that took us *years* to learn (and that we still get wrong). But as we move from trivial to more realistic problems, it is often difficult to obtain results that one can actually use. All too often one ends up with a result like the one in the most recent example, which — although “correct” — simply does not shed much light on the problem we tried to solve! And before one manually simplifies an expression like the one above, one might be better off solving the entire problem with paper and pencil, because this way one can introduce new variables for frequently occurring terms or even make useful approximations as one goes along.

I think computer algebra systems are most useful in scenarios that require the generation of a *very* large number of terms (combinatorial problems), which in the end are evaluated (numerically or otherwise) entirely within the computer to yield the final result, without providing a “symbolic” solution in the classical sense at all. If these conditions are fulfilled, computer algebra systems enable one to tackle problems that would simply not be feasible with paper and pencil. At the same time, one maintains a greater level of accuracy, because numerical (finite-precision) methods, although still required to obtain a useful result, are employed only in the final stages of the calculation (rather than from the outset). Neither of these conditions are fulfilled for relatively straightforward ad-hoc symbolic manipulations — despite their immediate “magic” appeal, computer algebra systems are most useful as specialized tools for specialized tasks!

One final word about the SAGE project. As an open-source project, it leaves a strange impression. You first become aware of this when you attempt to download the binary distribution: it consists of a 500MB bundle, which unpacks to 2GB on your disk! When you investigate what is contained in this huge package, the answer turns out to be *everything*. SAGE ships with *all* of its dependencies. It ships with its own copy of all libraries it requires. It ships with its own copy of R. It ships with its own copy of Python! In short, it ships with its own copy of *everything*.

This is partially due to the well-known difficulties in making deeply numerical software portable, but is of course also an expression of the fact that SAGE is an “umbrella” project that tries to combine a wide range of otherwise independent projects. And while on the one hand I sincerely appreciate the straightforward pragmatism of this solution, it also feels heavy-handed and ultimately unsustainable. Personally, it makes me doubt the wisdom of the

entire “all under one roof” approach which is the whole purpose of SAGE: if this is what it takes, then we are probably on the wrong track. In other words, if it is not feasible to integrate different projects in a more organic way, then maybe those projects should remain independent, and the user should have the freedom to choose which ones to use. (It might be worth mentioning in this context that there are reports of hostile forks of projects who do not agree to being incorporated by SAGE in this way.)

8.8 Further Reading

There are two or three dozen books out there specifically on the topic of modeling, but I have been disappointed by most of them. Some of the more useful (from the elementary to the quite advanced) include:

- *How to Model It: Problem Solving for the Computer Age.* A. M. Starfield, K. A. Smith, A. L. Bleloch. Interaction Book Company. 1994.
Probably the best elementary introduction to modeling that I am aware of. Ten (fictitious) case studies are presented and discussed, each demonstrating a different modeling method. (Out of print, but available used.)
- *An Introduction to Mathematical Modeling.* Edward A. Bender. Dover Publications. 2000.
Short and idiosyncratic. A bit dated but still insightful.
- *Concepts of Mathematical Modeling.* Walter J. Meyer. Dover Publications. 2004.
This book is a general introduction to many of the topics required for mathematical modeling at an advanced beginner level. It feels more dated than it is, and the presentation is a bit pedestrian, but nevertheless, there is a lot of accessible, and most of all practical material here.
- *Modeling Complex Systems.* Nino Boccara. Springer. 2004.
This is a book by a physicist (not a mathematician, applied or otherwise) and it demonstrates how a *physicist* thinks about building models. The examples are rich, but mostly of theoretical interest. Conceptually advanced, mathematically not too difficult.
- *Practical Applied Mathematics.* Sam Howison. Cambridge University Press. 2005.
This is a very advanced book on applied mathematics, with a heavy emphasis on partial differential equations. However, the introductory chapters, though short, provide one of the most insightful (and witty) discussions of models, modeling, scaling arguments, and related topics I have seen.

The following two books are not about the process of modeling; instead, they provide examples of modeling in action (with a particular emphasis on scaling arguments).

- *The Simple Science of Flight*. Henk Tennekes. MIT Press, 2nd ed.. 2009.
This is a fascinating short book about the physics and engineering of flying at the “popular science” level. The author makes heavy use of scaling laws throughout — if you are interested in aviation, you will be interested in this book.
- *Scaling Concepts in Polymer Physics*. Pierre-Gilles de Gennes. Cornell University Press. 1979.
This is a research monograph on polymer physics, and probably not suitable for a general audience. But the treatment, which relies almost exclusively on a variety of scaling arguments, is almost elementary. Written by the master of the scaling models.

Chapter 9

Arguments from Probability Models

When modeling systems that exhibit some form of randomness, the challenge in the modeling process is to find a way to handle the resulting uncertainty: we don't know for sure what the system will do — there is a range of outcomes, each of which is more or less likely, according to some probability distribution. Occasionally, it is possible to actually work out the exact probabilities for all possible events, but this quickly becomes very difficult, if not impossible, as we go from simple (and possibly idealized systems) to real applications. We need to find ways to simplify life!

In this chapter, I want to take a look at some of the “standard” probability models that occur frequently in practical problems, and also describe some of their properties that make it possible to reason about them, without having to perform explicit calculations for all possible outcomes. We will see that we can reduce the behavior of many random systems to their “typical” outcome and a narrow range around that.

This is true for many situations, but not for all! Systems characterized by power law distribution functions can *not* be summarized by a narrow regime around a single value, and you will obtain very misleading (if not outright wrong) results if you try to handle such scenarios with standard methods. It is therefore important to recognize this kind of behavior and to choose appropriate techniques.

9.1 The Binomial Distribution and Bernoulli Trials

Bernoulli Trials are random trials that can have only two outcomes, commonly called Success and Failure. Success occurs with probability p , Failure occurs with probability $1 - p$. We further assume that successive trials are independent and that the probability parameter p stays constant throughout.

Although this description may sound unreasonably limiting, in fact many different processes can be expressed in terms of Bernoulli Trials — we just have to be sufficiently creative when defining the class of events that we consider “Successes”. A few examples:

- Define “heads” as “Success” in n successive tosses of a fair coin. In this case, $p = 1/2$.
- Using fair dice, we can define getting an “ace” as “Success” and all other outcomes as “Failure”. In this case $p = 1/6$.
- Equally well, we can define *not* getting an “ace” as “Success”. In this case $p = 5/6$.
- Consider an urn, containing b black and r red tokens. If we define drawing a red token as “Success”, then repeated drawings (with replacement!) from the urn constitute Bernoulli Trials with $p = r/(r + b)$.
- Toss two identical coins. Consider obtaining two “heads” as “Success”. Each toss of the two coins constitutes a Bernoulli Trial with $p = 1/4$.

As you can see, the restriction to a binary outcome is not really limiting: even a process that has naturally more than two possible outcomes (such as throwing dice) can be cast in terms of Bernoulli Trials if we restrict the definition of “Success” appropriately. Furthermore, as the last example shows, even combinations of events (such as tossing two coins, or — equivalently — two successive tosses of a single coin) can be expressed in terms of Bernoulli Trials.

The restricted nature of Bernoulli Trials makes it possible to derive some exact results (we’ll see some in a moment). More importantly, though, the abstraction forced on us by the limitations of Bernoulli Trials can help to develop simplified conceptual models of a random process.

9.1.1 Exact Results

The central formula for Bernoulli Trials gives the *probability to observe k successes in N trials with success probability p* , and is also known as the *Binomial Distribution* (also see figure 9.1):

$$P(k, N; p) = \binom{N}{k} p^k (1 - p)^{N-k}$$

This should make good sense: we need to obtain k successes, each occurring with probability p , and $N - k$ failures, each occurring with probability $1 - p$. The term

$$\binom{N}{k} = \frac{N!}{k!(N - k)!}$$

consisting of a *binomial coefficient* is combinatorial in nature: it gives the number of distinct arrangements for k successes and $N - k$ failures. (This is easy to

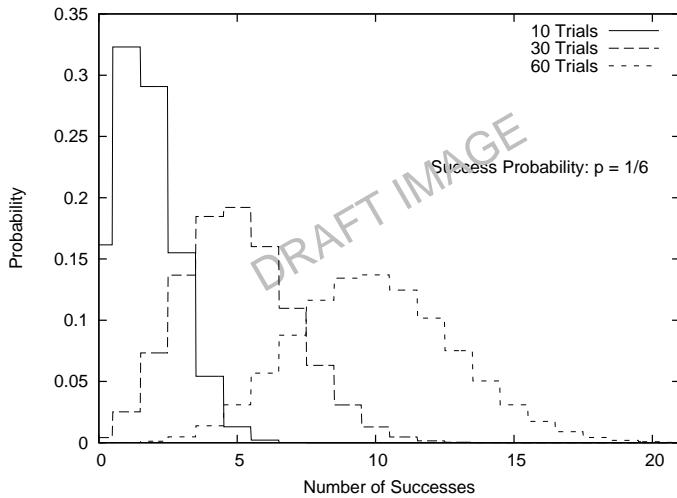


Figure 9.1: The Binomial Distribution: the probability to obtain k successes in N trials with success probability p .

see. There are $N!$ ways to arrange N *distinguishable* items: you have N choices for the first item, $N - 1$ choices for the second, and so on. However, the k successes are indistinguishable from each other, and the same is true for the $N - k$ failures. Therefore the total number of arrangements is reduced by the number of ways in which the successes can be rearranged, since all these rearrangements are identical to each other. With k successes, this means that $k!$ rearrangements are indistinguishable. Similarly for the $N - k$ failures.) Notice that the combinatorial factor does not depend on p .

This formula gives the probability to obtain a specific number k of successes. To find the expected number of successes μ in N Bernoulli Trials we need average it over all possible outcomes and find:

$$\begin{aligned}\mu &= \sum_k^N k P(k, N; p) \\ &= Np\end{aligned}$$

This should come as no surprise. We use this result intuitively whenever we say that we expect “about five heads in ten tosses of fair coin” ($N = 10, p = 1/2$), or that we expect to obtain “about ten aces in sixty tosses of a fair dice” ($N = 60, p = 1/6$).

Another result that can be worked out exactly is the standard deviation. The result is:

$$\sigma = \sqrt{Np(1 - p)}$$

The standard deviation gives us the range over which we expect the outcomes to vary. (For example, assume that we perform m experiments, each one consisting of N tosses of a fair coin. The expected number of successes in each experiment is Np , but of course we won't obtain exactly this number in each experiment. However, over the course of the m experiments, we expect to find the number of successes in the majority of them to lie between $Np - \sqrt{Np(1-p)}$ and $Np + \sqrt{Np(1-p)}$.)

Notice that σ is growing more slowly with the number of trials than μ ($\sigma \sim \sqrt{N}$ versus $\mu \sim N$): the relative width of the outcome distribution shrinks as we conduct more trials.

9.1.2 Using Bernoulli Trials To Develop Mean-Field Models

The primary reason why I am emphasizing the Bernoulli Trial concept so much is that it lends itself naturally to the development of "mean-field" models (see chapter 8). Let's say we try to develop a model to predict the staffing level required for a call center to deal with customer complaints. We know from experience that about one in every thousand orders will lead to a complaint (hence $p = 1/1000$). If we ship a million orders a day, we can use the Binomial Distribution to work out the probability to get 1, 2, 3, ..., 999999, 1000000 complaints a day and work out the required staffing levels accordingly — a daunting task! But in the spirit of mean-field theories, we can cut through the complexity by realizing that we will get "about $Np = 1000$ " complaints a day. So, rather than having to work with each possible outcome (and its associated probability), we limit our attention to a single *expected* outcome. (And we can now proceed to determine how many calls a single person can handle per day to find the required number of customer service people.) We can even go a step further and incorporate the uncertainty in the number of complaints, by considering the standard deviation. In this example it comes out to $\sqrt{Np(1-p)} \approx \sqrt{1000} \approx 30$. (Here I made use of the fact that $1-p$ is very close to 1 for the current value of p .) The spread is small compared to the expected number of calls, lending credibility to our initial approximation of replacing the full distribution with only its expected outcome! (This is an example for the observation we made earlier, that the width of the resulting distribution grows much more slowly with N than the expected value itself. As N gets larger, this effect becomes more drastic, meaning that mean-field theory gets *better* and more reliable the more urgently we need it! The tough cases can be situations where N is of moderate size — say, in the range of 10...100 — too large to work out all outcomes exactly, but not large enough to be safe working only with the expected values.)

Having seen this, we can apply similar reasoning to more general situations. For example, notice that the number of orders shipped each day is probably not going to be equal to exactly one million — instead, it will be a random quantity itself. So, by using $N = 1000000$, we have employed the same "mean-field" idea already. It should be easy to generalize to other situations from

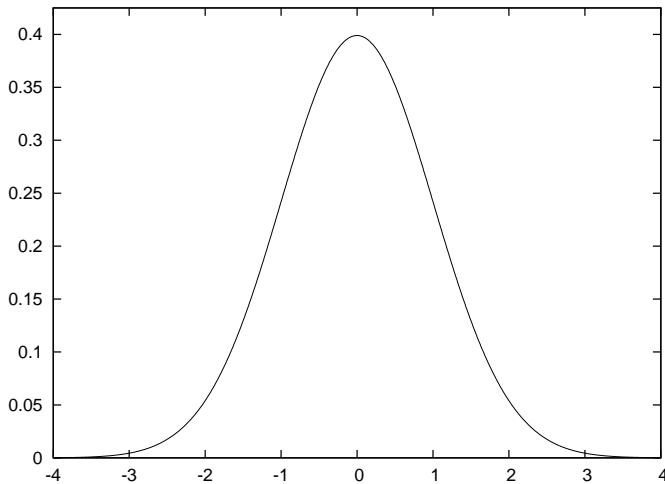


Figure 9.2: TBD

here.

9.2 The Gaussian Distribution and the Central Limit Theorem

Probably the most ubiquitous formula in all of probability theory and statistics is:

$$p(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

This is the formula for the Gaussian (or *Normal*) probability density. This is the proverbial “Bell Curve”. (See figure 9.2 and appendix B for additional detail.)

Two factors contribute to the elevated importance of the Gaussian distribution: on the foundational side, the Central Limit Theorem guarantees that the Gaussian distribution will arise naturally whenever we take averages (of almost anything). On the sheerly practical side, the fact that we can actually *do* most integrals involving the Gaussian explicitly means that expressions involving Gaussians make good building blocks for more complicated theories.

9.2.1 The Central Limit Theorem

Imagine you have a source of data points, which are distributed according to some common distribution. This could be numbers drawn from a uniform random-number generator, or prices of items in a store, or the body heights of a large group of people.

Now assume that you repeatedly take a sample of n elements from these sources: n random numbers, or n items from the store, or the measurements for n people, and form the total sum of the values. You can also divide by n to get the average. Notice that these sums (or averages) are random quantities themselves: since the points themselves are drawn from a random distribution, their sums will also be random numbers.

Note that we don't necessarily know the distributions that the original points come from, so it may seem as if it would be absolutely impossible to say anything about the distribution of their sums. Surprisingly, the opposite is true: we can make very precise statements about the form of the distribution according to which the sums are distributed. This is the content of the Central Limit Theorem.

The *Central Limit Theorem* states that the sums of a bunch of random quantities will be distributed according to a Gaussian distribution. This statement is not strictly true, it is only an approximation, with the quality of the approximation improving the more points we have in each sample (as n gets larger, the approximation gets better). In practice, though, the approximation is excellent even for quite moderate values of n .

This is an amazing statement, given that we made no assumptions whatsoever about the original distributions (I will qualify this in a moment): it seems as if we got something for nothing! On a moment's thought, however, this result should not be so surprising: if we take a single point from the original distribution, it may be large or it may be small — we don't know. But if we take many such points, then the highs and the lows will balance each other out “on average”, so we should not be too surprised that the distribution of the sums is a *smooth* distribution, with a *central peak*. It is, however, not obvious that this distribution should turn out to be the Gaussian specifically.

We can now state the Central Limit Theorem formally. Let $\{x_i\}$ be a sample of size n , with the following properties:

1. All x_n are mutually independent.
2. All x_n are drawn from a common distribution.
3. The mean μ and the standard deviation σ for the distribution of the individual data points x_i are finite.

Then the sample average $\frac{1}{n} \sum_i^n x_i$ is distributed according to a Gaussian with mean μ and standard deviation σ/\sqrt{n} . The approximation improves the larger the sample size n . In other words, the probability to find the value x for the sample mean $\frac{1}{n} \sum_i x_i$ becomes Gaussian as n gets large:

$$P \left(\frac{1}{n} \sum_i^n x_i = x \right) \rightarrow \sqrt{\frac{n}{2\pi}} \frac{1}{\sigma} e^{-\frac{1}{2} \left(\frac{x-\mu}{\sigma/\sqrt{n}} \right)^2}$$

Notice, again (as for the Binomial distribution), how the width of the resulting distribution of the average is smaller than the width of the original

distribution of the individual data points. This aspect of the Central Limit Theorem is the formal justification of the common use to “average out the noise”: no matter how widely the individual data points scatter, their averages will scatter less!

On the other hand, the reduction in width is not as fast as one might want: it is not reduced linearly with the number n of points in the sample, it is only reduced by \sqrt{n} . This means that if we take ten times as many points, the scatter is only reduced by $1/\sqrt{10} \approx 30$ percent. To reduce it by 10 percent, we need to increase the sample size hundred-fold. That’s a lot!

Finally, let’s take a look at the Central Limit Theorem in action. We draw samples from a uniform distribution, which takes on the values $1, 2, \dots, 6$ with equal probability — in other words, throws of a fair dice. This distribution has mean $\mu = 3.5$ (that’s pretty obvious) and standard deviation $\sigma = \sqrt{(6^2 - 1)/12} \approx 1.71$ (that’s not as obvious — but it’s not terribly hard to work it out, or you can look it up).

We now throw the dice a certain number of times and evaluate the average of the values that we observe. According to the Central Limit Theorem, these averages should be distributed according to a Gaussian distribution, which gets narrower as we increase the number of throws used to obtain an average. To see the distribution of values, we need to generate a histogram (see chapter 2). I use 1000 repeats to have enough data for a histogram. (Make sure you understand what is going on here: we throw the dice a certain number of times and calculate an average based on those throws. This entire process is then repeated 1000 times.)

The results are shown in figure 9.3. In the top left-hand corner, we throw the dice only once and consequentially form the “average” over only a single throw. You can see that all of the possible values are about equally likely: the distribution is uniform. In the top right-hand corner, we throw the dice *twice* every time and form the average over both throws. Already a central tendency in the distribution of the *average* of values can be observed! We then continue to make longer and longer averaging runs. (Also shown is the Gaussian distribution with the appropriately adjusted width: σ/\sqrt{n} , where n is the number of throws over which we form the average.)

I’d like to emphasize two observations in particular: first, note how quickly the central tendency becomes apparent — it only takes two or three throws to average over for a central peak to become established. Furthermore, note how well the properly scaled Gaussian distribution fits the observed histograms! This is the Central Limit Theorem in action.

9.2.2 The Central Term and The Tails

The most predominant feature of the Gaussian density function is the speed with which it falls to zero as $|x|$ becomes large. It is worth looking at some numbers to understand just *how* quickly it decays: for $x = 2$, the standard Gaussian with zero mean and unit variance is approximately $p(2, 0, 1) = 0.05\dots$. For

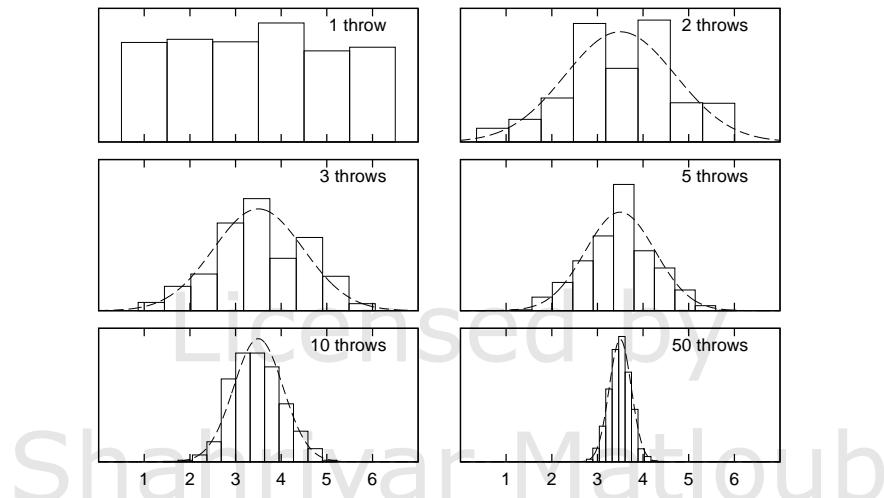


Figure 9.3: TBD

$x = 5$, it is already on the order of 10^{-6} , for $x = 10$ it about 10^{-22} , and not much further out, at $x = 15$, we find $p(15, 0, 1) \approx 10^{-50}$. One needs to keep this in perspective: the age of the universe is currently estimated to be about 15 billion years, which is about $4 \cdot 10^{17}$ seconds. So, even if we had made *a thousand drawings per second since the beginning of time*, we would not have found a value as large or larger than $x = 10$!

Although the Gaussian is defined for all x , its weight is so strongly concentrated within a finite, and actually quite small, interval (about $[-5, 5]$) that values outside this range will not occur. It is not just that only one in a million events will deviate from the mean by more than five standard deviations: the decline continues, so that less than one in 10^{22} events will deviate by more than 10 standard deviations. Large outliers are not just rare — they don't happen!

This is both the strength and the limitation of the Gaussian model: *if* the Gaussian model applies, then we know that all variation in the data will be relatively small and therefore "benign". At the same time, we know that for some systems large outliers do occur in practice — which means that for such systems the Gaussian model *and theories built on top of it* will not apply, giving bad guidance or outright wrong results. (We will come back to this.)

9.2.3 Why is the Gaussian so Useful?

It is the combination of these two properties that makes the Gaussian probability distribution so common and so useful: because of the Central Limit Theorem, we know that the Gaussian distribution will occur whenever we are dealing with averages. And because so much of its weight is concentrated

in the central region, we know that we can approximate basically any expression by concentrating only on the central region, while largely disregarding the tails.

As we will see in chapter 10 in more detail, the first of these two arguments has been put to good use by the creators of classical statistics: although we may not know anything about the distribution of the actual data points, the Central Limit Theorem enables us to make statements about their averages. Hence, if we concentrate on estimating the sample *average* of any quantity, we are on much firmer ground, theoretically. And it is impressive to see how classical statistics is able to make rigorous statements about the extent of confidence intervals for parameter estimates, using basically no information besides the data points themselves! I'd like to emphasize these two points again: through clever application of the Central Limit Theorem, classical statistics is able to give *rigorous* (not just intuitive) bounds on estimates, and it can do so without requiring detailed knowledge or making additional assumptions about the system under investigation. This is a remarkable achievement!

The price we pay for this rigor is that we loose a lot of the richness of the original data set: the distribution of points has been boiled down to a single number — the average.

The second argument is not so relevant from a conceptual point, but is of course of primary practical importance: we can actually do many integrals involving Gaussians, either exactly or in very good approximation. In fact, the Gaussian is so convenient in this regard that it is often the first choice when there is a need for an integration kernel (we have already seen examples of this in chapter 2, in the context of kernel density estimates, and in chapter 4, when we discussed the smoothing of a time series).

The basic idea goes like this: we want to evaluate an integral of the form:

$$\int f(x)e^{-x^2/2} dx$$

We know that the Gaussian is peaked around $x = 0$, so that only points nearby will contribute significantly to the value of the integral. Because of this, we can now expand $f(x)$ in a power series for small x . Even if this expansion is no good for large x the result will not be affected significantly, since those points are suppressed by the Gaussian. We now end up with a series of integrals of the form

$$a_n \int x^n e^{-x^2/2} dx$$

(a_n being the expansion coefficient from the expansion of $f(x)$), which can be performed exactly.

We can push this idea even further. Assume that the kernel is not exactly Gaussian, but still strongly peaked:

$$\int f(x)e^{-g(x)} dx$$

where the function $g(x)$ has a minimum somewhere (otherwise the kernel would not have a peak at all). We can now expand $g(x)$ into a Taylor series around its minimum (let's assume it is at $x = 0$), retaining only the first two terms: $g(x) \approx g(0) +$

$g''(0)x^2/2 + \dots$ — the linear term vanishes, because the first derivative g' must be zero at a minimum. Keep in mind that the first term in this expansion is a constant not depending on x , we have transformed the original integral to one of Gaussian type:

$$e^{-g(0)} \int f(x)e^{-g''(0)x^2/2} dx$$

which we already know how to solve.

This method goes by the name of *Laplace's Method*. (Not to be confused with "Gaussian Integration", which is something else entirely.)

9.2.4 Beware: The World is Not Normal!

Given that the Central Limit Theorem is a rigorously proven theorem, what could possibly go wrong? After all, the Gaussian distribution guarantees the absence of outliers, doesn't it? Yet, we all know that unexpected events *do* happen.

There are two things that can go wrong with the discussion so far:

- The Central Limit Theorem only applies to sums or averages of random quantities, not necessarily to the random quantities themselves. The distribution of individual data points may be quite different, and if we want to reason about individual events (rather than about an aggregate such as their average), we may need different methods.

For example, although the *average* number of items in a shipment may be Gaussian distributed around a typical value of three items per shipment, nobody says that actual distribution of items per shipment will follow the same distribution. In fact, it will probably be geometrical, with shipments containing only a single item being much more common than any other shipment size.

- More importantly, the Central Limit Theorem *may not apply*. Remember the three conditions I listed as requirements for the Central Limit Theorem to hold? Individual events had to be independent, follow the same distribution, and had finite mean and standard deviation. As it turns out, the first and second of these conditions can be weakened (meaning that individual events can be somewhat correlated and being drawn from slightly different distributions), but the third condition can *not*: individual events *must* be drawn from a distribution of finite width.

Now, that may seem like a minor matter: surely, all distributions occurring in practice would be of finite width, won't they? As it turns out, the answer is *NO!*. Apparently "pathological" distributions of this kind are much more common in real-life than one might expect. These are distributions following *power law* behavior, and they are the topic of the next section.

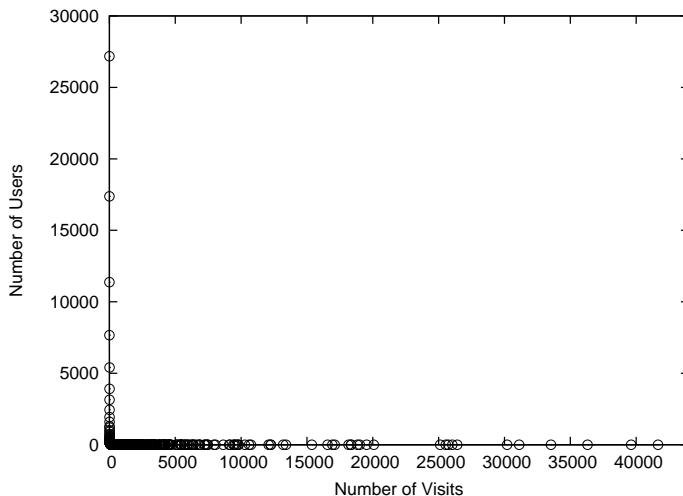


Figure 9.4: TBD

9.3 Power Law Distributions and non-Normal Statistics

Let's start with an example. Figure 9.4 shows a histogram for the number of visits per person that a sample of visitors made to a certain website over one month. Two things stand out: the huge number of people who made a handful of visits (less than about 5 or 6), and (on the other extreme) the huge number of visits that a few people made. (The heaviest user made 41661 visits: that's about one per minute over the course of the month — probably a bot or monitor of some sort.)

This distribution looks nothing like the “benign” case in figure 9.2. The distribution in figure 9.4 is not just skewed — it would be no exaggeration to say that it consists *entirely* of outliers! Ironically, the “average” number of visits per person (formed naively, by summing the visits and dividing by the number of unique visitors) comes out to 26 visits per person: this number is clearly not representative for anything: not for the huge majority of light users on the left-hand side of the graph, nor for the small group of heavy users to the right. (The standard deviation is ± 437 (!) — a clear indication that something is not right, given that the mean is 26 and the number of visits must be positive.)

This kind of behavior is typical for distributions with so-called *fat* or *heavy tails*. In contrast to systems ruled by a Gaussian distribution or another distribution with short tails, data values are not effectively limited to a narrow domain. Instead, we can find a non-negligible fraction of data points that are very far away from the majority of points.

Mathematically speaking, a distribution is heavy-tailed if it falls to zero

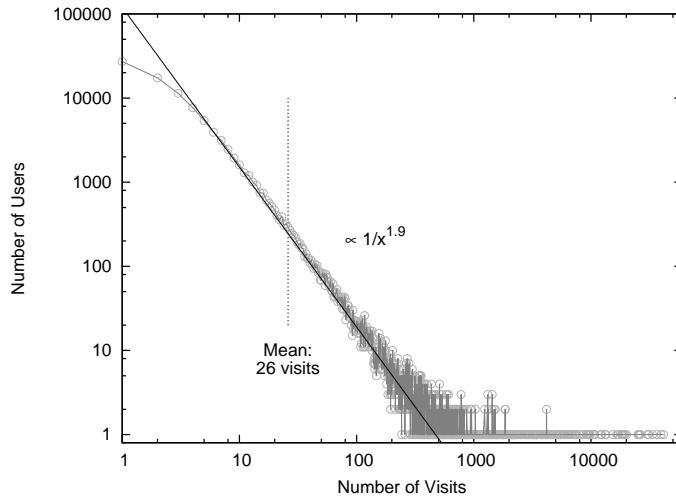


Figure 9.5: TBD

much slower than exponential. Power laws, that is functions that behave as $\sim 1/x^\beta$ for some exponent $\beta > 0$, are usually used to describe such behavior.

From chapter 3 we know how to recognize power-laws: data points falling onto a straight line on a double-logarithmic plot. A double logarithmic plot of the data from figure 9.4 is shown in figure 9.5, and we see that eventually (for more than 5 visits per person) the data indeed follows a power law (approximately $x^{-1.9}$). On the lefthand side of figure 9.5 (that is, for few visits per person), the behavior is different. (We will come back to this point.)

Power law distributions like the one describing the data set in figures 9.4 and 9.5 are surprisingly common. They have been observed in a number of (often very colorful) different areas: the frequency with which words are used in texts; the size of earthquakes and file sizes; the copies of books sold and the intensity of wars; the sizes of sand particles and solar flares; the population of cities and the distribution of wealth. Power law distributions go by different names in different contexts — you will find them being referred to as “Zipf” or “Pareto” distributions, but the mathematical structure is always the same. The term “power law distribution” is probably the most widely accepted, general term for this kind of heavy-tailed distribution.

Wherever they were found, power law distributions were met with surprise and (usually) consternation. The reason is that they have some unexpected and counter-intuitive properties:

- Observations span a wide range of values, often many orders of magnitude.
- There is no typical scale or value, which can be used to summarize the

distribution of points.

- The distribution is very skewed, with many data points at the low end, and few (but not negligibly few) data points at *very* high values.
- Expectation values often depend on the sample size: if you take the average over a sample of n points, you may find one value, but if you take the average over $2n$ or $10n$ data points, you may find a significantly greater value. (This is in marked contrast to most other distributions, where the quality of the average gets better when it is based on more points. Not so for power law distributions!)

It is the last item that is the most disturbing: didn't the Central Limit Theorem tell us that scatter of the average was always reduced by a factor of $1/\sqrt{n}$ as the sample size increases? Yes, but remember the caveat at the end of the last section: the Central Limit Theorem only applies for distributions that have finite mean and standard deviation. For power law distributions, this condition is not necessarily fulfilled, and hence the Central Limit Theorem does *not* apply!

The importance of this fact cannot be overstated. Not only goes much of our intuition out the window, but most of statistical theory, too! For the most part, distributions without expectations are simply not treated by standard probability theory and statistics.¹

9.3.1 Working with Power-Law Distributions

So, what should you do if you encounter a situation described by a power law distribution? The most important thing is to *stop using classical methods*. In particular, the “mean-field” approach, of replacing the distribution by its mean, is no longer applicable and will give misleading or incorrect results.

From a practical point of view, you can try segmenting the data (and, by implication, the system) into different groups: the majority of data points at small values (on the left-hand side in figure 9.5), the set of data points in the “tail” (for relatively large values), and possibly even a group of data points making up the intermediate regime. Each such group is now more homogeneous, so that standard methods apply. You will need insight into the business domain of the data and exercise a certain amount of discretion when determining where to make those cuts, since the data itself will not yield a natural “scale” or other quantity that could be used for this purpose.

There is one more practical point that you should be aware of when working with power-law distributions: the form $\sim 1/x^\beta$ is only valid “asymptotically” for large x . For small x , this rule has to be supplemented, since obviously it cannot hold for $x \rightarrow 0$ (we cannot divide by zero!). There are several ways to augment the original form near $x = 0$: We can either impose a minimum

¹The comment on page 48 (out of 440!) of Larry Wasserman’s (excellent!) “All of Statistics” is typical: “From now on, whenever we discuss expectations, we implicitly assume that they exist.”

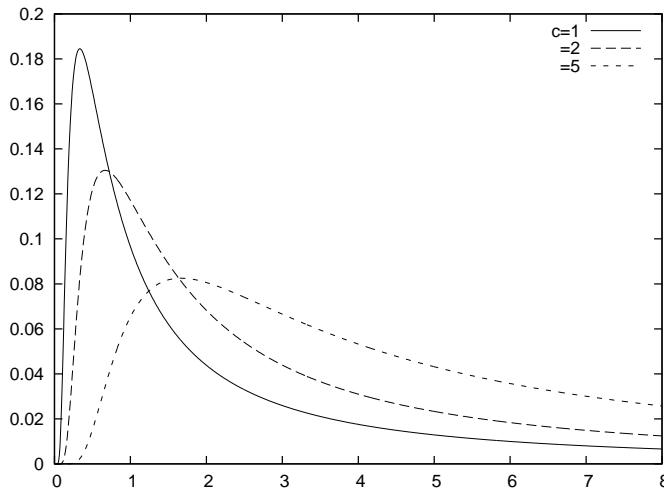


Figure 9.6: TBD

value x_{\min} of x , and only consider the distribution for values larger than this. That is often a reasonable approach, since such a minimum value may exist naturally: for example there is an obvious “minimum” number of pages that a website visitor can view and still be considered a “visitor” (namely: one page). Similarly for the population of a city and the copies of books sold — all are limited on the left by $x_{\min} = 1$. Alternatively, the behavior of the observed distribution may be different for small values. Look again at figure 9.5: for values less than about 5, the curve deviates from the power law behavior that we find elsewhere.

Depending on the shape that we require near zero, we can modify the original rule in different ways. Two examples stand out: if we want a flat peak for $x = 0$, we can try a form like $\sim 1/(a + x^\beta)$ for some $a > 0$, and if we require a peak at a non-zero location, we can use a distribution like $\sim \exp(-C/x)/x^\beta$ (also see figure 9.6). For specific values of β , two distributions of this kind have special names:

$$\frac{1}{\pi} \frac{1}{1+x^2} \quad \text{Cauchy Distribution}$$

$$\sqrt{\frac{c}{2\pi}} \frac{e^{-c/2x}}{x^{3/2}} \quad \text{Lévy Distribution}$$

9.3.2 Theory: Distributions with Infinite Expectation Values

The *expectation value* $E(f)$ of a function $f(x)$, which in turn depends on some random quantity x , is nothing but the weighted average of that function, where we use the probability density $p(x)$ of x as the weight function.

$$E(f) = \int f(x)p(x) dx$$

Of particular importance are the expectation values for simple powers of the variable x , the so called *moments* of the distribution.

$$\begin{aligned} E(1) &= \int p(x) dx && \text{(must always equal 1)} \\ E(x) &= \int x p(x) dx && \text{the mean or first moment} \\ E(x^2) &= \int x^2 p(x) dx && \text{the second moment} \end{aligned}$$

The first expression must always be equal to 1, because we expect $p(x)$ to be properly normalized. The second is the familiar mean, as the weighted average of x . The last expression is used in the definition of the standard deviation:

$$\sigma = \sqrt{E(x^2) - E(x)^2}$$

For power-law distributions, which behave as $\sim 1/x^\beta$ with $\beta > 1$ for large x , some of these integrals may not converge — in this case, the corresponding moment “does not exist”. Consider the k th moment (C is the normalization constant $C = E(1) = \int p(x) dx$):

$$\begin{aligned} E(x^k) &= C \int_0^\infty x^k \frac{1}{x^\beta} dx \\ &= C \int_0^\infty \frac{1}{x^{\beta-k}} dx \end{aligned}$$

Unless $\beta - k > 1$, this integral does not converge at the upper limit of integration. (I assume that the integral is proper at the lower limit of integration, through a lower cut-off x_{\min} , or another one of the methods discussed earlier.) In particular, if $\beta < 2$, the mean (and all higher moments) do not exist, if $\beta < 3$, the standard deviation does not exist.

We need to understand that this is an analytical result — it tells us that the distribution is ill-behaved, and that for instance the Central Limit Theorem does not apply in this case. Of course, for any *finite* sample of n data points drawn from such a distribution, the mean (or other moment) will be perfectly finite. But these analytical results are warning us that if we continue drawing additional data points from the distribution, their average (or other moment) will not settle down: it will grow as the number of data points in the sample grows. Any summary statistic calculated from a finite sample of points will therefore not be a good estimator for the true (that is: infinite) value of that statistic. This poses an obvious problem, since of course all practical samples contain only a finite number of points.

Basically, power law distributions don't have any parameters that could, nor need be, estimated, except for the exponent, which we know how to obtain from a double-logarithmic plot. There is also a maximum likelihood estimator for the exponent:

$$\beta = 1 - \frac{n}{\sum_i^n \log \frac{x_i}{x_0}}$$

where x_0 is the smallest value of x for which the asymptotic power-law behavior holds.

9.3.3 Where To Go From Here

If you want to dig deeper into the theory of heavy-tail phenomena, you will find that it is a mess. There are two reasons for that: on the one hand, the material is technically hard (since one has to make do without two standard tools: expectation values and the Central Limit Theorem), so that few simple, substantial, powerful results have been obtained — a fact, which is often being covered up through excessive formalism. On the other hand, the “colorful” and multi-disciplinary context in which power law distributions are found has led to much confusion. Similar results are being discovered and re-discovered in various fields, with each field imposing its own terminology and methodology, obscuring the mathematical commonalities.

The unexpected and often almost paradoxical consequences of power law behavior also seem to demand an explanation for *why* such distributions occur in practice, and whether they might all be expressions of some common mechanisms. Quite a few theories have been proposed to this end, but none of them has found widespread acceptance or proved particularly useful in predicting new phenomena — occasionally grandiose claims to the contrary notwithstanding.

At this point, I think it is fair to say that we don't understand heavy-tail phenomena: not when and why they occur, nor how to handle them if they do.

9.4 Other Distributions

There are some other distributions that describe common scenarios and that you should be aware of. Some of the most important (or at least most frequently used) ones are described in this section.

9.4.1 Geometric Distribution

The geometric distribution (see figure 9.7):

$$p(k, p) = p(1 - p)^{k-1} \quad \text{with } k = 1, 2, 3, \dots$$

is a special case of the binomial distribution. It can be regarded as the probability to obtain the first success at the k -th trial (that is, after observing $k - 1$ failures). Note that there is only a single arrangement of events for this outcome, hence the combinatorial factor is equal to one. It has mean $\mu = 1/p$ and standard deviation $\sigma = \sqrt{1 - p}/p$.

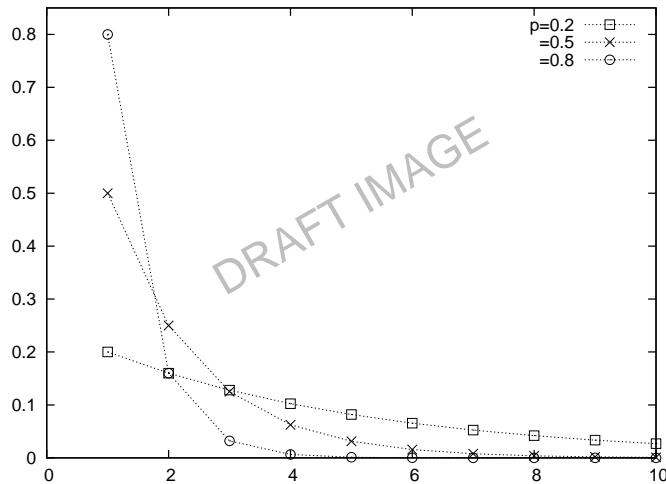


Figure 9.7: TBD

9.4.2 Poisson Distribution

The binomial distribution gives us the probability to observe exactly k events in n distinct trials. By contrast, the Poisson distribution describes the probability to find k events during some continuous observation *interval* of known length. Rather than being characterized by a probability parameter and a number of trials (as is the case for the binomial distribution), the Poisson distribution is characterized by a *rate* λ and an *interval length* t .

The Poisson distribution $p(k, t, \lambda)$ provides the probability to observe exactly k events during an interval of length t , when the rate at which events occur is λ (figure 9.8):

$$p(k, t, \lambda) = \frac{(\lambda t)^k}{k!} e^{-\lambda t}$$

Because t and λ only occur together, this expression is often written in a two-parameter form as: $p(k, \nu) = \exp(-\nu)\nu^k/k!$. Also note that the term $\exp(-\lambda t)$ does not depend on k at all — it is merely there as a normalization factor. All the action is in the fractional part of the equation.

Let's look at an example. Assume that phone calls arrive at a call center at a "rate" of 15 calls per hour (that is: $\lambda = 0.25$ calls/minute). Then the Poisson distribution $p(k, 1, 0.25)$ will give us the probability that $k = 0, 1, 2, \dots$ calls will arrive in any given minute. But we can also use it to calculate the probability that k calls will arrive during any 5 minute time period: $p(k, 5, 0.25)$. Note that in this context it does not make sense to speak of independent trials — time passes continuously, and the expected number of events depends on the length of the observation interval.

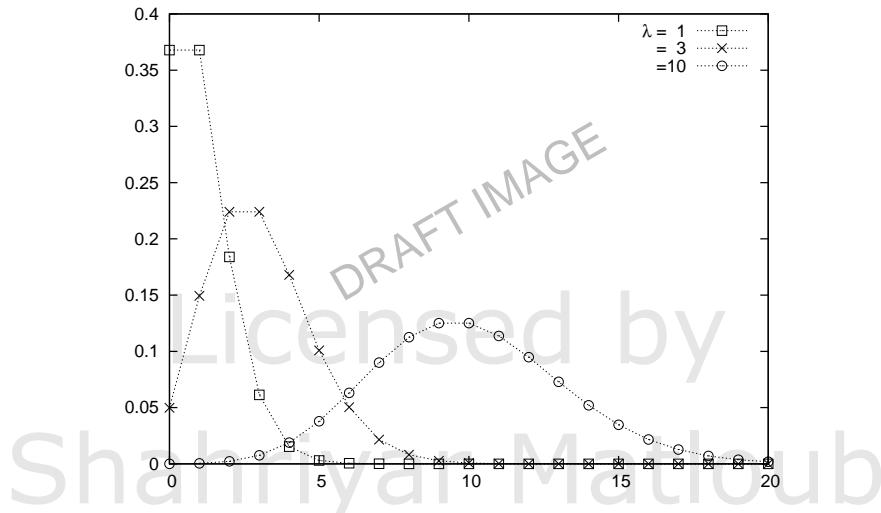


Figure 9.8: TBD

We can collect a few results: mean μ and standard deviation σ for the Poisson distribution are given by:

$$\begin{aligned}\mu &= \lambda t \\ \sigma &= \sqrt{\lambda t}\end{aligned}$$

Notice that only a single parameter (namely λt) controls both the location and the width of the distribution! For large λ the Poisson distribution approaches a Gaussian distribution with $\mu = \lambda$ and $\sigma = \sqrt{\lambda}$. Only for quite small values of λ are the differences notable (say, for $\lambda < 20$).

Conversely, to estimate the parameter λ from observations, we divide the number k of events observed by the length t of the observation period: $\lambda = k/t$. Notice that when evaluating the formula for the Poisson distribution, the rate λ and the length t of the interval of interest must be of compatible units. To find the probability of k calls over 6 minutes in the call center example above, we can either use $t = 6$ minutes and $\lambda = 0.25$ calls per minute or $t = 0.1$ hours and $\lambda = 15$ calls per hour, but we cannot mix them. (Also note that $6 \cdot 0.25 = 0.1 \cdot 15 = 1.5$ as it should.)

The Poisson distribution models processes in which discrete events occur independently and at a constant rate: calls to a call center, misprints in a manuscript, traffic accidents, and so on. However, you have to be careful: it only applies if you can identify a rate at which events occur *and* if you are interested specifically in the number of events occurring during intervals of varying length. (You can not assume every histogram to follow a Poisson distribution just because “we are counting events”.)

9.4.3 Log-Normal Distribution

Some quantities are inherently asymmetrical. Consider, for example, the time it will take people to complete a certain task: because everyone is different, we expect a distribution of values. However, all values are necessarily positive (since times can not be negative). Moreover, we can expect a particular shape of the distribution: there will be some minimum time that nobody can beat, then a small group of very fast champions, a peak at the most typical completion time, and finally a long tail of stragglers. Clearly, such a distribution will not be well described by a Gaussian, which is defined for both positive and negative values of x , is symmetric, and has short tails!

The log-normal distribution is an example of an asymmetric distribution suitable for such situations. It is related to the Gaussian: a quantity follows the log-normal distribution if its logarithm is distributed according to a Gaussian.

The probability density for the log-normal distribution looks like this:

$$p(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma x} e^{-\frac{1}{2}\left(\frac{\log(x/\mu)}{\sigma}\right)^2}$$

(The additional factor of x in the denominator stems from the Jacobian in the change of variables from x to $\log x$.) In the literature, you will often find it written slightly differently:

$$p(x; \tilde{\mu}, \sigma) = \frac{1}{\sqrt{2\pi}\sigma x} e^{-\frac{1}{2}\left(\frac{\log(x)-\tilde{\mu}}{\sigma}\right)^2}$$

which is the same, once you realize that $\log(x/\mu) = \log(x) - \log(\mu)$ and you make the identification $\tilde{\mu} = \log(\mu)$. The first form is much better, because it makes it clear that μ is the *typical scale* of the problem. It also ensures that the argument of the logarithm is dimensionless (as it must be).

Figure 9.9 shows the log-normal distribution for a few different values of σ . The parameter σ controls the overall “shape” of the curve, whereas the parameter μ controls its “scale”. In general, it can be difficult to predict what the curve will look like for different values of the parameters, but below we collect some results (the *mode* is the position of the peak, by the way):

$$\begin{aligned} \text{mode: } & \mu e^{-\sigma^2} \\ \text{mean: } & \mu e^{\frac{\sigma^2}{2}} \\ \text{standard deviation: } & \mu \sqrt{e^{\sigma^2} (e^{\sigma^2} - 1)} \end{aligned}$$

Values for the parameters can be estimated as follows from a data set:

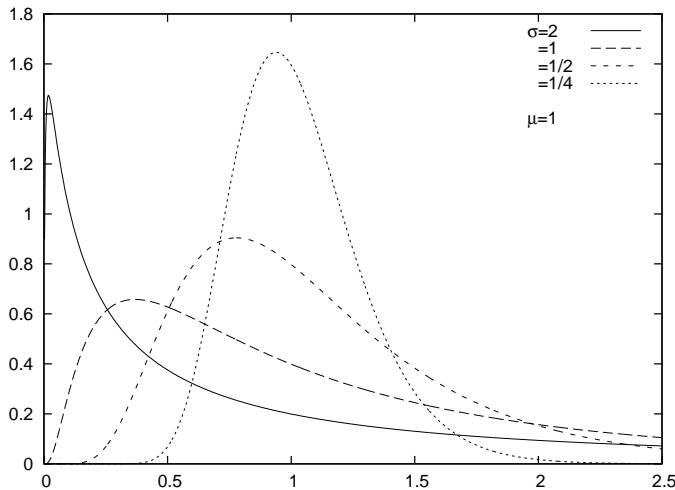


Figure 9.9: TBD

$$\mu = e^{\frac{1}{n} \sum_{i=1}^n \log x_i}$$

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n \left(\log \frac{x_i}{\mu} \right)^2}$$

Having seen this, I will mention that the log-normal distribution can be fickle to use in practice: not all asymmetric point distributions are described well by a log-normal distribution, and you may not be able to obtain a good fit for your data using a log-normal. In particular for truly heavy-tail phenomena, you will still need a power-law distribution after all. Also keep in mind that the log-normal distribution approaches the Gaussian σ becomes small compared to μ ($\sigma/\mu \ll 1$), at which point it becomes easier to work with the familiar Gaussian directly.

The log-normal distribution is important, however, as an example of a standard statistical distribution that provides an alternative to the Gaussian model for situations that require an asymmetrical distribution.

9.4.4 Special Purpose Distributions

Many additional distributions have been defined and studied. Some, such as the Gamma distribution, are mostly of theoretical importance, while others, such as the Chi-Square-, t-, and F-distributions are at the core of classical, frequentist statistics (we will encounter them again in chapter 10). Yet others

have been developed to model specific scenarios occurring in practical applications, in particular in reliability engineering, where the objective is to make predictions about likely failure rates and survival times.

I just want to mention in passing a couple of names that you may encounter. The *Weibull* distribution is used to express the probability that a device will fail after a certain time. Like the log-normal distribution, it depends on both a shape and a scale parameter. Depending on the value of the shape parameter, the Weibull distribution can be used to model different failure modes: from “infant mortality” scenarios, where devices are more likely to fail early but the failure rate declines over time as defective items disappear from the population, to “fatigue death” scenarios where the failure rate rises over time as items age.

Yet another set of distributions goes by the name of *Extreme Value* or *Gumbel* distributions. They can be used to obtain the probability for the smallest (or largest) value of some random quantity to be of a certain size. In other words, they answer the question: what is the probability that the largest element in a set of random numbers is precisely x ?

Quite intentionally, I don’t give formulas for these distributions here — these are rather advanced and specialized tools, and if you want to use them, you will need to consult the appropriate references. However, the important point to take away here is that for many typical scenarios involving random quantities, people have developed explicit models and studied their properties: a little research may well turn up a solution to whatever your current problem is!

9.5 Case Study: Unique Visitors over Time

To put some of the ideas that we have introduced in the last two chapters into practice, we will study an example that is a bit more involved. We will start with a probabilistic argument and use it to develop a mean-field model, which in turn will lead to a differential equation that we proceed to solve for our final answer. The reason for including this example here is to demonstrate how all the different ideas we have been introducing in the last few chapter can be made to fit together to tackle more complicated problems.

Imagine you are running a website. Users visit this website every day of the month at a rate that is roughly constant. We can also assume that we are able to track the identity of these users (through a cookie or something like that). By studying those cookies, we can see that some users visit the site only once in any given month, and others visit it several times. What we are interested in is the number of *unique users* for the month, and in particular how this number develops over the course of the month. (The number of unique visitors is a key metric in Internet advertising, for instance.)

The essential difficulty is that some users visit several times during the month, hence the number of unique visitors is smaller than the total number of visitors. Furthermore, we will observe a “saturation effect”: on the first day, basically every user is new. But on the last day of the month, we can expect to have seen many of the visitors earlier in the month already.

What we would like to develop is some understanding for the number of unique

visitors we can expect for each day of the month (for instance to monitor whether we are on track to meet some monthly goal for the number of unique visitors). To make progress, we need to develop a model.

To see more clearly, we consider the following idealization, which is equivalent to the original problem: consider an urn, containing N identical tokens (total number of potential visitors). At each turn (every day), we draw k tokens randomly from the urn (average number of visitors per day). We then mark all of the drawn tokens to indicate that we have “seen” them, and place them back into the urn. Then we repeat the cycle.

Because at each turn we mark all unmarked tokens from the random sample drawn at this turn, the number of marked tokens in the urn will increase over time. Because each token is marked at most once, the number of marked tokens in the urn at the end of the month is the number of unique visitors that have visited during that time period.

Phrased this way, the process can be modeled as a sequence of Bernoulli Trials. We define drawing an already marked token as “Success”. Because the number of marked tokens in the urn is increasing, the success probability p will change over time. The relevant variables are:

N	total number of tokens in urn
k	number of tokens drawn at each turn
$m(t)$	number of already marked tokens drawn at turn t
$n(t)$	total number of marked tokens in urn at time t
$p(t) = \frac{n(t)}{N}$	probability to draw an already marked token at turn t

Each day consists of a new Bernoulli Trial, in which k tokens are drawn from the urn. However, because the number of marked tokens in the urn increases every day, the probability $p(t)$ is different every day.

On day t , we have $n(t)$ marked tokens in the urn. We now draw k tokens, of which we expect $m(t) = kp(t)$ to be marked (“successes”). This is simply an application of the basic result for the expectation value of Bernoulli Trials, using the current value for the probability. (Working with the expectation value in this way constitutes a mean-field approximation.)

The number of unmarked tokens in the current drawing is:

$$k - m(t) = k - kp(t) = k(1 - p(t))$$

We now mark these tokens and place them back into the urn. That means that the number of marked tokens in the urn grows by $k(1 - p(t))$:

$$n(t + 1) = n(t) + k(1 - p(t))$$

This equation simply expresses the fact that the new number of marked tokens $n(t + 1)$ consists of the previous number of marked tokens $n(t)$ plus the newly marked tokens $k(1 - p(t))$.

We can now divide both sides by N (the total number of tokens). Recalling that $p(t) = n(t)/N$, we write:

$$p(t + 1) = p(t) + f(1 - p(t)) \quad \text{with } f = \frac{k}{N}$$

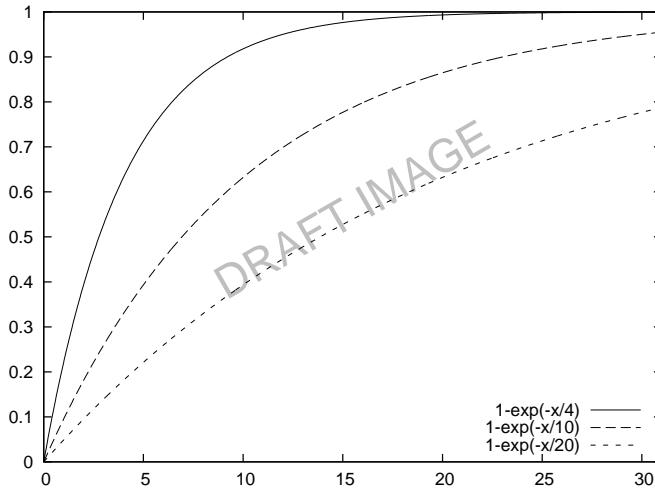


Figure 9.10: TBD

This is a recurrence relation for $p(t)$. We rewrite it as:

$$p(t+1) - p(t) = f(1 - p(t))$$

In the continuum limit, we replace the difference between the “new” and the “old” values by the *derivative* at time t , which turns the recurrence relation into a more convenient differential equation:

$$\frac{dp(t)}{dt} = f(1 - p(t))$$

with initial condition: $p(t = 0) = 0$ (because initially there are no marked tokens in the urn).

This differential equation has the solution:

$$p(t) = 1 - \exp(-ft)$$

Figure 9.10 shows $p(t)$ for various values of the parameter f . (The parameter f has an obvious interpretation as size of each drawing, expressed as a fraction of the total number of tokens in the urn.)

This is the result that we have been looking for. Remember that $p(t) = n(t)/N$, hence the probability is directly proportional to the number of unique visitors so far. We can rewrite it more explicitly as:

$$n(t) = N(1 - e^{-\frac{k}{N}t})$$

In this form, the equation gives us the number of unique visitors for the month-to-date, for each day of the month. There is only one unknown parameter, namely N , the total number of *potential* visitors. (We know k , the average number of total visitors per day, because this number is immediately available from the web server logs.) We can now try to fit one or two months worth of data to this formula to obtain a value for N . Once

we have determined N , the formula predicts the expected number of unique visitors for each day of the month. We can use this information to track whether the actual number of unique visitors for the current month is above or below expectations, and respond accordingly.

This little example is typical for a lot of modeling: we start with a real problem in a specific situation. To make head-way, we recast it in an idealized format, which tries to retain only the most relevant information. (In this example, we did this as we mapped the original problem to an idealized urn model.) Expressing things in terms of an idealized model helps to recognize the problem as one that we know how to solve. (Urn models have been studied extensively; in this example, we could identify it with Bernoulli Trials, which we know how to handle.) To find a solution, we often need to make actual approximations, in addition to the abstraction from the problem domain to an idealized model. (Working with the expectation value only constituted such an approximation to make the problem tractable; replacing the recurrence relation with a differential equation was another.) Finally, we end up with a “model”, involving some unknown parameters. If we are mostly interested in developing conceptual understanding, then we don’t need to go any further, since we can read off the behavior of the model directly from the formula. However, if we actually want to make numerical predictions, we need to find numerical values for those parameters: this is usually done by fitting the model to some already available data. (We should also try to validate the model, to see whether it gives a good “fit”: check the discussion in chapter 3 on examining residuals, for instance.)

Lastly, I should point out that the model in this example is simplified — as models usually are. The most critical simplification (which would most likely *not* be correct in a real application) is that every token in the urn has the same probability to be drawn at each turn. If we look at actual visitors, by contrast, we will find that some of them are much more likely to visit more frequently, while others are less likely to visit. Another simplification is that we assumed the total number of potential visitors to be constant — if we have a website that sees significant growth from one month to the next, this assumption may not be correct either. You may want to try and build an improved model that takes these (and other?) considerations into account. (The first one in particular is not easy — in fact, if you succeed, let me know how you did it!)

9.6 Workshop: Power Law Distributions

The crazy effects of power law distributions have to be seen to be believed. In this workshop, we are going to generate (random) data points distributed according to a power law distribution and begin to study their properties.

First question: how do you actually generate non-uniformly distributed random numbers on a computer? A random generator that produces uniformly distributed numbers is available in almost all programming environments, but generating random numbers distributed according to some other distribution requires a little bit more work. There are different ways of going about it; some are specific to certain distributions only, others are designed for speed in particular applications. We’ll discuss a simple method that works generally for distributions which are analytically known.

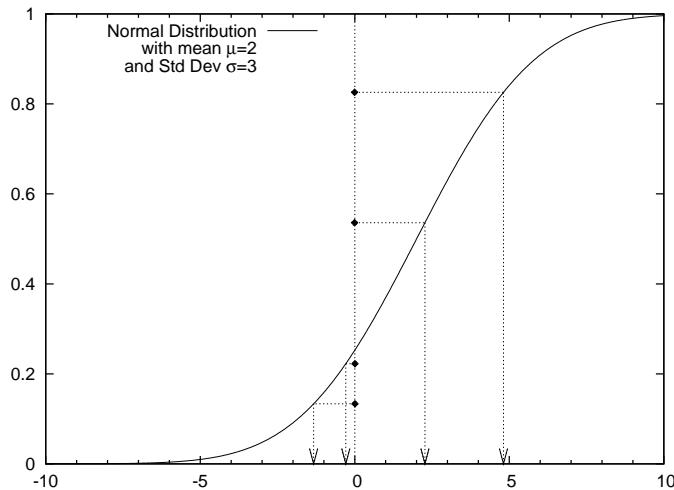


Figure 9.11: TBD

The starting point is the cumulative distribution function for the distribution in question. By definition, it is strictly monotonic and takes on values in the interval $[0, 1]$. If we now generate uniformly distributed numbers between 0 and 1, we can find the locations at which the cumulative distribution function assumes these values. These points will be distributed according to the desired distribution (see figure 9.11).

(A good way to think about this is as follows: imagine you distribute n points *uniformly* on the interval $[0, 1]$ and find the corresponding locations at which the cumulative distribution function assumes these values. These locations are spaced according to the distribution in question — after all, by construction, the probability grows by the same amount between successive locations. Now use points that are randomly distributed, rather than uniformly, and you end up with random points, distributed according to the desired distribution.)

For power law distributions, we can work out the cumulative distribution function and its inverse easily. Let the probability density $p(x)$ be:

$$p(x) = \frac{\alpha}{x^{\alpha+1}} \quad x \geq 1, \alpha > 0$$

This particular form is known as the the standard form of the Pareto distribution. It is valid for values of x greater than 1. (Values less than 1 have zero probability to occur.) The parameter α is the “shape parameter” and must be greater than zero, otherwise the probability is not normalizable. (This is a different convention than the one we used earlier: $\beta = 1 + \alpha$.)

We can work out the cumulative distribution function $P(x)$:

$$\begin{aligned} P(x) &= y = \int_1^x p(t) dt \\ &= 1 - \frac{1}{x^\alpha} \end{aligned}$$

This expression can be inverted easily to give:

$$x = \frac{1}{(1-y)^{1/\alpha}}$$

If we now use uniformly distributed random values for y , then the values for x will be distributed according to the Pareto distribution that we started with. (For other distributions, such as the Gaussian, inverting the expression for the cumulative distribution function is often harder, and you may have to find a numerical library that includes the inverse of the distribution function explicitly.)

Now remember what we said earlier: if the exponent in the denominator is less than 2 (that is: $\beta \leq 2$ or $\alpha \leq 1$), the “mean does not exist”. In practice, we can evaluate the mean for any sample of points, and for any *finite* sample, the mean will of course come out finite as well. But as we take more and more points, the mean does not settle down — instead it keeps on growing. On the other hand, if the exponent in the denominator is strictly larger than 2 ($\beta > 2$ or $\alpha > 1$), then the mean does exist and its value is not dependent on the sample size.

I would like to emphasize again how counter-intuitive the behavior for $\alpha \leq 1$ is: we usually expect that larger samples will give us better results with less noise. But in this particular scenario, the opposite is true!

We can explore this behavior using the very simple program shown in listing 9.1. All it does is generate 10 million random numbers, distributed according to a Pareto distribution. I generate those numbers using the method described at the beginning of this section; alternatively, I could have used the `paretovariate()` function in the standard `random` module. We maintain a running total of all values (so that we can form the mean) and also keep track of the largest value seen so far. The results for two runs with $\alpha = 0.5$ and $\alpha = 1.2$ are shown in figures 9.12 and 9.13, respectively.

The typical behavior for situations with $\alpha \leq 1$ and $\alpha > 1$ is immediately apparent: whereas in figure 9.13, the mean settles down pretty quickly to a finite value, the mean in figure 9.12 continues to grow.

We can also recognize clearly what drives this behavior: for $\alpha \leq 1$, very large values occur relatively frequently. Each such occurrence leads to an upwards jump in the total sum of values seen, which is reflected in a concomitant jump in the mean. Over time, as more trials are conducted, the denominator in the mean grows, and hence the value of the mean begins to fall. However (and this is what is different for $\alpha \leq 1$ and $\alpha > 1$), before the mean has fallen back to its previous value, a *further* extraordinarily large value occurs, driving the

```

import sys, random

def pareto( alpha ):
    y = random.random()
    return 1.0/pow( 1-y, 1.0/alpha )

alpha = float( sys.argv[1] )

n, ttl, mx = 0, 0, 0

while n<1e7:
    n += 1

    v = pareto( alpha )

    ttl += v
    mx = max( mx, v )

    if( n%50000 == 0 ):
        print n, ttl/n, mx

```

Example 9.1: TBD

sum (and hence the mean) up again, with the consequence that the numerator of the expression ttl/n in listing 9.1 grows faster than the denominator.

You may want to experiment with this kind of system yourself. The behavior at the borderline value of $\alpha = 1$ is particularly interesting. You may also want to investigate how quickly ttl/n grows with different values of α . Finally, don't just look at the mean. Similar considerations hold for the standard deviation (see our discussion regarding this point earlier in the chapter).

9.7 Further Reading

- *An Introduction to Probability Theory and Its Applications, Vol. 1.* William Feller. 3rd ed., Wiley. 1968.

Every introductory book on probability theory covers most of the material in this chapter. This classic is my personal favorite, for its deep, yet accessible treatment, and for its large selection of interesting or amusing examples.

- *An Introduction to Mathematical Statistics and Its Applications.* Richard J. Larsen and Morris L. Marx. 4th ed., Prentice Hall. 2005.

This is my favorite book on theoretical statistics. The first third contains

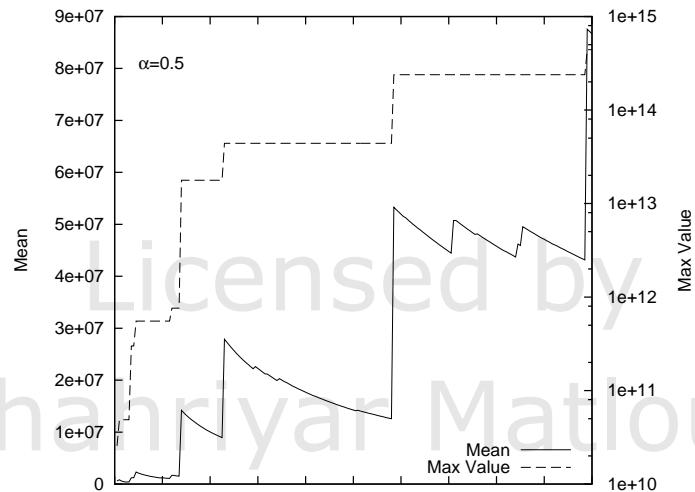


Figure 9.12: TBD

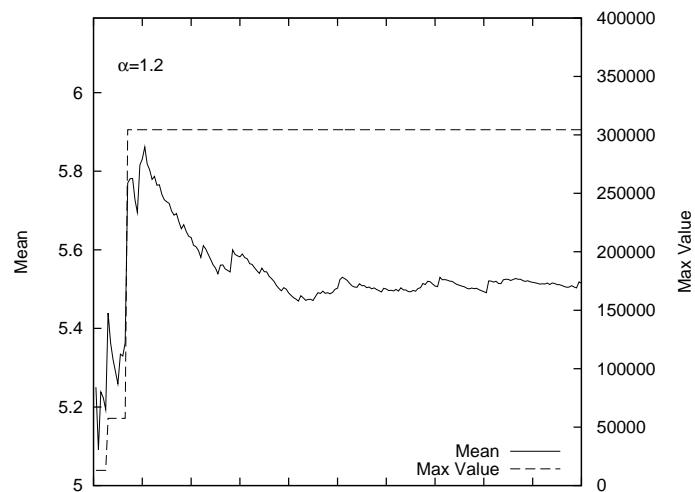


Figure 9.13: TBD

a good, practical introduction to many of this chapter's topics.

- *NIST/SEMATECH e-Handbook of Statistical Methods*. NIST. <http://www.itl.nist.gov/div898/handbook/>. 2010.

This free eBook is made available by the National Institute for Standards and Technology (NIST). There is a wealth of reliable, high-quality information here.

- *Statistical Distributions*. Merran Evans, Nicholas Hastings, Brian Peacock. 3rd ed., Wiley-Interscience. 2000.

This short, but accessible reference includes basic information on 40 of the most useful or important probability distributions. If you want to know what distributions exist and what their properties are, this is a good place to start.

- *Power laws, Pareto distributions and Zipf's law*. M. E. J. Newman. Contemporary Physics, Vol46, No5, p323. 2005.

This review paper provides a knowledgeable, yet very accessible introduction to the field of power laws and heavy-tail phenomena. Highly recommended. (Versions of the document can be found on the Web.)

- *Modeling Complex Systems*. Nino Boccara. Springer. 2004.

Chapter 8 of this book provides a succinct and level-headed overview of the current state of research into power law phenomena. (A second edition of this book is expected for late 2010.)

Chapter 10

What You Really Need to Know About Classical Statistics

Basic classical statistics has always been somewhat of a mystery to me: a topic full of obscure notions such as t -tests and p -values, and confusing statements like that “we fail to reject the null hypothesis” — which I have to read three times, and I still don’t know what it is saying: yes, no, or maybe?¹ To top it all off, all this formidable machinery is then used to draw conclusions that don’t seem to be all that interesting...it’s usually something about whether the means of two data sets are the same or different. Why would I care?

Eventually, I figured it out, and I also figured out why the field seemed so particularly obscure initially. In this chapter, I want to explain to you what classical statistics does, why it is the way it is, and what it is good for. I am not attempting to teach you how to perform the typical statistical methods: this would require a separate book, and many have indeed been written. (I will make some recommendations for further reading in this regard at the end of this chapter.) Instead, in this chapter, I will tell you what all these other books *omit*.

Let me take you on a trip. I hope you know where your towel is.

10.1 Genesis

To understand classical statistics, it is necessary to realize how it came about. The basic statistical methods that we know today were developed in the late 19th and early 20th century, mostly in Great Britain, by a very small group of

¹I am not alone — even professional statisticians have the same experience. Read the preface of “Bayesian Statistics” by Peter M. Lee (Hodder and Arnold, 2004), for example.

people (basically three or four). Of those, one worked for the Guinness brewing company, and another, the most influential one of them all, worked at an agricultural research lab (trying to increase crop yields and the like). This bit of historical context tells us something about their working conditions and primary challenges:

- No computational capabilities — all computations had to be performed with paper and pencil.
- No graphing capabilities, either — all graphs had to be generated with pencil, paper, and a ruler. (And complicated graphs, such that required prior transformations or calculations using the data, were particularly cumbersome.)
- Finally, very small, and very expensive data sets (often not more than 4 or 5 points), which could only be obtained at great difficulty (if it always takes a full growing season to generate a new data set, you try *very* hard to make do with the data you already have!).

In other words, their situation was almost entirely the opposite of our situation today:

- Computational power is basically free (within reason).
- Interactive graphing and visualization capabilities on every desktop.
- Often huge amounts of data.

So, it should come as no surprise that the methods those early researchers developed seem so out of place to us: they spent a great amount of effort and ingenuity solving problems we simply no longer have! This bit of realization goes a long way to explain why classical statistics is the way it is, and why it often seems so strange to us today.

By contrast, *modern* statistics is very different, with a greater emphasis on non-parametric methods and Bayesian reasoning, and leveraging current computational capabilities through simulation and resampling methods. The book by Larry Wasserman (see the Further Reading section at the end of this chapter) provide an overview for a more contemporary point of view.

However, almost all *introductory* statistics books, that is, those books one is likely to pick up as a beginner, continue to limit themselves to the same selection of slightly stale topics. Why is that? I would think that institutional inertia has much to do with it, but even more so the expectations from the “end-user” community. Statistics has always been a support science for other fields: originally agriculture, but also medicine, psychology, sociology, and others. And these fields, which merely apply statistics, but are not actively developing it themselves, continue to operate largely using classical methods. On the other hand, the Machine Learning community, with its roots in computer science, but

with great demands for statistical methods, provides a welcome push for the more wide-spread adoption of more modern methods.

Keep this historical perspective in mind as we take a deeper look at statistics in the rest of this chapter.

10.2 Statistics, defined

All of statistics deals with the following scenario: we have a *population*, that is the set of all possible outcomes. Typically, this set is large: all male US citizens, for example, or: all possible web server response times. Rather than dealing with the total population (which may be either impossible, infeasible, or merely inconvenient), we instead work with a *sample*. A sample is a subset of the total population, which has been chosen in such a way that it is representative for the overall population. Now we may ask: what conclusions about the *population* at large can we draw given the concrete *sample*? It is this specific question that classical statistics answers, in a process known as *statistical inference*: properties of the population are inferred from properties of a sample.

Intuitively, we do this kind of thing all the time. For example, given the heights of five men (let's say: 178cm, 180cm, 179cm, 178cm, and 180cm), we are immediately comfortable forming the average (which is 179cm) and concluding that the "typical" body size, for *all* men in the population (not just the five in the sample!) is 179cm, "more or less". This is where formal classical statistics comes in: it provides us with a way of making the vague "more or less" statement precise and *quantitative*. It allows us to make specific statements about the population, given the sample, such as: "We expect x percent of men to be between y and z cm tall" or: "We expect fewer than x percent of all men to be taller than y cm", and so on.

Classical statistics is mostly concerned with two procedures: *parameter estimation* (or "estimation" for short) and *hypothesis testing*. Parameter estimation works as follows: we assume that the population is described by some distribution, for example the Gaussian:

$$N(x; \mu, \sigma) = \exp\left(\left(-\frac{x - \mu}{2\sigma}\right)^2\right)$$

and we would like to estimate values for the parameters (μ and σ this case) from a sample. Note that once we have estimates for the parameters, the distribution describing the population is fully determined, and we can calculate any desired property of the population directly from the distribution (at least in principle). Parameter estimation comes in two flavors: *point estimation* and *interval estimation*. The first just gives us a specific value for the parameter, whereas the second gives us a range of values which is supposed to contain the true value.

Compared with parameter estimation, hypothesis testing is the weirder of the two procedures. It does not attempt to quantify the size of an effect, but

merely tries to answer the question whether there is any effect at all. Note well that this is a largely theoretical argument — from a practical point of view, the existence of an effect cannot be separated entirely from its size. We will come back to this point later. But first, let's understand how hypothesis testing works.

Imagine we have developed a new fertilizer, but we don't know yet whether it actually works. Now we run an experiment: we divide a plot in two, and treat the crops on half of the plot with the new fertilizer. Finally, we compare the yields: are they different? The specific amounts of the yield will almost surely differ, but is this difference due to the treatment, or is it merely a chance fluctuation? Hypothesis testing helps us to decide how large the difference needs to be to be *statistically significant*.

Formal hypothesis testing now proceeds as follows. First we have to set up two hypothesis that we want to decide between: the *null hypothesis* (no effect, that is no difference between the two experiments) and the *alternate hypothesis* (there is an effect so that the two experiments have significantly different outcomes). If the difference between the outcomes of the two experiments is statistically significant, then we have sufficient evidence to *reject the null hypothesis*, otherwise we *fail to reject the null hypothesis*. In other words, if the outcomes are not sufficiently different, we retain the null hypothesis that there is no effect.

This convoluted, indirect line of reasoning is required because, strictly speaking, no hypothesis can ever be proved correct by empirical means: if we find evidence *against* a hypothesis, then we can surely reject it, but if we *don't* find evidence against the hypothesis, then we retain it as a hypothesis — at least until we do find evidence against it (which may possibly never happen, in which case we retain the hypothesis indefinitely).

This, then, is the process by which hypothesis testing proceeds: because we can never prove that a treatment was successful, we instead invent a contradicting statement which we can prove to be *false*. The price we pay for this double negative ("it's *not* true that there is *no* effect") is that the test results mean exactly the opposite from what they seem to be saying: "retaining the null hypothesis", which sounds like a success, means that the treatment had no effect; whereas "rejecting the null hypothesis" means that the treatment did work. This is the first problem with hypothesis testing: it involves a convoluted, indirect line of reasoning, and a terminology which seems to be saying the exact opposite from what it means.

But there is another problem with hypothesis testing: it makes a statement that has almost no practical meaning! By reducing the outcome of an experiment to the boolean choice between "significant" and "not significant", it creates an artificial dichotomy, which is not an appropriate view of reality. Experimental outcomes are not either strictly significant or strictly non significant: they form a continuum, and to judge the results of an experiment we need to know where along the continuum the experimental outcome falls, *and* how robust the estimate is. If we have this information, we can decide how to interpret the experimental result and what importance to attach to it.

Classical hypothesis testing exhibits two well-known traps: the first is that

an experimental outcome which is *marginally* outside the statistical significance level abruptly changes the interpretation of the experiment from “significant” to “not significant” — a discontinuity in the interpretation which is not borne out by the minimal change in the actual outcome of the experiment. The other problem is that almost any effect, no matter how small, can be made “significant” by increasing the sample size. This can lead to “statistically significant” results, which nevertheless are too small to be of any practical importance. All of this is compounded by the arbitrariness of the chosen “significance level” (typically 5 percent). Why not 4.99? Or 1? Or 0.1? This seems to render the whole hypothesis testing machinery (at least as generally practiced) fundamentally inconsistent: on the one hand, we introduce an absolutely sharp cutoff into our interpretation of reality — on the other hand, we choose the position of this cutoff in an arbitrary manner. That does not seem right.

(There is a third trap: at the 5 percent significance level, you can expect one out of 20 tests to give the wrong result. This means that if you run enough tests, you will always find one that supports whatever conclusion you want to draw. This practice is known as *data dredging* and is very strongly frowned upon.)

Moreover, in any practical situation, the actual size of the effect is so much more important than its sheer existence, that hypothesis testing very often simply misses the point. A project I recently worked on provides an example of this: the question arose whether two events were statistically independent (this is a form of hypothesis testing). Yet, for the decision that was ultimately made, it did not matter whether the events truly were independent (they were not), but that treating them as independent made no measurable difference to the company’s balance sheet.

Hypothesis testing has its place, but typically in rather abstract or theoretical situations, where the mere existence of an effect constitutes an important discovery (“Is this coin loaded?” — “Are people more likely to die a few days after their birthdays rather than before?”). If this describes your situation you will quite naturally employ hypothesis tests. If, on the other hand, the size of an effect is of interest to you, then you should feel free to ignore tests altogether, and instead work out an estimate of the effect, including its confidence interval. This will give you the information that you need. You are not “doing it wrong” if you have not used a significance test somewhere along the way.

Lastly, I’d like to point out that the statistics community itself has become uneasy with the emphasis that is placed on tests in some fields (notably medicine, but also social sciences). Historically, hypothesis testing was invented to deal with sample sizes which were so small (possibly containing only 4 or 5 events), that drawing any conclusion at all was a challenge, and therefore the very broad “effect/no effect” distinction was about the best one could do. If interval estimates are available, there is no reason to use statistical tests. The Wikipedia entry on *p*-values (explained below) provides some starting points to the controversy.

I have devoted quite a bit of space to a topic that I don’t seem to consider particularly relevant. But hypothesis tests feature so large in introductory statistics books and classes, and are so obscure and counter-intuitive, that

I found it important to provide some background. In the next section, we will take a more detailed look at some of the concepts and the terminology that you are likely to find in introductory (or not so introductory) statistics books and classes.

10.3 Statistics, explained

We have already encountered several well-known probability distributions before (see chapter 9), such as the Binomial (used for trials resulting in Success or Failure), the Poisson (applicable in situations where events are evenly distributed according to some density), and the ubiquitous Normal or Gaussian distribution. All of these distributions describe some real-world, observable phenomenon.

In addition, classical statistics uses several distributions that describe the distribution of certain quantities which are not observed, but calculated. These distributions are not (or not usually) used to describe events in the real world. Instead, they describe how the outcomes of specific typical calculations involving random quantities will be distributed. There are four of these distributions, and they are known as *sampling distributions*.

The first of these (and the only one having much use outside of theoretical statistics), is the Normal distribution. As a sampling distribution, it is of interest because we already know that it describes the distribution of a sum of independent, identically distributed random variables. In other words, if X_1, X_2, \dots, X_n are random variables, then $Z = X_1 + X_2 + \dots + X_n$ will be normally distributed, and (because we can divide by a constant), the average $m = (X_1 + X_2 + \dots + X_n)/n$ will also follow a Gaussian. It is this last property that makes the Gaussian important as a sampling distribution: it describes *the distribution of averages*. One caveat: to arrive at a closed formula for the Gaussian, we need to know the variance (that is, the width) of the distribution from which the individual X_i are drawn. For most practical situations, this is not a realistic requirement, and in a moment we will discuss what to do if it is not known.

The second sampling distribution is the χ^2 (*chi-square*) distribution. It describes the distribution of the sum of *squares* of independent, identically distributed Gaussian random variables. So, if X_1, X_2, \dots, X_n are Gaussian random variables with unit variance, then $U = X_1^2 + X_2^2 + \dots + X_n^2$ will follow a chi-square distribution. Why should we care? Because we form this kind of sum every time we calculate the variance. (Remember that the variance is defined as $\frac{1}{n} \sum (x_i - m)^2$.) Hence, the chi-square distribution is used to describe *the distribution of variances*. The number n of elements in the sum is referred to as *the number of degrees of freedom* of the chi-square distribution and is an additional parameter that we need to know if we want to evaluate the distribution numerically.

The third sampling distribution describes the behavior of the ratio T of a normally (Gaussian) distributed random variable Z and a chi-square dis-

tributed random variable U . This distribution is the famous *Student t distribution*. Specifically, let Z be distributed according to the standard Gaussian distribution and U according to the chi-square distribution with n degrees of freedom. Then $T = Z / \sqrt{U/n}$ is distributed according to the t distribution with n degrees of freedom. As it turns out, that is the correct formula to use for the *distribution of the average, if the variance is not known*, but has to be determined from the sample together with the average.

The t distribution is a symmetric, bell-shaped curve like the Gaussian, but with fatter tails. How fat the tails are depends on the number of degrees of freedom (that is: the number of data points in the sample). As the number of degrees of freedom goes up, the t distribution becomes more and more like the Gaussian. In fact, for n larger than about 30, the differences between them are negligible. This is an important point to keep in mind: the distinction between the t distribution and the Gaussian matters only for small samples, that is samples containing less than approximately 30 data points. For larger samples, it is alright to use the Gaussian instead of the t distribution.

The final of the four sampling distributions is *Fisher's F distribution*, which describes the behavior of the ratio of two chi-square random variables. We care about this in situations where we want to compare variances against each other (for example to test whether they are equal or not).

These are the four sampling distributions of classical statistics. I am not going to trouble you with the formulas for these distributions, or show you their graphs. You can find them in every statistics book. What is important to me here is to make clear what it is they are describing and why they are important. In short, if you have n independent, but identically distributed measurements, then the sampling distributions describe how the average, the variance, and their ratios will be distributed. The sampling distributions therefore allow us to reason about averages and variances. That's why they are important, and why statistics books spend so much time on them.

One way to use them is to construct confidence intervals for an estimate. Here is how it works: we have n observations. We can find the average and variance of these measurements, and the ratio of the two. Finally, we know that the ratio is distributed according to the t distribution. Hence we can find that interval, which has a 95 percent probability of containing the true value (see figure 10.1). The boundaries of this range are the 95-percent confidence interval, that is, in only 1 out of 20 cases, we expect the true value to fall outside this confidence range.

A somewhat similar concept can be applied to hypothesis testing, where sampling distributions are often used to calculate so-called *p-values*. A *p-value* is an attempt express the strength of the evidence in a hypothesis test and in doing so to soften the sharp binary "significant/not significant" distinction mentioned earlier. A *p-value* is *the probability to obtain a value as or more extreme than the one actually observed* under the assumption that the null hypothesis is true (see figure 10.2.) In other words, if the null hypothesis is that there is no effect, and the observed effect size is x , then the *p-value* is the probability to observe an effect at least as large as x . Obviously, a large effect is improbable (small

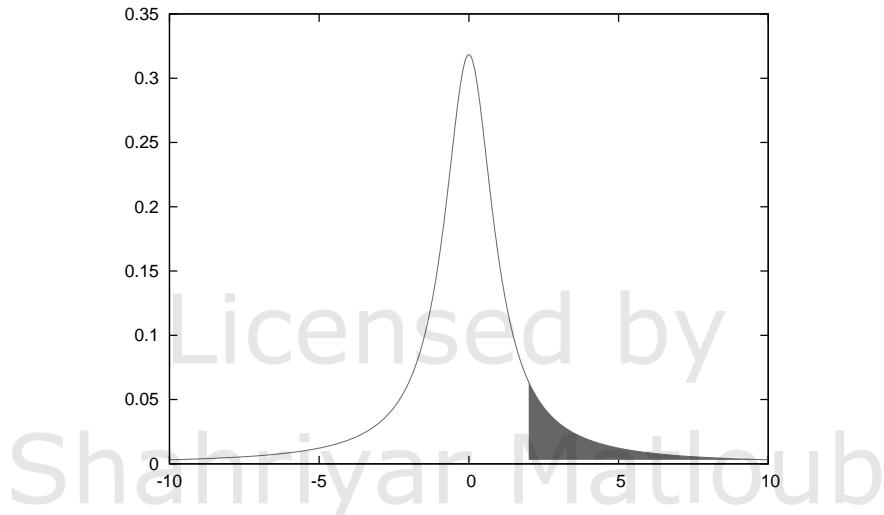


Figure 10.1: TBD

p-value) if the null hypothesis (zero effect) is true; hence a small *p*-value is considered strong evidence against the null hypothesis. However, a *p*-value is not “the probability that the null hypothesis is true” — such an interpretation (although appealing!) is incorrect. The *p*-value is the probability to obtain an effect as large or larger than the observed one, *if* the null hypothesis is true. (Classical statistics does not make probability statements about the truth of hypotheses. Doing so puts us into the realm of Bayesian statistics. We’ll come back to that topic towards the end of this chapter.)

By the way, if you would be thinking that this approach to hypothesis testing, with its sliding *p*-values, appears to be quite different from the fast-and-dry significant/not-significant approach discussed earlier, you would be right. Historically, two competing theories of significance tests have been developed and generated quite a bit of controversy; and even today they sit a little awkwardly next to each other. (The approach based on sliding *p*-values, which need to be interpreted by the researcher, is due to Fisher; the decision-rule approach was developed by Pearson and Neyman.) But enough already. You can consult any statistics book if you want to know more details.

10.3.1 Example: Formal Tests versus Graphical Methods

Historically, classical statistics evolved the way it did because working with actual *data* was hard. The early statisticians therefore made a number of simplifying assumptions (mostly that data would be normally distributed) and then proceeded to develop mathematical tools (such as the sampling distribution introduced earlier), which allowed them to reason about data sets in a

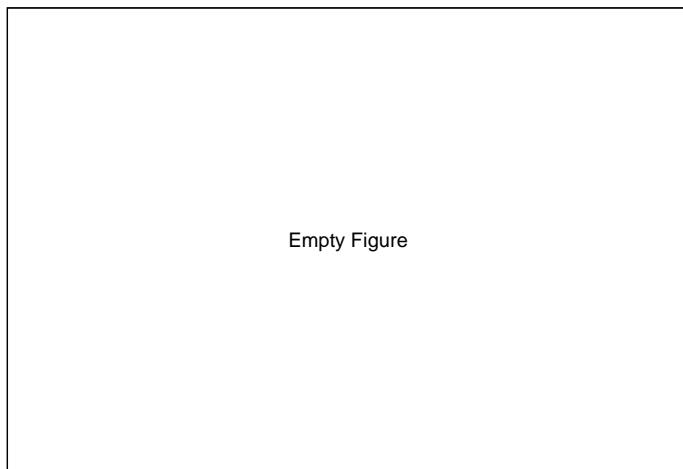


Figure 10.2: TBD

general way, requiring only the knowledge of a few, easily calculated summary statistics (such as the mean). The ingenuity of it is amazing. But it has lead to an emphasis on formal technicalities, as opposed to the direct insight into the data. Today, our situation is different, and we should take full advantage of that.

An example will demonstrate what I mean. Below are two data sets. Are they the same or different (in the sense that their means are the same or different)?²

0.209	0.225
0.205	0.262
0.196	0.217
0.210	0.240
0.202	0.230
0.207	0.229
0.224	0.235
0.223	0.217
0.220	
0.201	

In case study 9.2.1 of their book, Larsen and Marx (see the section with Further Reading at the end of this chapter) labor for several pages, and finally conclude that the data sets are different at the 99 percent level of significance.

Figure 10.3 shows a box plot of both data sets. Case closed.

²This is a famous data set and its history is colorful but not all that interesting. A web search for "Quintus Curtius Snodgrass" will turn up plenty of references.

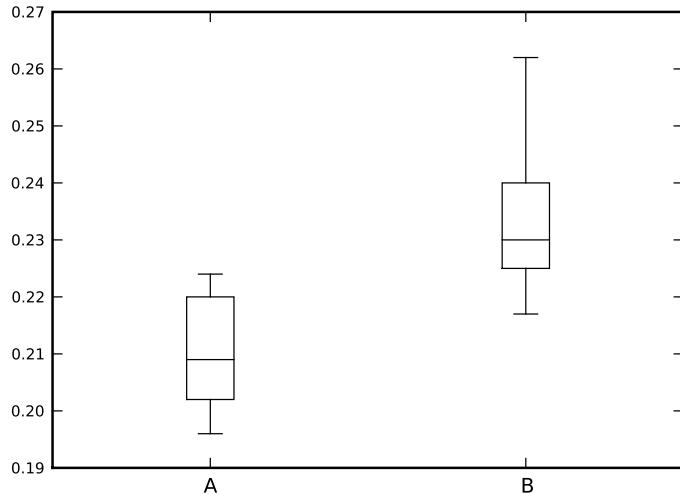


Figure 10.3: TBD - Boxplot of Quintus Curtius Snodgrass

(In fairness, the formal test does something that a graphical method can not do: it gives us a quantitative criterion by which to make a decision. I hope that the discussion in this chapter has convinced you that this is not always an advantage, because it can lead to blind faith in “the number”. Graphical methods require you to interpret the results and take responsibility for the conclusions. Which is why I like them: they keep you honest!)

10.4 Controlled Experiments vs Observational Studies

Besides the machinery of formal statistical inference (using the sampling distributions just discussed), the early statistics pioneers also developed a general theory how best to undertake statistical studies. This conceptual framework is sometimes known as *Design of Experiment* and is worth knowing about — not least because so much of typical data mining activity does *not* make use of it.

The most important distinction formalized by the Design of Experiment theory is the one between an *observational study*, compared to a *controlled experiment*. As the name implies, in a controlled experiment, we are able to control many aspects of the experimental setup and procedure; in particular we control which treatment is applied to which experimental unit (we will define these terms in a moment). For example, in an agricultural experiment, we would treat some, but not all, of the plots with a new fertilizer, and later compare the

yields from the two treatment groups. By contrast, in an observational study, we merely collect data as it becomes (or already is) available. In particular, retrospective studies are always observational (not controlled).

In a controlled experiment, we are able to control the “input” of an experiment (namely the application of a treatment), and therefore are able to draw much more powerful conclusion from the output. In contrast to observational studies, a properly conducted controlled experiment can provide strong support for cause-and-effect relationships between two observations and can be used to rule out hidden (or confounding) causes. Observational studies can merely *suggest* the existence of a relationship between two observations, but can neither prove that one observation is caused by the other, nor can they rule out that additional (un-observed) factors have played a role.

The following (intentionally whimsical) example will serve to make the point: let’s say we have data that suggests that cities with many lawyers also have many espresso stands, and that cities with few lawyers have few espresso stands. In other words, there is strong correlation between the two quantities. But what conclusions can we draw about the causal relationship between the two? Are lawyers particularly high consumers of expensive coffee? Or does caffeine make people more litigious? In short, there is no way for us to determine what is cause and what is effect in this example. In contrast, in the controlled agricultural experiment mentioned earlier, if the fertilized yields are higher than the ones from the untreated control plots, we have strong reason to conclude that this is due to the effect of the fertilizer treatment.

In addition to the desire to establish that the treatment indeed causes the effect, we also want to rule out that there are additional, unobserved factors which might account for the observed effect. Such factors, which influence the outcome of a study but are not themselves part of it, are known as *confounding* (or “hidden” or “lurking”) variables. In our agricultural example, differences in soil quality might have a significant influence on the yield — larger, perhaps, than the fertilizer. The spurious correlation between the number of lawyers and espresso stands mentioned earlier is almost certainly due to confounding: larger cities have more of everything! (Even if we account for this effect and consider the per capita *density* of lawyers and espresso stands, there is still a plausible confounding factor: the income generated per head in the city.) I will explain below how *randomization* can help to remove the effect of confounding variables.

The distinction between controlled experiments and observational studies is most critical. Many of the most controversial scientific or statistical issues involve observational studies. In particular reports in the mass media often involve studies which (inappropriately) draw causal inferences from observational studies (about topics such as the relationship between gun laws and homicide rates, for example). Sometimes controlled experiments are not possible, with the result that it becomes almost impossible to settle certain questions for good — the controversy around the connection between smoking and lung cancer provides a good example.

In any case, make sure you understand clearly the difference between ob-

servational and controlled studies, and the fundamental limitations of observational studies!

10.4.1 Design of Experiments

In a controlled experiment, we divide the *experimental units* which make up our sample into two or more groups and apply different *treatments* or *treatment levels* to the units in each group. In the agricultural example mentioned earlier, the plots correspond to the “experimental units”, fertilization is the “treatment”, and the options “fertilizer” and “no-fertilizer” are the “levels”).

Experimental design involves several techniques to improve the quality and reliability of any conclusions drawn from a controlled experiment.

Randomization Randomization means that treatments (or treatment levels) are assigned to experimental units in a random fashion. Proper randomization suppresses systematic errors. (If we assign fertilizer treatment randomly to plots, we remove the systematic influence of soil quality, which might be a confounding factor, because high-quality and low-quality plots are equally likely to receive the fertilizer treatment.) To achieve true randomization is not as easy as it looks — I’ll come back to this point shortly.

Replication Replication means that the same treatment is applied to more than one experimental unit. Replication serves to reduce the variability of the results, by averaging over a larger sample. Replicates should be independent of each other: nothing is gained if we repeat the same experiment on the same unit multiple times.

Blocking Occasionally, we know (or at least strongly suspect) that not all experimental units are equal. If this is the case, it may make sense to group equivalent experimental units into “blocks”, and to treat each such block as a separate sample. For example, if we know that plots A and C have poor soil quality, and B and D have better soil, then we would form two blocks, consisting of (A,C) and (B,D), respectively, and then proceed to make a randomized assignment of treatments *for each block separately*. Similarly, if in a study of web traffic we knew that traffic was drastically different in the morning and the afternoon, we could collect and analyze data for both time periods separately: this also is a form of blocking.

Factorization The last of these techniques applies only to experiments involving several treatments (for example irrigation and fertilization, to stay within our agricultural framework). The simplest experimental design would make only a single change at any given time, so that we would observe yields with and without irrigation, and with and without fertilizer. But this approach misses the possibility that there are *interactions* between the two treatments — possibly the effect of the fertilizer is significantly higher if coupled with improved irrigation? Therefore, in

a factorial experiment, all possible combinations of treatment levels are tried out. Even if a fully factorial experiment is not possible (the number of combinations goes up quickly as the number of different treatments grows), there are rules how best to select combinations of treatment levels to draw the optimal conclusions from the study.

Another term you may come across in this context is ANOVA (Analysis of Variance), which is a standard way of summarizing results from controlled experiments. It emphasizes the variations within each treatment group for easy comparison with the variances between the treatments, so that we can determine whether the differences between different treatments are significant compared to the variation within each treatment group. ANOVA is a clever bookkeeping technique, but does not introduce particularly noteworthy new statistical concepts.

A word of warning: When conducting a controlled experiment, make sure that you apply the techniques properly; in particular, beware of *pseudo-randomization* and *pseudo-replication*.

Pseudo-randomization occurs if the assignment of treatments to experimental units is not truly random. This can occur relatively easily, even if the assignment is *seemingly* random. For example, if you would like to try out two different drugs on lab rats, it is not sufficient to “pick a rat at random” from the cage to administer the treatment. What does “at random” mean? It might very well mean picking the most active rat first, because it comes to the cage door. Or maybe the least aggressive-looking one. In either case, there is a systematic bias!

Here is another example, possibly closer to home: the web-lab. Two different site designs are to be presented to viewers, and the objective is to measure conversion rate or click-throughs, or whatever. There are multiple servers, so we dedicate one of them (chosen “at random”!) to serve the pages with the new design. What’s wrong with that?

Everything! Do you have any indication that web requests are assigned to servers in a random fashion? Or do servers for example have strong geographic bias? Let’s assume the servers are behind some “big-IP” box, which routes requests to the servers. How is the routing conducted? Randomly? Round-robin? Traffic based? Is the routing smart, so that servers with slower response times get fewer hits? What about sticky sessions? What about the relationship between slower response times and sticky sessions? Is the router re-ordering incoming requests in some way? Many questions — questions that randomization is intended to *avoid*. In fact, you are not running a controlled experiment at all: you are running an observational study!

The only way that I know to run a controlled experiment is to decide ahead of time which experimental unit will receive which treatment. In the lab-rat example, rats should have been labeled, and then treatments assigned to the labels using a (reliable) random number generator or random table. In the web server example, it is harder, because the experimental units are not known ahead of time. A simple rule (such as: show the new design to every $n -$

th request) will not do, because there may be significant correlation between subsequent requests. It's not so easy.

Pseudo-replication occurs if experimental units are not truly independent. Injecting the same rat five times with the same drug does not reduce variability! Similarly, running the same query against a database might be highly misleading, due to changing cache utilization. And so on. In my experience, pseudo-replication is easier to spot, and hence tends to be less of a problem than pseudo-randomization.

Finally, I should mention one other term, that often comes up in the context of proper experimental process: *blind* and *double-blind* experiments. The idea with a blind experiment is that the experimental unit should now know which treatment it receives, in a double-blind experiment, the investigator him- or herself does not know either (at least, at the time of the experiment — of course, we need to keep track which is which, but this information is not available until the experiment is complete and is being analyzed). The purpose of blind and double-blind experiments is to avoid the knowledge of the treatment level becoming a confounding factor — if people know that they are given a new drug, this knowledge itself may contribute to their well-being. Or if an investigator knows which field is receiving the fertilizer, he or she might be weeding that particular field more vigorously and therefore introduce some invisible and unwanted bias. Blind experiments play a huge role in the medical field, but can also be important in other situations. However, I would like to emphasize that the question of blindness (which is about experimental process) is different from the Design of Experiment prescription (which are intended to reduce statistical uncertainty).

10.4.2 Perspective

It is important to maintain an appropriate perspective on these matters.

In practice, many studies are observational, not controlled. Occasionally, this is a painful loss, and only due to the inability to conduct a proper controlled experiment (smoking and lung cancer, again!). At other times, observational studies add great value. One reason is that they can be exploratory and discover new and so far unknown behavior. In contrast, controlled experiments are always confirmatory, deciding between the effectiveness or ineffectiveness of a specific “treatment”.

Observational studies can be used to derive predictive models, even while setting aside the question of causation. The Machine Learning community for instance attempts to develop *classification* algorithms, which use descriptive attributes or *features* of the unit to predict whether the unit belongs to a given class. They work entirely without controlled experiments and have developed methods to quantify the accuracy of their results. (We will describe some in chapter 18.)

On the other hand, it is important to understand the limitations of observational studies, in particular their inability to support strong conclusions regarding cause-and-effect relationships and the absence of confounding factors.

In the end, the power of controlled experiments can be their limitation, because they require a level of control that limits their application to areas that aren't that contentious to begin with.

10.5 The Other Point of View: Bayesian Statistics

There is an alternative approach to statistics, based on a different interpretation of the concept of *probability* itself. That may come as a surprise, since probability seems to be such a basic concept. The problem is that, although we have a very strong intuitive sense of what we mean by the word "probability", it is not so easy to give it a rigorous meaning that can be used to develop a mathematical theory.

The interpretation of probability used by classical statistics (and to some degree by abstract probability theory) treats probability as a *limiting frequency*: if you toss a fair coin "a large number of times", you will obtain heads about half of the time, hence the probability for heads is 1/2. Arguments and theories starting from this interpretation are often referred to as "frequentist".

An alternative interpretation of probability considers it as the degree of our ignorance about an outcome: since we don't know which side is going to be on top in the next toss of a fair coin, we assign each possible outcome the same probability, namely 1/2. We can therefore make statements about the probabilities associated with individual events, without having to invoke the notion of a large number of repeated trials. Because this approach to probability and statistics makes use of *Bayes Theorem* at a central step in its reasoning, it usually called *Bayesian Statistics*, and has become increasingly popular in recent years. Let's compare the two interpretations in a bit more detail.

10.5.1 The Frequentist Interpretation of Probability

In the frequentist interpretation, probability is interpreted as the limiting frequency of each outcome, if an experiment is repeated a large number of times. This "frequentist" interpretation of probability is the reason for some of the peculiarities of classical statistics. For example, in classical statistics it is incorrect to say that a 95% confidence interval for some parameter has a 95% chance of containing the true value — the true value is either contained in the interval or not; period. The only statement that we can make is that if we perform an experiment to measure this parameter many times, in about 95% of all cases, the experiment will yield a value for this parameter that lies within the 95% confidence interval.

This line of reasoning has a number of drawbacks:

- It is awkward and clumsy and liable to (possibly even unconscious) misinterpretations (see the paragraph above!).
- The constant appeal to a "large number of trials" is very artificial even in situations where such a sequence of trials would at least in principle be possible (such as the tossing of a coin). But it becomes wholly fictitious in situations where the trial can not possibly be repeated. The weather report will state things like: "80% chance of rain tomorrow". What is that supposed to mean? It is either going to rain tomorrow or not! So, we have to back in the unlimited sequence of trials again by saying that in 8 out of 10 cases that we observe the current meteorological conditions, we expect rain on the following day. But even this is illusionary,

because we will never observe these *precise* conditions ever again: that's what we have been learning from Chaos theory and similar areas.

- Lastly, we frequently would like to make statements such as the one about the weather above, or similar ones, like: "the patient has 60% survival probability" — "I am 25% certain that the contract will be approved". In all of these cases, the actual outcome is not of a probabilistic nature: it will rain or it will not; the patient will survive or not; the contract will be approved or not. But we would like to express a degree of certainty about the expected outcome, even if appealing to an unlimited sequence of trials is not practical or even meaningful.

From a strictly frequentist point of view, a statement such as "80% chance of rain tomorrow" is non-sensical. On the other hand, it seems to make so much intuitive sense — in what way can this intuition be made more rigorous? This question leads us to *Bayesian Statistics* or *Bayesian Reasoning*.

10.5.2 The Bayesian Interpretation of Probability

To understand the Bayesian point of view, we first need to review the concept of *conditional probability*. The conditional probability $P(A|B)$ gives us the probability for the event A , *given* (or assuming) that event B has occurred. You can easily convince yourself that the following is true:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

where $P(A \cap B)$ is the *joint probability* of finding both event A and event B . For example, it is well-known that men are much more likely to be color-blind than women: about ten percent of men are color-blind, but less than one percent of women are color-blind. These are *conditional* probabilities, namely the probability to be color-blind, *given* the gender:

$$\begin{aligned} P(\text{color-blind}|\text{male}) &= 0.1 \\ P(\text{color-blind}|\text{female}) &= 0.01 \end{aligned}$$

On the other hand, if we "randomly" pick a person off the street, we are talking about the *joint* probability that this person is color-blind *and* male, with a fifty percent chance of being male, and a ten percent conditional probability to be color-blind, given male. Hence, the joint probability for a random person to be color-blind *and* male is five percent, in agreement with the definition of conditional probability I gave earlier.

One can now rigorously prove the following theorem, which is known as *Bayes Theorem*:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

In words: the probability to find A , given B is equal to the probability to find B given A , multiplied by the probability to find A , and divided by the probability to find B .

Now, let's come back to statistics and data analysis. Assume that there is some parameter which we try to determine through an experiment (the mass of the proton or the survival rate after surgery — whatever). We are now dealing with two "events": event B is the occurrence of the specific set of measurements that we have observed.

On the other hand, the fact that the parameter is going to assume some specific value constituents the event A . We can now re-write Bayes Theorem as follows:

$$P(\text{parameter}|\text{data}) \propto P(\text{data}|\text{parameter})P(\text{parameter})$$

(I have dropped the denominator. I can do so, because the denominator is simply a constant which does not depend on the parameter that we wish to determine. The left- and right-hand sides are now no longer equal, and I have replaced the equality sign with \propto to indicate that the two sides of the expression are merely proportional: equal to within a numerical constant.)

Let's look at this equation term by term.

On the left-hand side we have *the probability to find a certain value for the parameter, given the data*. That's pretty exciting, because this is an expression that makes an explicit statement about the *probability* of an event (in this case: that the parameter has a certain value), given the data. This probability is called the *posterior probability* or simply *the posterior*, and is defined solely through Bayes Theorem, without reference to any unlimited sequence of trials. Instead, it is a measure of our "belief" or "certainty" about the outcome (that is: the value of the parameter), given the data.

The first term on the right-hand side, $P(\text{data}|\text{parameter})$, is known as the *likelihood function*. This is a mathematical expression which links the parameter to the probability to obtain specific data points in an actual experiment. The likelihood function constitutes our "model" for the system under consideration: it tells us what data we can expect to observe, given a particular value of the parameter. (We will see an example in the next section; that will help to make the meaning of this term more clear.)

Finally, the term $P(\text{parameter})$ is known as the *prior probability* or simply *the prior* and captures our "prior" (prior to the experiment) belief to find a certain outcome, specifically our prior belief that the parameter has a certain value. It is the existence of this prior that makes the Bayesian approach so controversial: it seems to introduce an inappropriately subjective element into the analysis! In reality, however, the influence of the prior on the final result of the analysis is typically small, in particular when there is plenty of data. One can also find so-called "non-informative" priors that express our complete ignorance about the possible outcomes. However, the prior is there, and forces us to think about our assumptions regarding this experiment, and to state some of these assumptions explicitly (in form of the prior distribution function.)

10.5.3 Bayesian Data Analysis — A Worked Example

All of this will become much more clear once we demonstrate all these concepts in an actual example. The example is very simple, so as not to distract from the concepts.

Assume we have a coin, which has been tossed ten times, which produced the following set of outcomes (H for heads, and T for tails):

T H H H H T T H H H

If you count them, you will find that we obtained seven heads and three tails in ten tosses of the coin.

Given this data, we would like to determine whether the coin is fair or not. Specifically, we would like to determine the probability p that a toss of this coin will turn out heads. (This is the "parameter" we would like to estimate.) If the coin is fair, p should be very close to $1/2$.

Let's write down Bayes equation, adapted to this system:

$$P(p | \{T H H H H T T H H H\}) \propto P(\{T H H H H T T H H H\} | p)P(p)$$

Notice how at this point the problem has become *parametric*: all that is left to do is to determine the value of the parameter p — more precisely, the posterior probability distribution for all values of p .

To make progress, we need to supply the likelihood function and the prior. Given this system, the likelihood function is particularly simple: $P(H|p) = p$ and $P(T|p) = 1 - p$. You should convince yourself that this choice of likelihood function gives us exactly what we want: the probability to obtain heads or tails, given p .

We also assume that the tosses are independent, so that only the total number of heads or tails matters, but not the order in which they occurred. This means that we don't need to find the combined likelihood for the specific sequence of ten tosses; instead, the likelihood of the set of events is simply the product of the ten individual tosses. (The likelihood "factors" for independent events — this argument occurs very frequently in Bayesian analysis.)

Finally, we know nothing about this coin. In particular, we have no reason to believe that any value of p is more likely than any other, so we choose as prior probability distribution the "flat" distribution $P(p) = 1$ for all p .

Collecting everything, we end up with the following expression (where I have dropped some combinatorial factors which do not depend on p):

$$P(p | \{7 \text{ heads}, 3 \text{ tails}\}) \propto p^7(1-p)^3$$

This is the posterior probability distribution for the parameter p , based on the experimental data. A graph of it is shown in figure 10.4. We can see that it has a peak near $p = 0.7$, and that is also the most probable value for p . Note that there are no tic marks on the y-axis in figure 10.4: the denominator, which we dropped earlier, is still undetermined, and therefore the overall scale of the function is not yet fixed. If we are only interested in the *location* of the maximum, this does not matter.

But we are not restricted to a single (point) estimate for p — the entire distribution function is available to us! We can now use it to construct confidence intervals for p (and since we are talking about Bayesian probabilities, it would be legal to state that "the confidence interval has a 95% chance of containing the true value of p ").

But we can also evaluate any function that depends on p , by integrating it against the posterior distribution for p . As a particularly simple example, we could calculate the expectation value of p as a single "best" estimate of p (rather than using the most probable value as we did above):

$$E[p] = \frac{\int p P(p | \{7 \text{ heads}, 3 \text{ tails}\}) dp}{\int P(p | \{7 \text{ heads}, 3 \text{ tails}\}) dp}$$

Here we finally need to worry about all the factors that we dropped along the way, and the denominator in the formula is our way of fixing the normalization after the fact. To make sure that the probability distribution is properly normalized, we divide explicitly by the integral over the whole range of values, thus ensuring that the total probability equals one (as it must).

It is interesting to look at the roles played by the likelihood and the prior in the result. In Bayesian analysis, the posterior "interpolates" between the prior and the data-based likelihood function: if there is only very little data, the likelihood function will

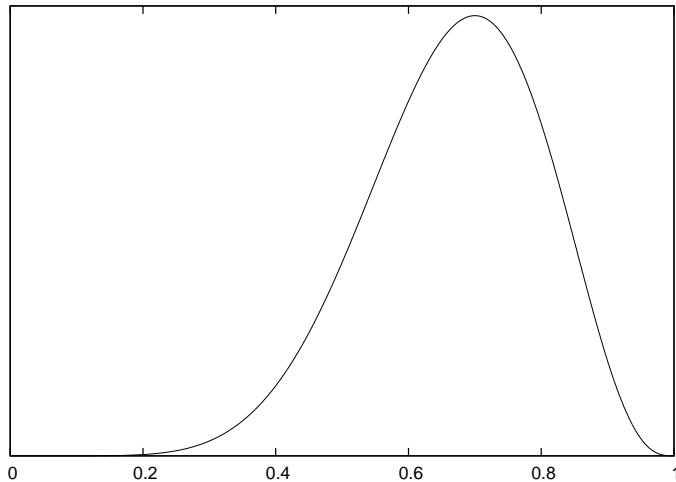


Figure 10.4: TBD

be rather flat and therefore the posterior will be more dominated by the prior. But as we collect more data (in other words, as the empirical evidence gets stronger), the likelihood function becomes more and more narrowly peaked at the most likely value of p , regardless of the choice of prior. Figure 10.5 demonstrates this effect: we see the posterior for a total of ten trials and a total of thirty trials (while keeping the same ratio of heads to tails): as we gather more data, the uncertainty in the resulting posterior shrinks.

Finally, in figure 10.6 I show the effect of the prior: whereas the posterior shown in figure 10.5 had been calculated using a flat prior, in figure 10.6 I am using a Gaussian prior, which expresses a rather strong belief that p will be between 0.35 and 0.65. For the smaller data set, the influence of this prior belief is rather significant, but as we take more and more data, the influence of the prior is ever more diminished.

10.5.4 Bayesian Inference: Summary and Discussion

Let's summarize what we have learned about Bayesian data analysis or *Bayesian Inference* and discuss what it can do for us — and what not.

First of all, the Bayesian (as opposed to frequentist) approach to inference allows us to compute a true probability distribution for any parameter in question. This has great intuitive appeal, because it allows us to make statements such as "There is a 90% chance of rain tomorrow" without having to appeal to the notion of extended trials of identical experiments.

The posterior probability distribution arises as product between the likelihood function and the prior. The likelihood links experimental results to values of the parameter, and the prior expresses our previous knowledge or belief about the parameter.

The Bayesian approach has a number of very appealing features. Of course, there is the intuitive nature of the results obtained using Bayesian arguments: real probabilities,

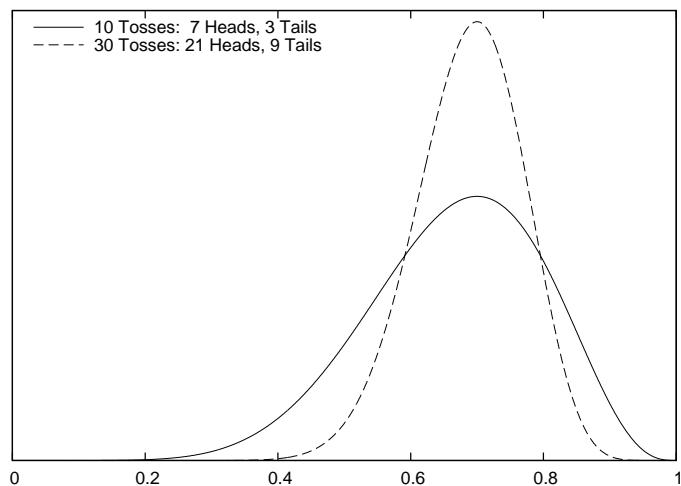


Figure 10.5: TBD

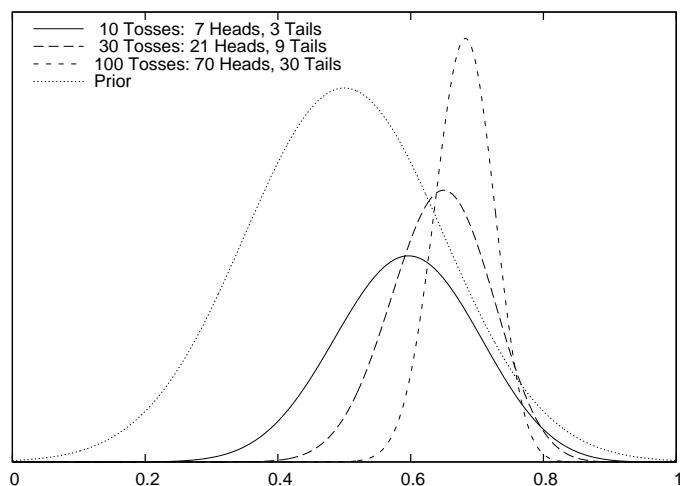


Figure 10.6: TBD

and 95% confidence intervals that have exactly the kind of interpretation one would expect! Moreover, we obtain the posterior probability distribution in full generality, without having to make limiting assumptions (such as having to assume that the data is Normally distributed, and so on).

Additionally, the way the likelihood function enters into the calculation allows for great flexibility in the way we build “models” — using the Bayesian approach, it is very easy to deal with missing data, data that is becoming available over time, or with heterogeneous data sets (with different attributes known about each data point). Because the result of Bayesian inference is a probability distribution itself, it can be used as input for a new model that builds on the previous one (hierarchical models). Additionally, we can use the prior to incorporate previous (domain) knowledge that we may have about the problem under consideration.

On the other hand, Bayesian inference has some problems, too — even when we concentrate on practical applications only, and leave the entire philosophical debate about priors and subjectivity aside.

First of all, Bayesian inference is always *parametric*, it is never just exploratory or descriptive: because Bayesian methods force us to supply a likelihood function explicitly, they force us to be very specific about our choice of model assumptions — we already must have a likelihood function in mind, otherwise we can't even get started (hence: not exploratory). Furthermore, the result of a Bayesian analysis is always a posterior distribution, that is a conditional probability of *something*, given the data. Here, *something* is some form of hypotheses that we have, with the posterior giving us the probability of this hypothesis to be true. To make this operational (and, in particular, expressible through a likelihood function) we pretty much have to parameterize the hypothesis. The inference then consists of finding the best value for this parameter, given the data — which is a parametric problem, given a specific choice of the model (that is, the likelihood function). (There are so-called “non-parametric” Bayesian methods, but in reality they boil down to parametric models with very large numbers of parameters.)

Additionally, actual Bayesian calculations are often difficult. On the one hand, Bayesian inference gives us the full explicit posterior distribution function — on the other hand, it gives us the full explicit posterior distribution function! If we want to summarize this function, we either need to find its maximum or integrate it to find an expectation value. Either of these problems is hard, in particular when the likelihood function is complicated and there is more than one parameter that we try to estimate. Instead of explicitly integrating the posterior, it is very common to *sample* it, that is, to draw random points, which are distributed according to the posterior distribution. Clearly, this is an expensive process, requiring computer time and specialized software (and the associated know-how!). There can also be some additional problems: for example, if the parameter space is very high-dimensional, evaluation of the likelihood function (and therefore the posterior), can become a problem.

By contrast, frequentist methods tend to make more assumptions up front and rely more strongly on general analytic results and approximations: with frequentist methods, the hard work has typically already been done (analytically), giving you an asymptotic or approximate formula that you only need to plug in. Bayesian methods give you the full, non-approximate result, but leave it up to you to evaluate it. The disadvantage of the plug in approach, of course, is that you might be plugging into an inappropriate formula, because some of the assumptions or approximations that were used to derive it, do not apply to your system or data set!

To bring this discussion to a close, I'd like to end with a cautionary note. Bayesian methods are very appealing and even exciting — something that is rarely said about

classical frequentist statistics. On the other hand, they are probably not very suitable for casual uses:

- Bayesian methods are parametric and specific, they are never exploratory or descriptive. If we already know the specific question to ask, then Bayesian methods may be the best way of obtaining an answer. But if we don't know the proper questions to ask (yet), Bayesian methods are not applicable.
- Bayesian methods are difficult, requiring a fair deal of sophistication, both in setting up the actual model (likelihood function and prior), as well as in the execution of the required calculations.

As far as results are concerned, there is anyway no great difference between frequentist and Bayesian analysis: if there is sufficient data (so that the influence of the prior is small), the end results are typically very similar, whether they were obtained using frequentist methods or Bayesian methods.

Finally, you may encounter some other terms and concepts in the literature, which also carry the moniker "Bayesian" in their names: Bayesian classifier, Bayesian network, Bayesian risk, and more. Very often, these have nothing to do with Bayesian (as opposed to frequentist) inference as explained in this chapter — these are typically methods which involve conditional probabilities and therefore appeal at one point or another to Bayes Theorem. A Bayesian classifier, for instance, is the conditional probability that an object belongs to a certain class, given what we know about it. A Bayesian network is a particular way of organizing the causal relationships that exist among events that depend on many interrelated conditions. And so on.

10.6 Workshop: R

R is an environment for data manipulation and numerical calculations, specifically statistical applications. Although it can be used in a more general fashion for programming or computation, its real strength is the large number of built-in (or user-contributed) statistical functions.

R is an open-source clone of the S programming language, which was originally developed at Bell Labs in the 1970s. It was one of the first environments to combine the capabilities that we today expect from a scripting language (such as memory management, proper strings, dynamic typing, easy file handling) with integrated graphics and intended for an interactive usage pattern.

I tend to stress the word *environment* when referring to R, because the way it integrates its various components is essential to R. It is misleading to think of R as a programming language that also has an interactive shell (like Python or Groovy). Instead, you better consider it as a shell, but for handling data instead of files. Alternatively, you might want to view R as a text-based spreadsheet on steroids. The "shell" metaphor in particular is helpful to motivate some of the design choices made by R.

The essential data structure offered by R is the so-called *data frame*. A data frame encapsulates a data set, and is the central abstraction that R is built on — basically all operations involve the handling and manipulation of frames in one way or the other.

Possibly the best way to think of a data frame is as comparable to a *relational database table*: a data frame is a rectangular data structure, consisting of rows and columns. Each *column* has a designated data type, and all entries in that column must be of that type. Consequentially, each *row* will in general contain entries of different types (as defined by the types of the columns), but all rows must be of the same form. All this should be familiar from relational DBs. The similarities continue: operations on frames can either project out a subset of columns, or filter out a subset of rows, resulting in a new data frame. There is even a command (`merge`) which can perform a join of two data frames on a common column. In addition (and in contrast to databases), we will frequently *add* columns to an existing frame: for example to contain the results of an intermediate calculation.

We can refer to columns by name. The names are either read from the first line of the input file, or (if not provided) R will substitute synthetic names of the form V1, V2, ... By contrast, we filter out a set of rows through various forms of “indexing magic”. Let’s look at some examples.

Consider the following input file:

Name	Height	Weight	Gender
Joe	6.2	192.2	0
Jane	5.5	155.4	1
Mary	5.7	164.3	1
Jill	5.6	166.4	1
Bill	5.8	185.8	0
Pete	6.1	201.7	0
Jack	6.0	195.2	0

Let’s investigate this data set using R, placing particular emphasis on the way you handle and manipulate data with R. Listing 10.1 shows a transcript of the complete session. The commands I typed are prefixed by the prompt >, R output is shown without the prompt.

```
> d <- read.csv("data", header = TRUE, sep = "\t" )
> str(d)
'data.frame': 7 obs. of 4 variables:
 $ Name : Factor w/ 7 levels "Bill","Jack",...: 5 3 6 4 1 7 2
 $ Height: num 6.2 5.5 5.7 5.6 5.8 6.1 6
 $ Weight: num 192 155 164 166 186 ...
 $ Gender: int 0 1 1 1 0 0 0
```

First, we read the file in and assign it to the variable `d`, which is a data frame, as discussed earlier. The function `str(d)` shows us a string representation of the data frame. We can see that the frame consists of five named columns, and we can also see some typical values for each column. Notice that R has assigned a data type to each column: height and weight have been recognized as floating-point values, the names are considered a “factor”, which is R’s way of indicating a categorical variable, and finally the gender flag is being interpreted as an integer. That is not ideal — we will come back to that.

```
> mean( d$Weight )
[1] 180.1429
> mean( d[,3] )
[1] 180.1429
```

Let's calculate the mean of the weight column to show some typical ways in which we select rows and columns. The most convenient way to specify a column is by name: `d$Weight`. The use of the dollar-sign (\$) to access members of a data structure is one of R's quirks that one learns to live with. Think of a column as a shell variable! (By contrast, the dot (.) is not an operator and can be part of a variable or function name, in the same way that an underscore (_) is used in other languages. Here, the shell metaphor is useful: remember that shells allow the dot as part of filenames!)

Although its name is often the most convenient method to specify a column, we can also use its numeric index. Each element in data frame can be accessed using its row and column index, using the familiar bracket notation: `d[row, col]`. Keep in mind that the vertical (row) index comes first, followed by the horizontal (column) index. Omitting one of them selects all possible values, as we do in the listing: `d[, 3]` selects *all* rows from the third column. Also note that indices in R start at 1 (mathematical convention), not at 0 (programming convention).

```
> mean( d$Weight[ d$Gender == 1 ] )
[1] 162.0333
> mean( d$Weight[ 2:4 ] )
[1] 162.0333
```

Now that we know how to select a column, let's see how to select rows. In R, this is usually done through various forms of "indexing magic", two examples of which are shown next in the listing. We want to find the mean weight of only the women in the sample. To do so, we take the weight column, but now index it with a logical expression. This kind of operation takes some getting used to: inside the brackets, we seem to compare a column (`d$Gender`) with a scalar — and then use the result as index on another column. What is going on here? Several things: first, the scalar on the right-hand side of the comparison is expanded into a vector of the same length as the operator on the left-hand side. The result of the equality operator is then a *boolean* vector of the same length as `d$Gender` or `d$Weight`. A boolean vector of the appropriate length can be used as index and selects only those rows for which it evaluates to True — which in this case it only does for the women in the sample. By contrast, the second line of code is much more conventional: the colon operator (:) creates a range of numbers, which are used to index into the `d$Weight` column. Remember that indices start at 1, not at 0!

These kinds of operation are very common in R: both to use some form of creative indexing to filter out a subset of rows (there more ways to do so, which I don't show), and to mix vectors and scalars in expressions. Here is another example:

```
> d$Diff <- d$Height - mean( d$Height )
```

Here, we create an additional column, called `d$Diff`, as the residual that remains when the mean height is subtracted from each individual's height. Notice how we mix a column with a scalar expression to obtain another vector.

```
summary(d)
```

Next, we calculate the summary of the entire data frame, with the new column added. Take a look at the gender column: because R interpreted the gender flag as an integer, it went ahead and calculated its "mean" and other quantities. Obviously, this is meaningless, the values in this column should be treated as categorical. This can be achieved using the `factor()` function, which also allows us to replace the uninformative numeric labels with more convenient string labels.

```
> d$Gender <- factor( d$Gender, labels = c("M", "F") )
```

As you can see if you run `summary(d)` again, R treats categorical variables differently: it counts how often each value occurs in the data set.

Lastly, let's take a look at R's plotting capabilities. First, we plot the height "as a function of" the gender. (R uses the tilde (~) to separate control and response variables; the response variable is always on the left.)

```
> plot( d$Height ~ d$Gender )
```

This gives us a box plot, which is shown in figure 10.7. On the other hand, if we plot the height as a function of the weight, we obtain a scatter plot (see figure 10.8 — without the lines, we will add them in a moment).

```
> plot( d$Height ~ d$Weight, xlab="Weight", ylab="Height" )
```

Given the shape of the data, we might want to fit a linear model to it. This is very easy to do in R — it's a single line of code:

```
> m <- lm( d$Height ~ d$Weight )
```

Notice again the tilde notation to indicate control and response variable.

Finally, we may want to add the linear model to the scatter plot with the data. This can be done using the `abline()` function, which plots a line given its offset ("a") and slope ("b"). We can either specify both parameters explicitly, or simply supply the result `m` of the fitting procedure: the `abline` function can use either.

```
> abline(m)
> abline( mean(d$Height), 0, lty=2 )
```

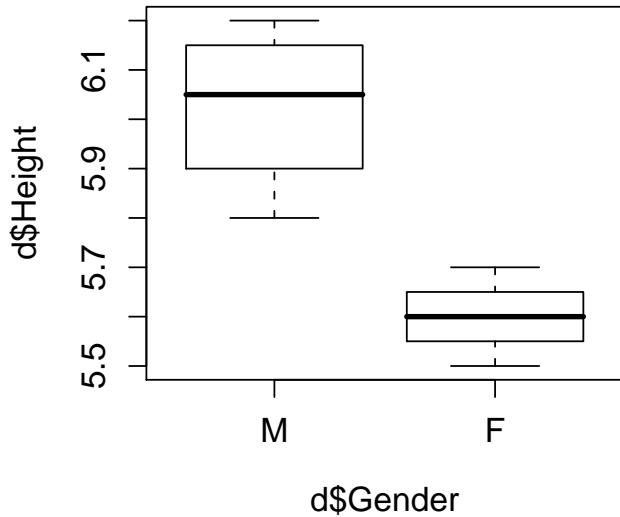


Figure 10.7: TBD

This short example should have given you an idea what working with R is like.

R can be difficult to pick up: it uses some unfamiliar idioms (such as creative indexing), and some obscure function and parameter names. But the greatest challenge to the newcomer (in my opinion) is its indiscriminate use of function overloading. The same function can behave quite differently, depending on the (usually opaque) type of inputs it is given. If the default choices the system makes are good, this can be very convenient, but it can be hellish if you would like to exercise greater, manual control.

Look at our example again: the same `plot()` command generates entirely different plot *types*, depending on whether the control variable is categorical or numeric (box plot in the first case, scatter plot in the latter). In chapter 14, we will see another example, where the same `plot()` command will generate yet a different plot, depending on its input. The problem is that all these different plot types take different optional arguments (such as the `xlab`, `ylab` options used in the example), and it can be very difficult to predict which actual implementation of the generic `plot()` interface is being called at any given moment.

These kinds of issues do not matter much if you use R interactively (since you see the results immediately or in the worst case get an error message, so

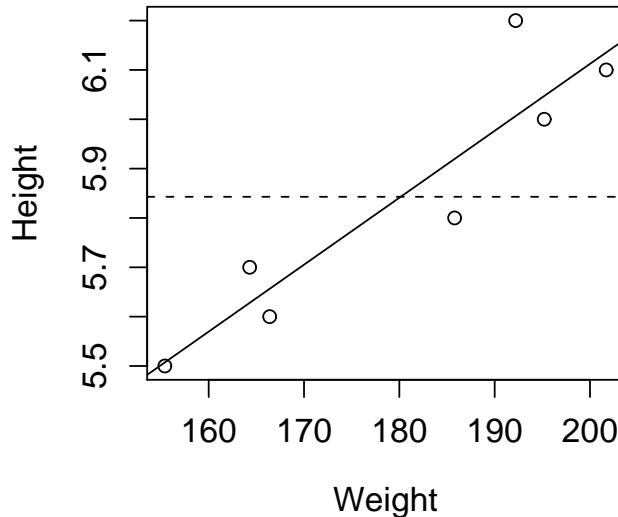


Figure 10.8: TBD

that you can try something else), but can be unnerving if you approach R with the mindset of a contemporary programmer, who prefers for operations to be explicit. It can also be difficult to find which operations are available in a given situation. For instance, it is not at all obvious that the (opaque) return type of the `lm()` function is admissible input to the `abline()` function — it sure does not look like the explicit set of parameters that we use in the second call to `abline()`. These kinds of issues make it hard to predict what R will do at any point and to develop a comprehensive understanding of its capabilities and how to achieve a desired effect in a specific situation.

10.7 Further Reading

The number of introductory statistics texts seems almost infinite — which makes it that much harder to find good ones. Below are some texts that I personally have found useful.

- *An Introduction to Mathematical Statistics and Its Applications*. Richard J. Larsen and Morris L. Marx. 4th ed., Prentice Hall. 2005.
This is my preferred introductory text for the mathematical background

of classical statistics: how it all works. This is a math book, you won't learn how to *do* practical statistical field work from it. (It contains a large number of uncommonly interesting examples; however, on close inspection many of them exhibit serious flaws in their experimental design — at least as described in this book.) But as a mathematical treatment, it very neatly blends accessibility with sufficient depth.

- *Statistics for Technology: A Course in Applied Statistics*. Chris Chatfield. 3rd ed., Chapman & Hall/CRC. 1983.

This book makes a good companion to the book by Larsen/Marx. It eschews most mathematical development and instead concentrates on the pragmatics of it, with an emphasis on engineering applications.

- *The Statistical Sleuth: A Course in Methods of Data Analysis*. Fred Ramsey and Daniel Schafer. 2nd ed., Duxbury Press. 2001.

This advanced undergraduate textbook emphasizes the distinction between observational studies and controlled experiments more than any other book I am aware of. After working through some of their examples, you will not be able to look at the description of a statistical study without immediately classifying it as observational or controlled and questioning the conclusions if it was merely observational. Unfortunately, the development of the general theory gets lost in the detailed description of application concerns.

- *The Practice of Business Statistics*. David S. Moore, George P. McCabe, William M. Duckworth, Layth Alwan. 2nd ed., W. H. Freeman. 2008.

This is a "for business" version of a very popular beginning undergraduate text book. The coverage of topics is comprehensive and the presentation is particularly easy to follow. This book can serve as a first course, but will probably not provide sufficient depth to develop proper understanding.

- *Problem Solving: A statistician's guide*. Chris Chatfield. 2nd ed., Chapman & Hall/CRC. 1995.

and *Statistical Rules of Thumb*. Gerald van Belle. 2nd ed., Wiley-Interscience. 2008.

Two nice books with lots of practical advice on statistical field work. Chatfield's book is more general; van Belle's book contains much material specific to epidemiology and related applications.

- *All of Statistics: A Concise Course in Statistical Inference*. Larry Wasserman. Springer. 2004.

A thoroughly modern treatment of mathematical statistics, presenting all kinds of fascinating and powerful topics that are sorely missing from the standard introductory curriculum. The treatment is advanced and very condensed, requiring general previous knowledge in basic statistics and a solid grounding in mathematical methods.

- *Bayesian Methods for Data Analysis*. Bradley P. Carlin, Thomas A. Louis. 3rd ed., Chapman & Hall. 2008.

This is a book specifically on Bayesian methods, applied to data analysis problems (as opposed to Bayesian theory only). It is a thick book, and some of the topics are pretty advanced. But the early chapters provide the best introduction to Bayesian methods that I am aware of.

- *Sifting the evidence — what's wrong with significance tests?*. Jonathan A. C. Sterne, George Davey Smith. British Medical Journal v322(7280), p226231. 2001.

This paper provides a penetrating and unpartisan overview over the problems associated with classical hypothesis tests, with an emphasis on applications in medicine (but the conclusions are much more generally valid). The full text is freely available on the Web; a search will turn up multiple locations.

260CHAPTER 10. WHAT YOU REALLY NEED TO KNOW ABOUT CLASSICAL STATISTICS

```

> d <- read.csv( "data", header = TRUE, sep = "\t" )
> str(d)
'data.frame': 7 obs. of 4 variables:
 $ Name : Factor w/ 7 levels "Bill","Jack",...: 5 3 6 4 1 7 2
 $ Height: num 6.2 5.5 5.7 5.6 5.8 6.1 6
 $ Weight: num 192 155 164 166 186 ...
 $ Gender: int 0 1 1 1 0 0 0
>
> mean( d$Weight )
[1] 180.1429
> mean( d[,3] )
[1] 180.1429
>
> mean( d$Weight[ d$Gender == 1 ] )
[1] 162.0333
> mean( d$Weight[ 2:4 ] )
[1] 162.0333
>
> d$Diff <- d$Height - mean( d$Height )
> print(d)
  Name Height Weight Gender      Diff
1  Joe     6.2   192.2    0  0.35714286
2 Jane     5.5   155.4    1 -0.34285714
3 Mary     5.7   164.3    1 -0.14285714
4 Jill     5.6   166.4    1 -0.24285714
5 Bill     5.8   185.8    0 -0.04285714
6 Pete     6.1   201.7    0  0.25714286
7 Jack     6.0   195.2    0  0.15714286
> summary(d)
  Name      Height      Weight      Gender      Diff
Bill:1 Min. :5.500 Min. :155.4 Min. :0.0000 Min. :-3.429e-01
Jack:1 1st Qu.:5.650 1st Qu.:165.3 1st Qu.:0.0000 1st Qu.:-1.929e-01
Jane:1 Median :5.800 Median :185.8 Median :0.0000 Median :-4.286e-02
Jill:1 Mean   :5.843 Mean   :180.1 Mean   :0.4286 Mean   : 2.538e-16
Joe :1 3rd Qu.:6.050 3rd Qu.:193.7 3rd Qu.:1.0000 3rd Qu.: 2.071e-01
Mary:1 Max.   :6.200 Max.   :201.7 Max.   :1.0000 Max.   : 3.571e-01
Pete:1
>
> d$Gender <- factor( d$Gender, labels = c("M", "F") )
> summary(d)
  Name      Height      Weight      Gender      Diff
Bill:1 Min. :5.500 Min. :155.4 M:4   Min. :-3.429e-01
Jack:1 1st Qu.:5.650 1st Qu.:165.3 F:3   1st Qu.:-1.929e-01
Jane:1 Median :5.800 Median :185.8          Median :-4.286e-02
Jill:1 Mean   :5.843 Mean   :180.1          Mean   : 2.538e-16
Joe :1 3rd Qu.:6.050 3rd Qu.:193.7          3rd Qu.: 2.071e-01
Mary:1 Max.   :6.200 Max.   :201.7          Max.   : 3.571e-01
Pete:1
>
> plot( d$Height ~ d$Gender )
> plot( d$Height ~ d$Weight, xlab="Weight", ylab="Height" )
> m <- lm( d$Height ~ d$Weight )
> print(m)

Call:
lm(formula = d$Height ~ d$Weight)

```

i

Chapter 11

Intermezzo: Mythbusting — Bigfoot, Least Squares, and All That

Everybody has heard of Bigfoot, the mystical figure that lives in the woods, but nobody has ever actually seen him. Similarly, there are some concepts from basic statistics that everybody has heard of, but that — like Bigfoot — always remain a little shrouded in mystery. Here, we take a look at three of them: the average of averages, the mystical standard deviation, and the ever-popular least squares.

11.1 How to Average Averages

Recently somebody approached me with the following question: given the numbers in table 11.1, what number should be entered into the bottom right hand corner? Just adding up the individual defect rates per item and dividing by three (in effect, averaging them) did not seem right — if only because it would come out to about 0.5, which is pretty high, given that *most* of the units produced (100 out of 103) are not defective, after all. The specific question asked was: “Should I weight the individual rates somehow?”

This situation comes up a lot, but is not always recognized: we have a set of rates (or averages), and would like to summarize them into an overall rate (or overall average). The problem is that the naive way of doing this (namely to sum up the individual rates and to divide by their number) will give an *incorrect* result. But unless the numbers involved are as extreme as in the present example, this is rarely noticed.

The correct way to approach this task is to start from scratch. What is the “defect rate” anyway? It is the number of defective items, divided by the number of items produced. Hence, the *total* defect rate is the total number of de-

Item Type	Units Produced	Defective Units	Defect Rate
A	2	1	0.5
B	1	1	1.0
C	100	1	0.01
Total Defect Rate			???

Table 11.1: Defect rates: what value should go into the bottom righthand corner?

fective items, divided by the total number of items produced: $3/103 \approx 0.03$. There should be no question about that.

Can we arrive at this result in a different way, by starting with the individual defect *rates*? Absolutely — provided we weight them appropriately. Each individual defect rate should contribute to the overall defect rate in the same way that the corresponding item type contributes to the total item count. In other words, the weight for item type A is $2/103$, the weight for B is $1/103$, and finally for C it is $100/103$. Pulling all this together, we have: $0.5 \cdot 2/103 + 1.0 \cdot 1/103 + 0.01 \cdot 100/103 = (1 + 1 + 1)/103 = 3/103$ as before.

To show that this agreement is not accidental, let's write things out in greater generality.

$$\begin{aligned} n_k & \quad \text{Number of items of type } k \\ d_k & \quad \text{Number of defective items of type } k \\ \epsilon_k = \frac{d_k}{n_k} & \quad \text{Defect rate for type } k \\ f_k = \frac{n_k}{\sum_k n_k} & \quad \text{Contribution of type } k \text{ to total production} \end{aligned}$$

Now look at what it means to weight each individual defect rate:

$$\begin{aligned} f_k \epsilon_k &= \frac{n_k}{\sum_k n_k} \frac{d_k}{n_k} \\ &= \frac{d_k}{\sum_k n_k} \end{aligned}$$

In other words, weighting the individual defect rate ϵ_k by the appropriate weight factor f_k has the effect of turning the defect *rate* back to the the defect *count* d_k (normalized by total number of items).

In this example, each item could get only one of two “grades”, namely 1 (for defective) and 0 (for not defective), so that the “defect rate” was a measure of the “average defectiveness” of a single item. The same logic as demonstrated applies if you have a greater (or different) range of values. (You can make up your own example and give items grades from 1 to 5 and calculate the overall “average grade” to see how it works.)

	Male	Female	Overall
Department A	$80/100 = 0.8$	$9/10 = 0.9$	$89/110 = 0.81$
Department B	$5/10 = 0.5$	$60/100 = 0.6$	$65/110 = 0.59$
Total	$85/110 = 0.77$	$69/110 = 0.63$	

Table 11.2: Simpson's Paradox: applications and admissions per applicant's gender

11.1.1 Simpson's Paradox

Since we are talking about mystical figures that sometimes can be found in tables, we should also mention *Simpson's Paradox*. Look at table 11.2, showing applications and admissions to some (fictional) college, broken down by the applicants' gender.

If you just check the bottom line with the totals, it may appear that the college is discriminating against women: the acceptance rate for male applicants is higher than the one for female applicants (0.77 to 0.63).¹ But when you look at the rates for each individual department within the college it turns out that women have *higher* acceptance rates than mean for *every* department. How can that be?

The short and intuitive answer is that many more women apply to department B, which has a lower overall admission rate than department A (0.59 to 0.81), and that drags down their (gender-specific) acceptance rate.

The more general explanation speaks of a "reversal of association due to a confounding (lurking) factor". When just considering the totals, it may appear as if there is an association between gender and admission rates, with male applicants being accepted more frequently. However, this view ignores the presence of a hidden, but important, factor, namely the choice of department. In fact, the choice of department has a *greater* influence on the acceptance rate than the original explanatory variable (the gender, in this case). By lumping the observations for the different departments into a single number, we have in fact masked the influence of this factor, with the consequence that the association between acceptance rate (which favors women for each department) and gender was reversed.

The important insight here is that this kind of "reversal of association" due to a confounding factor is always possible. However, both conditions must occur: the confounding factor must be sufficiently strong (in our case: the acceptance rates for departments A and B were sufficiently different), and the assignment of experimental units to the levels of this factor must be sufficiently imbalanced (in our case: many more women applied to department B than to department A).

As opposed to Bigfoot, Simpson's paradox is known to occur in the real-

¹You should check that the entries in the bottom row have been calculated properly, per the discussion in the preceding section!

world. The example in this section, for instance, was based on a well-publicized case involving the University of California (Berkeley) in early 1970s. A quick Internet search will turn up additional examples.

11.2 The Standard Deviation

The fabled standard deviation is another close relative of Bigfoot. Everybody (it seems) has heard of it, everybody knows how to calculate it, and — most importantly — everybody knows that 68% of all data points fall into one standard deviation, 95% within two, and basically all (that is: 99.7%) within three.

Problem is: this is utter nonsense.

It is true that the standard deviation is a measure for the spread (or width) of a distribution. It is also true that for a given set of points, it can always be calculated. But that does not mean that the standard deviation is always a *good* or appropriate measure for the width of a distribution: in fact, it can be quite misleading if applied indiscriminately to an unsuitable data set. Furthermore, we have to be careful in interpreting it: the whole 68% business only applies if the data set fulfills some very specific requirements.

In my experience, the standard deviation is probably the single most misunderstood and misapplied quantity in all of statistics.

Let me tell you a true story (some identifying details changed to protect the guilty). The story is a bit involved, but this is no accident: in the same way that Bigfoot sightings never occur in a suburban front yard on a sunny Sunday morning, severe misunderstandings in mathematical or statistical methods usually don't reveal themselves as long as the applications are as clean and simple as the homework problems in a textbook. But once people try to apply these same methods in situations that are a bit less standard, *anything* can happen. Here is what happened in this particular company...

I was looking over a bit of code, which was used to identify outliers in the response times from a certain database server. The purpose of this program was to detect and report on uncommonly slow responses. The piece of code in question processed log files containing the response times and reported a threshold value: responses that took longer than this threshold were considered "outliers".

As part of a service-level agreement, it had been defined earlier that any value "outside of three standard deviations" was to be considered an outlier. So, what did this piece of code do? It sorted the response times to identify the top 0.3% of data points and used those to determine the threshold. (In other words, if there were 1000 data points in the log file, it reported the response time of the third-slowest as threshold.) After all: top 0.3% = 3 standard deviations. Right?

After reading chapter 2, I hope you can immediately tell where the original programmer had gone wrong: the threshold that the program reported had *nothing at all* to do with standard deviations — instead, it reported the top 0.3-percentile. So, the program completely failed to do what it was supposed (and

I am sure, intended) to do. Also, keep in mind that it is *incorrect* to blindly consider the top x percent of any distribution as outliers (see the discussion of box plots in chapter 2 if you need a reminder why).

But the story continues. This was a database server, with typical response times less than a few seconds. It was clear that anything that took longer than one or two minutes had to be considered “slow”—that is: an outlier. But when the program was run, the threshold value it reported (the 0.3-percentile), was on the order of *hours*. Clearly, this threshold value did not make any sense.

In what must have been a growing sense of desperation, the original programmer now made a number of changes: from selecting the top 0.3%, to the top 1%, then the top 5% and finally the top 10%. (I could tell, because each such change had dutifully been checked into source control!) Finally, the programmer had simply fixed some “reasonable” seeming value (such as 47 seconds or something), and that’s what the program reported as “three standard deviations”, regardless of input.

It was the only case of outright technical fraud that I have ever witnessed: a technical work product, which — to the original author’s full knowledge — in no way did what it claimed to do.

What went wrong here? Several things. First, there was a fundamental misunderstanding regarding the definition of the standard deviation (and how it is calculated) on the one hand, and some of the properties that in practice it often (but not always) has. The second mistake was the application of the standard deviation to a situation where it is not a suitable measure.

Let’s recap some basics: we often want to characterize a point distribution by a typical value (its location) and its spread around this location. A *convenient* measure for the location is the mean: $\mu = \frac{1}{n} \sum_i^n x_i$. Why is the mean so convenient? Because it is easy to calculate: just sum all the values and divide by n .

To find the spread of the distribution, we would like see how far points “typically” stray from the mean. In other words, we would like the *mean* of the *deviations* $x_i - \mu$. But since the deviations can be positive and negative, they would simply cancel, and so instead we calculate the mean of the *squared* deviations: $\sigma^2 = \frac{1}{n} \sum_i^n (x_i - \mu)^2$. This quantity is called the *variance*, and its square root is the *standard deviation*. Why do we bother with the square root? Because it has the same units as the mean, whereas the variance has the units raised to the second power.

Now, *if and only if* the point distribution is well-behaved (and that means in practice: Gaussian), *then* it is true that about 68% of points will fall into the interval $[\mu - \sigma, \mu + \sigma]$, and 95% into the interval $[\mu - 2\sigma, \mu + 2\sigma]$ and so on. The inverse is *not* true: you cannot conclude that 68% of points defines a “standard deviation” (this is where the programmer in our story had made the first mistake). If the point distribution is not Gaussian, there are no particular patterns what fraction of points will fall within one, two, or any number of standard deviations from the mean. However, keep in mind that the definition of both the mean and the standard deviation (as given by the equations above)

retain their meaning: you can calculate them for any distribution and any data set.

However (and this is the second mistake that was made), if the distribution is strongly asymmetrical, then mean and standard deviations are no longer good measures of location and spread, respectively. You can still *calculate* them, but their values are just not going to be very informative. In particular, if the distribution has a fat tail, then both mean and standard deviation will be influenced heavily by extreme values in the tail.

In the concrete example, the situation was even worse than this: the distribution of response times was a *power law* distribution, which is extremely poorly summarized by quantities such as mean and standard deviation. This explains why the top 0.3% of response times were on the order of hours: with power law distributions, all values, even extreme ones, can (and do!) occur, whereas for Gaussian or exponential distributions, the range of values that do occur in practice is pretty strictly limited. (See chapter 9 for a more information about power law distributions.)

To summarize, the standard deviation, defined as $\sqrt{\frac{1}{n} \sum_i^n (x_i - \mu)^2}$ is a measure for the width of a distribution (or a sample). It is a good measure for the width only if the distribution of points is well-behaved (symmetric and without fat tails). Points that are far away from the center (compared to the width of the distribution) can be considered outliers. For distributions that are less well-behaved, you will have to use other measures for the width (such as the inter-quartile range), but you can usually still identify outliers as points that fall outside the “typical” range of values. (For power law distributions, which don’t have a “typical” scale, it does not make sense to define outliers by statistical means, and you will have to justify them differently, for instance by appealing to requirements from the business domain.)

11.2.1 How to calculate

Here is a good trick to calculate the standard deviation efficiently. At first glimpse, it appears that you need to make two passes over the data, to calculate both mean and standard deviation: in the first pass, you calculate the mean, but then you need to make a second pass to calculate the deviations from that mean:

$$\sigma^2 = \frac{1}{n} \sum (x_i - \mu)^2$$

and it seems as if you can’t find the deviations, until the mean μ is known.

However, it turns out that you can calculate both quantities in a single pass through the data — all you need to do is to maintain both the sum of the values ($\sum x_i$) and the sum of the squares of the values ($\sum x_i^2$), because you can write the above equation for σ^2 in a form that depends only on those two sums:

$$\begin{aligned}
\sigma^2 &= \frac{1}{n} \sum (x_i - \mu)^2 \\
&= \frac{1}{n} \sum (x_i^2 - 2x_i\mu + \mu^2) \\
&= \frac{1}{n} (\sum x_i^2 - 2\mu \sum x_i + \mu^2 \sum 1) \\
&= \frac{1}{n} \sum x_i^2 - 2\mu \frac{1}{n} \sum x_i + \mu^2 \frac{1}{n} n \\
&= \frac{1}{n} \sum x_i^2 - 2\mu \cdot \mu + \mu^2 \\
&= \frac{1}{n} \sum x_i^2 - \mu^2 \\
&= \frac{1}{n} \sum x_i^2 - \left(\frac{1}{n} \sum x_i \right)^2
\end{aligned}$$

This is a good trick, which is apparently too little known. Keep it in mind — similar situations crop up in different contexts from time to time.

(To be sure, the floating point properties of both methods are of course different, but if you care enough that you worry about the difference, you will be using a library anyway.)

11.2.2 One over What?

You may occasionally see the standard deviation defined with an n in the denominator, and sometimes with a factor of $(n - 1)$ instead.

$$\sqrt{\frac{1}{n} \sum_i^n (x_i - \mu)^2} \quad \text{or} \quad \sqrt{\frac{1}{n-1} \sum_i^n (x_i - \mu)^2}$$

What *really* is the difference, and which one should you use?

The factor $1/n$ only applies if you know the exact value of the mean μ ahead of time. This is usually not the case, instead, you will usually have to calculate the mean from the data. This adds a bit of uncertainty, which leads to the widening of the proper estimate for the standard deviation. A theoretical arguments leads to the use of the factor $1/(n - 1)$ instead of $1/n$.

In short: if you calculated the mean from the data (as is usually the case), you should really be using the $1/(n - 1)$ factor. The difference is going to small, unless you are dealing with very small data sets.

11.2.3 The Standard Error

And while we are on the topic of dealing with obscure sources of confusion, let's talk about the *standard error*.

The standard error is the standard deviation of an estimated quantity. Let's say we estimate some quantity (for example, the mean). If we repeatedly take samples, then

the means calculated from those samples will scatter around a little, according to some distribution. The standard deviation of this distribution is the “standard error” of the estimated quantity (the mean, in this example).

The following observation will make this more clear:

Take a sample of size n from a normally distributed population with standard deviation σ . Then, 68% of members of the *sample* will be within $\pm\sigma$ from the estimated mean (that is: the sample mean).

However, the mean itself is normally distributed (because of the Central Limit Theorem: the mean is a sum of random variables), with standard deviation σ/\sqrt{n} (again because of the Central Limit Theorem). Hence, if we take several samples, each of size n , then we can expect 68% of the estimated means to lie within $\pm\sigma/\sqrt{n}$ of the *true* mean (that is: the mean of the overall population).

In this situation, the quantity σ/\sqrt{n} is therefore the *standard error or the mean*.

11.3 Least Squares

Everybody loves least squares. In the confusing and uncertain world of data and statistics, they provide a sense of security — something to rely on! They give you, after all, the “best” fit. Doesn’t that say it all?

Problem is, I have *never* — not once! — seen least squares applied appropriately, and I have come to doubt that it should ever be considered a suitable technique. In fact, when I see somebody doing anything involving “least-squares fitting” today, I am pretty certain that this person is at wits end — and probably does not even know it!

There are two problems with least squares. The first is that it is used for two very different purposes, which are commonly confused. The second problem is that least squares fitting is usually not the best (or even merely a suitable) method for either purpose, and different techniques should be used — which one, depends on the overall purpose (see first problem), and on what, in the end, we want to do with the result.

Let’s try to unravel these issues.

Why do we ever want to “fit” a function to data to begin with? There are two different reasons:

Statistical Parameter Estimation Data is corrupted by random noise, and we want to extract parameters from it.

Smooth Interpolation or Approximation Data is given as individual points, and we would like to find a smooth interpolation to arbitrary positions between those points, or determine an analytical “formula” describing the data.

The two scenarios are conceptually depicted in figures 11.1 and 11.2.

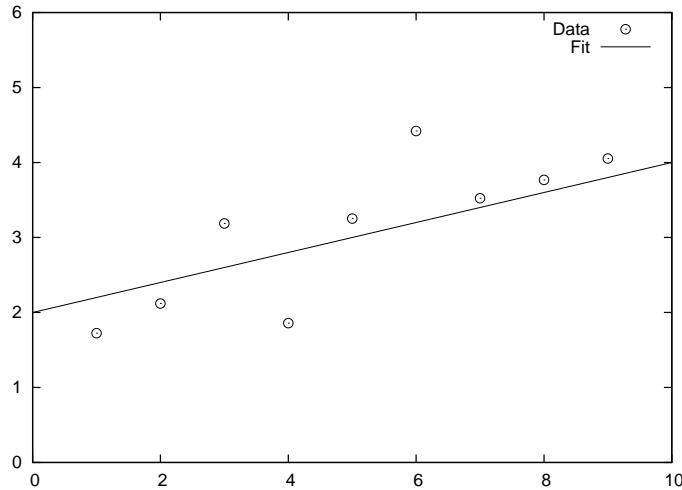


Figure 11.1: Fitting for statistical parameter estimation: data affected by random noise. What is the slope of the straight line?

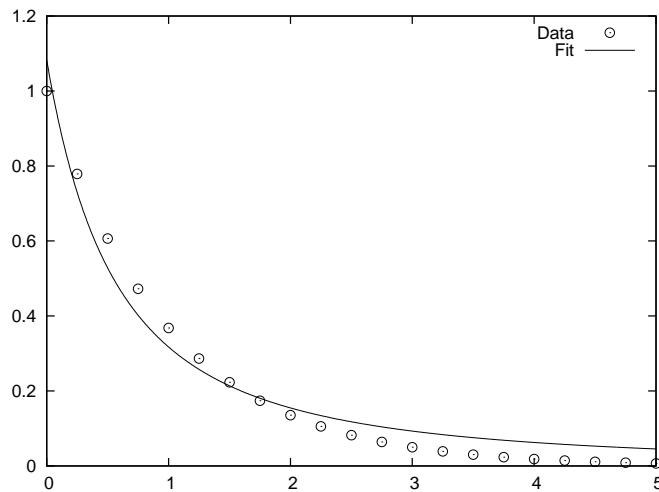


Figure 11.2: Fitting a function to approximate a curve known only at discrete locations. Is the fit a good representation of the data?

11.3.1 Statistical Parameter Estimation

Statistical parameter estimation is the more legitimate of the two. In that case, we have a control variable x and an outcome y — we set the former and measure the latter, ending up with a data set of pairs: $\{(x_1, y_1), (x_2, y_2), \dots\}$. Furthermore, we assume that the outcome is related to the control variable through some function $f(x; \{a, b, c, \dots\})$ of known form, which depends on the control variable x and also on a set of (initially unknown) parameters $\{a, b, c, \dots\}$. However, in practice, the actual measurements are affected by some random noise ϵ , so that the measured values y_i are a combination of the “true” value and the noise term:

$$y_i = f(x_i, \{a, b, c, \dots\}) + \epsilon_i$$

We now ask: how should we choose values for the parameters $\{a, b, c, \dots\}$, such that the function $f(x, \{a, b, c, \dots\})$ reproduces the measured values of y most faithfully? The usual answer is that we want to choose the parameters such that the *total mean-squared error* E^2 (sometimes called the *residual sum of squares*):

$$E^2 = \sum_i (f(x_i, \{a, b, c, \dots\}) - y_i)^2$$

is minimized. As long as the distribution of errors is reasonably well-behaved (not too asymmetric and without heavy tails), the results are adequate. If in addition the noise is Gaussian, we can even make contact with other parts of statistics and show that the estimates for the parameters that we obtain through the least squares procedure agree with the results we would have gotten through a “maximum likelihood estimate”. So, the least squares results are consistent with alternative ways of calculation.

But there is another very important aspect to least squares estimation that is frequently lost: we can not only obtain *point estimates* for the parameters $\{a, b, c, \dots\}$, but we can also find *confidence intervals* for them as well, through a self-consistent argument that links the distribution of the parameters to the distribution of the measured values.

I cannot stress this enough: a point estimate by itself is of limited use: what good is it if I know that the point estimate for a is 5.17, if I have no idea whether this means $a = 5.17 \pm 0.01$ or $a = 5.17 \pm 250$? We *must* have some way of judging the range over which we expect our estimate to vary, which is the same as finding a confidence interval for it. Least squares works, when applied in a probabilistic context like this, because it not only gives us an estimate for the parameters, but also for their confidence intervals.

One last point: in statistical applications, you rarely have to do the minimization of E^2 by numerical means. For most of the function $f(x, \{a, b, c, \dots\})$ that are commonly used in statistics, the conditions that will minimize E^2 can be worked out explicitly. (See chapter 3 for the results in the case that the function is linear.) In general, you should be reluctant to resort to numerical minimization procedures — there might be better ways of obtaining the result.

11.3.2 Function Approximation

In practice, though, least squares fitting is often used for a different purpose. Consider the situation in figure 11.2, where we have a set of individual data points. These points clearly fall on a smooth curve, which we don't know. But it would be more convenient if we had an explicit formula to summarize these data points, rather than having to work with the collection of points directly. So, can we "fit" a formula to them?

Notice that in this second application of least squares fitting there is *no random noise*, in fact, there is no random component at all! This is an important insight, because it implies that statistical methods and arguments don't apply.

This becomes relevant when we want to determine the degree of confidence that we have in the results of a fit. Let's say we have performed a least squares routine and obtained some values for the parameters. What confidence intervals should we associate with the parameters and how good is the overall fit? Whatever errors we may incur in the fitting process, they are not going to be of a random nature, and we therefore cannot make probabilistic arguments about them.

The scenario in figure 11.2 is typical: the plot shows the data together with the best fit for a function of the form $f(x; a, b) = a/(1 + x)^b$, with $a = 1.08$ and $b = 1.77$. Is this a good fit? And what uncertainty do we have in the parameters? That depends on what you want to do with the results — but notice that the deviations between the fit and the data are not at all "random" and that therefore statistical "goodness of fit" measures are inappropriate. We have to find other ways to answer our questions. (For instance, we may find the largest of the residuals between the data points and our fitted function and report that the fit "represents the data with a maximum deviation of...".)

This situation is typical in yet another way: given how smooth the curve is that the data points seem to fall on, our "best fit" seems really *bad* — just intuitively speaking. Is this really the best we can do? The problem here is that the least squares approach forces us to specify the functional form of the function we are attempting to fit, and if we get it wrong, the results won't be any good. For this reason, we should use less constraining approaches (such as non-parametric or local approximations), unless we have *strong* reasons to favor one particular functional form.

In other words, what we really have here is a problem of function interpolation or approximation: we know the function on a discrete set of points, and we would like to extend it smoothly to all values. How we should do this depends on what we want to do with the results. Here is some advice for common scenarios:

- To find a "smooth curve" for plotting purposes, you should use one of the smoothing routines we discussed in chapter 3, such as splines or LOESS. These "non-parametric" methods have the advantage that they do not impose a particular functional form on the data (in contrast to the situation in figure 11.2).

- If you want to be able to evaluate the function easily at an arbitrary location, you should use a local interpolation method. Such methods build a local approximation, using the three or four data points closest to that desired location. It is not necessary to find a global expression in this case: the local approximations will suffice.
- Sometimes you may want to summarize the behavior of the data set in just a few “representative” values (for example so that you can compare one data set more easily against another). This is tricky — it is probably a better idea to compare data sets *directly* against each other, using similarity metrics such as those discussed in chapter 13. If you still need to do this, consider a *basis function expansion*, using Fourier, Hermite, or Wavelet functions. (These are special sets of functions, which allow you to extract greater and greater amounts of detail from a data set. Expansion in basis functions also allows you to evaluate and improve the quality of the approximation in a systematic fashion.)
- At times you might be particularly interested in some feature of the data: for example, you suspect the data to follow a power law x^b and you would like to extract the coefficient. Or the data is periodic and you need to know the length of one period. In such cases, it is usually a better idea to transform the data in such a way that you can obtain that particular directly, rather than fitting a global function. (To extract exponents, you should consider a logarithmic transform; while measuring the peak-to-peak (better: the zero-to-zero) distance can be a remarkably effective way to obtain the length of an oscillatory period, and so on.)
- Finally, use specialized methods if available and applicable. Time series, for instance, should be treated with the techniques we discussed in chapter 4.

You may have noticed that none of these involves least squares...

Part III

Computation: Mining Data

Chapter 12

Simulation

Coming soon

Licensed by
Shahriyar Matloub
1969616

Chapter 13

Clustering

The term *clustering* refers to the process of finding groups of points within a data set which are in some way “lumped together”. It is also called *unsupervised learning* — unsupervised, because we don’t know ahead of time where the clusters are located and what they look like. (In contrast to supervised learning or classification, where we attempt to assign data points to pre-existing classes. See chapter 18.)

I regard clustering as an *exploratory* method: a computer-assisted (or even: computationally driven) approach to discover structure in a data set. As an exploratory technique, it usually needs to be followed by a confirmatory analysis, which validates the findings and makes them more precise.

Clustering is a lot of fun: it is a very rich topic, with a wide variety of different problems (as we will see in the next section, when we discuss the different *kinds* of cluster one may encounter). The topic also has a lot of intuitive appeal, and most clustering methods are rather straightforward, which allows for all kinds of ad-hoc modifications and enhancements, in response to the specific problem one is working on.

13.1 What constitutes a Cluster?

Clustering is not a very rigorous field: there are precious few established results, rigorous theorems, or algorithmic guarantees. In fact, the whole notion of a “cluster” is not particularly well-defined. Descriptions such as points that are “similar” or “close to each other” are insufficient — clusters must also be *well separated* from each other. Look at figure 13.1: some points are certainly closer to each other than to other points, yet there are no discernible clusters. (In fact, it is an interesting exercise to define what constitutes the *absence* of clusters.) This leads to one possible definition of clusters, as *contiguous regions of high data point density, separated by regions of lower point density*. Although not particularly rigorous either, it does seem to capture the essential elements of typical clusters. (For a different point of view, see below.)

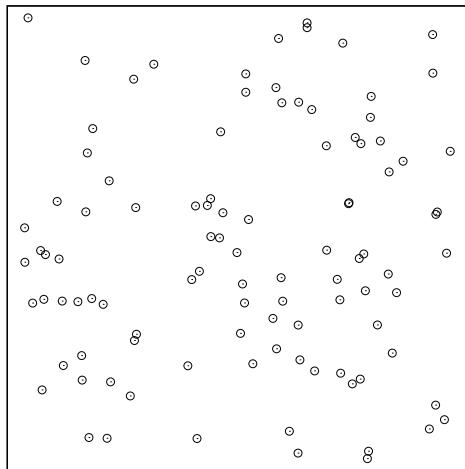


Figure 13.1: TBD

The definition just given allows for very different kinds of clusters. Figures 13.2 and 13.3 show two very different types. Of course, figure 13.2 is the “happy” case, showing a data set consisting of well-defined and clearly separated regions of high data point density. The clusters in figure 13.3 are of a different type of cluster, which is more easily thought of by means of nearest-neighbour (graph) relationships than through point density. Nevertheless, in this case as well, there are higher density regions separated by lower density regions, although we might want to exploit the nearest-neighbour relationship instead of the higher density when coming up with a practical algorithm for this case.

Clustering is not limited to points in space. Figures 13.4 and 13.5 show two rather different cases, for which it nevertheless makes sense to speak of clusters. Figure 13.4 shows a bunch of street addresses. Not two of which are exactly the same — yet if we look closely, we will easily recognize that all of them can be grouped into just a few neighborhoods! Figure 13.5 shows a bunch of different time series: again, some of them are more alike than others. The challenge in both of these examples will be to find a way to express the “similarity” between these non-numeric, non-geometric objects!

Finally, we should keep in mind that clusters may have complicated *shapes*. Figure 13.6 shows two very well behaved clusters, as well separated regions of high point density; yet, the complicated and intertwined shape of the regions will challenge many commonly used clustering algorithms.

A bit of terminology can help to distinguish different cluster shapes: if the line connecting any two points lies entirely within the cluster itself, (as in figure 13.2), then the cluster is *convex*. This is the easiest shape to handle. A cluster

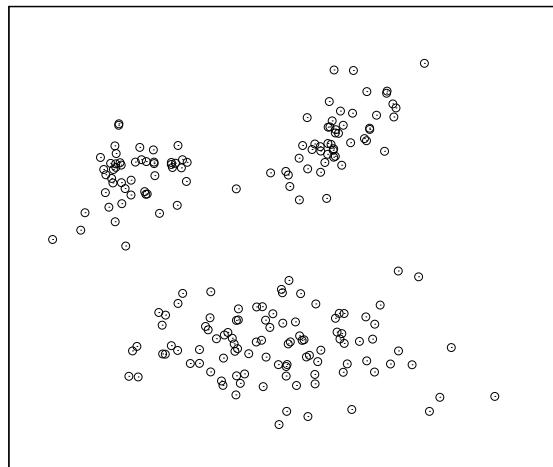


Figure 13.2: TBD

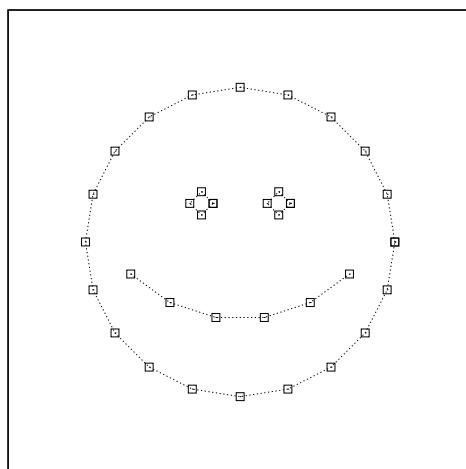


Figure 13.3: TBD

First Avenue 35	44 Secnd Street	First Av 30
48 Second Street	Centrol Place 3	Fst Avenue 23
Urban Court 7	Urban Ct 3	44 2nd Street
Main Boulevard 15	Maine Blvd 12	Ctl Place 12
Central Place 9	Furst Avenue 7	Ctrl Pl 11
First Avenue 17	45 Sec Street	M Boulevrd 17
Urban Court 7a	47 Secnd St	U Ct 8
49 2nd Streent	Mine Boulevard 9	1st Ave 30
First Ave 93	M Blvd 10	Central P 7
Central Pl 2	Urbane Ct 9	Urban C 9
Main Boulevard 8	Urban Courtyard 92	Maine Boulevard 12
1st Ave 39	1st Avenue 41	First Ave 23
first Avenue 30	Main Boulevard 12	45 Second Street
Urbn Ct 9	U Court 9	Central Place 10

Figure 13.4: TBD

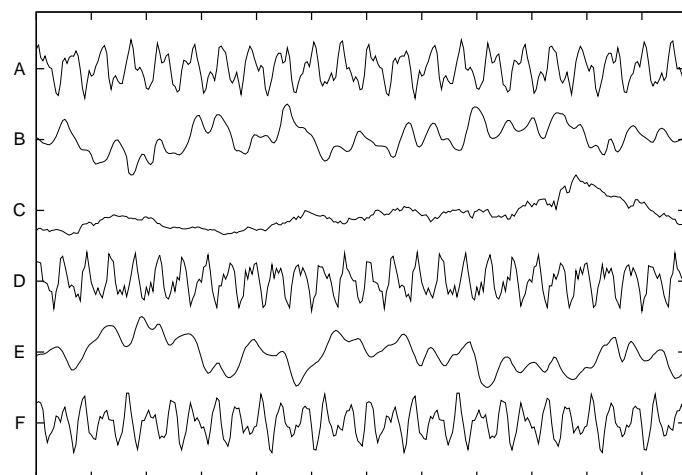


Figure 13.5: TBD

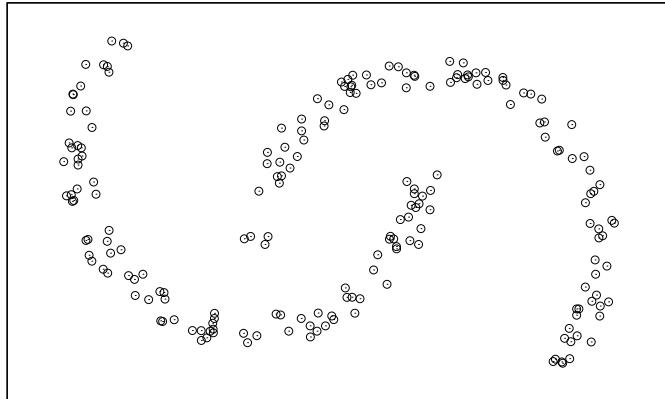


Figure 13.6: TBD

is only convex if the connecting line between two points lies entirely within the cluster for *all* pairs of points. Sometimes this is not the case, but we can still find at least one point (the *center*), such that the connecting line from the center to any other point lies entirely within the cluster: such a cluster is called *star convex*. Notice that the clusters in figure 13.6 are neither convex, nor star convex. Sometimes one cluster is entirely surrounded by another cluster, without actually being part of it: in this case we speak of a *nested* cluster. Nested clusters can be particularly challenging (see figure 13.3).

13.1.1 A different point of view

Without a precise (mathematical) definition of a cluster, a cluster can be whatever we consider as one. That is important, because our minds have a different, alternative way of grouping (“clustering”) objects: not by proximity or density, but by the way objects fit into a larger structure. Figures 13.7 and 13.8 show two examples.

Intuitively, we have no problem to group the points in figure 13.7 into two overlapping clusters. Yet, the density based definition of a cluster we introduced above will not support that. Similarly for the set of points in figure 13.8. The distance between any two adjacent points is the same. But in our minds we observe the larger structures of the vertical and horizontal arrangements, and assign points to clusters based on those.

This notion of a cluster does not hinge on the similarity or proximity of any pair of points to each other, but on the similarity between a point and a property of the *entire cluster*. For any algorithm that considers a single point (or

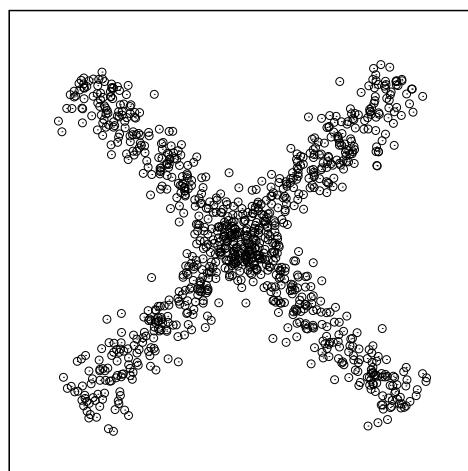


Figure 13.7: TBD

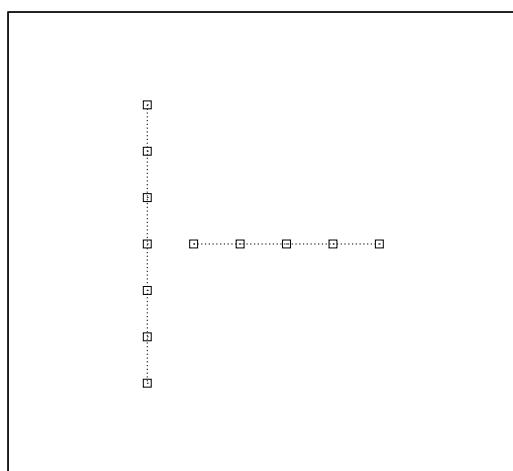


Figure 13.8: TBD

a single pair of points) at a time, this leads to a problem: to determine cluster membership, we need the property of the whole cluster; but to determine the properties of the cluster, we need to assign points to clusters first.

To handle such situations, we would need to perform some kind of global structure analysis — an activity that our minds are incredibly good at (which is why we tend to think of clusters this way), but which we have had a very hard time to teach computers to do it. For problems in two dimensions, *digital image processing* has developed methods to recognize and extract certain features (such as edge detection). But general clustering methods, such as those described in the rest of this chapter, deal only with local properties, and therefore can't handle problems such as those in figures 13.7 and 13.8.

13.2 Distance and Similarity Measures

Given how strongly our intuition about clustering is shaped by geometric problems such as those in figures 13.2 and 13.3, it is an interesting and possibly surprising observation that clustering does not actually require data points to be embedded into a geometric space: all that is required is a *distance* or (equivalently) a *similarity measure* for any *pair of points*. This makes it possible to perform clustering on a set of strings, such as those in figure 13.4, which do not map to points in space. However, if the data points have properties of a vector space (see chapter 1), we can develop more efficient algorithms that exploit these properties.

A *distance* is any function $d(x, y)$ that takes two points and returns a scalar value, which is a measure for how different these points are: the more different, the larger the distance. Depending on the problem domain, it may make more sense to express the same information in terms of a *similarity* function $s(x, y)$, which returns a scalar that tells us how similar two points are: the more different they are, the smaller the similarity. Any distance can be transformed into a similarity and vice versa: for example if we know that our similarity measure s can take on only values in the range $[0, 1]$, then we can form an equivalent distance by setting $d = 1 - s$. In other situations, we might decide to use $d = 1/s$, or $s = \exp(-d)$, and so on. The choice will depend on the problem that we are working on. In the following, I will express problems in terms of either distances or similarities, whatever seems more natural. Just keep in mind that you can always transform between the two.

What we use as a distance function is largely up to us, and we can express different semantics about the data set through the appropriate choice of distance. For some problems, a particular distance measure will present itself naturally (if the data points are points in space, we will most likely employ the Euclidean distance or a distance measure very similar to it), but for other problems we will have more freedom to define our own metric. We will see several examples shortly.

There are certain properties that a distance (or similarity) function should have. Mathematicians have developed a set of properties that a function must

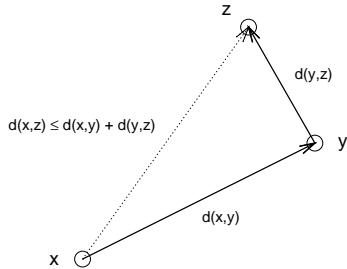


Figure 13.9: TBD

possess to be considered a metric (or distance) in a mathematical sense. These properties can provide valuable guidance, but don't take them too seriously: for our purposes different properties might be more important. The four axioms of a mathematical metric are:

$$\begin{aligned} d(x, y) &\geq 0 \\ d(x, y) &= 0 \quad \text{if and only if } x = y \\ d(x, y) &= d(y, x) \\ d(x, y) + d(y, z) &\geq d(x, z) \end{aligned}$$

The first two say that a distance is always positive and that it only is null if the two points are equal. The third property ("symmetry") states that the distance between x and y is the same as the distance between y and x — no matter which way we consider the pair. The final property is the so-called "triangle inequality", which says that to get from x to z it is never shorter to take a detour through a third point y instead of going directly (see figure 13.9).

This all seems rather uncontroversial, but is not necessarily fulfilled in practice. A funny example for an asymmetric distance occurs if you ask everyone in a group of people how much they like every other member of the group, and use the responses to construct a distance measure: it is not at all guaranteed that the feelings of person A for person B are requited by B ! (Using the same example, it is also possible to construct scenarios that violate the triangle inequality.) For technical reasons, the symmetry property is usually highly desirable. Keep in mind that you can always construct a symmetric distance

function from an asymmetric one:

$$d_S(x, y) = \frac{d(x, y) + d(y, x)}{2}$$

is always symmetric!

One property of great practical importance which is not included in the list above is *smoothness*. For example we might define a particularly simple-minded distance function that is zero if and only if both points are equal to each other, and that is one if the two points are not equal:

$$d(x, y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{otherwise} \end{cases}$$

You can convince yourself that this distance fulfills all four of the distance axioms given above. Yet, it is not particularly helpful, because it gives us no information *how* different two non-identical points are! Most clustering algorithms require this information: a certain kind of tree-based algorithm, for example, works by successively considering the pairs of points with the smallest distance between them — with this choice of distance, the algorithm will make only very limited progress before having exhausted all information available to it.

The practical upshot of this discussion is that a good distance function for clustering should change smoothly as its inputs become more or less similar. (For classification tasks, a binary one as in the example just discussed might be fine.)

13.2.1 Common Distance and Similarity Measures

Depending on the data set and the purpose of our analysis, there are different distance and similarity measures available.

First, let's clarify some terminology. We are looking for ways to measure the distance between any two data points. Very often, we will find that a point has a number of *dimensions* or *features*. (The first usage is more common for numerical data, the latter for categorical data.) In other words, each point is a collection of individual values: $x = \{x_1, x_2, \dots, x_d\}$, where d is the number of dimensions (or features). For example, the data point $\{0, 1\}$ has two dimensions, describing a point in space; whereas the tuple $[\text{'male'}, \text{'retired'}, \text{'Florida'}]$, describing a person, has three features.

For any given data set containing n elements, we can form n^2 pairs of points. The set of the all distances for all possible pairs of points can be arranged in a quadratic table, called the *distance matrix*. The distance matrix embodies all information about the mutual relationships between all points in the data set. (If the distance function is symmetric, as is usually the case, then the matrix is also symmetric. Furthermore, the entries along the main diagonal typically are all zero, since the $d(x, x) = 0$ for most well-behaved distance functions.)

Name	General	In 2 dimensions
Manhattan	$d(x, y) = \sum_i^d x_i - y_i $	$d(x, y) = x_1 - y_1 + x_2 - y_2 $
Euclidean	$d(x, y) = \sqrt{\sum_i^d (x_i - y_i)^2}$	$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$
Maximum	$d(x, y) = \max_i x_i - y_i $	$d(x, y) = \max(x_1 - y_1 , x_2 - y_2)$
Minkowski	$d(x, y) = \left(\sum_i^d x_i - y_i ^p \right)^{1/p}$	$d(x, y) = (x_1 - y_1 ^p + x_2 - y_2 ^p)^{1/p}$
Dot Product	$x \cdot y = \frac{\sum_i^d x_i y_i}{\sqrt{\sum_i^d x_i^2} \sqrt{\sum_i^d y_i^2}}$	$x \cdot y = \frac{x_1 y_1 + x_2 y_2}{\sqrt{x_1^2 + x_2^2} \sqrt{y_1^2 + y_2^2}}$
Correlation Coefficient	$\text{corr}(x, y) = \frac{\sum_i^d (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i^d x_i^2} \sqrt{\sum_i^d y_i^2}}$ $\bar{x} = \frac{1}{d} \sum_i^d x_i ; \quad \bar{y} = \frac{1}{d} \sum_i^d y_i$	$\text{corr}(x, y) = \frac{(x_1 - \bar{x})(y_1 - \bar{y}) + (x_2 - \bar{x})(y_2 - \bar{y})}{\sqrt{\sum_i^d x_i^2} \sqrt{\sum_i^d y_i^2}}$ $\bar{x} = \frac{x_1 + x_2}{2} ; \quad \bar{y} = \frac{y_1 + y_2}{2}$

Table 13.1: TBD

Numerical Data

If the data is numerical and “mixable” or vector-like (in the sense of chapter 1), then the data points bear a strong resemblance to points in space, and we can use a metric such as the familiar *Euclidean Distance*. The Euclidean distance is only the most commonly used example from a large family of related distance measures, which also contains the so-called *Manhattan* (or *Taxicab*) *Distance* and the *Maximum* (or *Supremum*) *Distance*. All of these are in fact special cases of a more general *Minkowski* or *p-Distance*. Table 13.1 shows some examples. (The Manhattan distance is so named because it measures distances the way a New York taxicab moves: at right angles, along the city blocks. The Euclidean distance measures distances the way the crow flies. Finally, it is an amusing exercise to show that the Maximum distance corresponds to the Minkowski metric as $p \rightarrow \infty$.)

All these distance measures have very similar properties, and the differences between them usually do not matter much. The Euclidean distance is by far the most commonly used. I list them here mostly give a flavor for the kind of leeway that exists in defining a suitable distance measure — without significantly affecting the results!

If the data is numeric, but *not* mixable (so that it does not make sense to add a random fraction of one data set to a random fraction of a different data set), then these distance measures are not appropriate. Instead, you may want to consider a metric based on the *correlation* between two data points.

Correlation-based measures are measures of *similarity*: they are large when objects are similar, and zero when the objects are very dissimilar. There are two related measures: the *dot product* and the *correlation coefficient*. The definitions are also given in table 13.1. The only difference is that in the calculation of the correlation coefficient we first center both data points by subtracting their respective means.

In both measures, we multiply entries for the same “dimension” and sum

up the results; then divide by the correlation of each data point with itself. Doing so provides a *normalization* and ensures that the correlation of any point with itself is always 1. This normalization step makes correlation-based distance measures suitable for data sets where different data points may have widely different numeric values.

By construction, the value of a dot product is always in the interval $[0, 1]$ and the correlation coefficient always falls in the interval $[-1, 1]$. You can therefore easily transform either into a distance measure if need be (for example, $1 - d$, where d is the dot product, will be a good distance).

I should point out that the dot product has a geometric meaning: if we regard the data points as vectors in some suitable space, then the dot product of two points is the cosine of the angle that the two vectors make with each other. If they are perfectly aligned (that is, they fall onto each other), then the angle is zero and the cosine (and the correlation) is one. If they are at right angles to each other, the cosine is zero.

Correlation-based distance measures are suitable whenever numeric data is not readily mixable — for instance, to measure the similarity of the time series in figure 13.5.

Categorical Data

If the data is categorical, we can count the number of features that do *not* agree in both data points (that is, the number of mismatched features): this is the *Hamming Distance*. (We might want to divide by the total number of features to obtain a number between zero and one, namely the *fraction of mismatched features*.)

In certain data mining problems, the number of possible features is large, but only relatively few of them will be present for each data point. Moreover, the features may be binary: we only care whether or not they are present, but the values don't matter. (As an example, imagine a patients health record: each possible medical condition constitutes a features, and we want to know whether the patient has ever suffered from it.) In such situations, where features are not just categorical, but binary and sparse (meaning that only few of the features are ON), we may be more interested in matches between features that are ON, compared to matches between features that are OFF. This leads us to the *Jaccard Coefficient* s_J , as the number of matches between features that are ON, divided by the number of features that are set in at least one of the data points. The Jaccard coefficient is a *similarity* measure, the corresponding distance function is the *Jaccard Distance* $d_J = 1 - s_J$.

- n_{00} features that are OFF in both points
- n_{10} features that are ON in the first, and OFF in the second point
- n_{01} features that are OFF in the first, and ON in the second point
- n_{11} features that are ON in both points

$$s_J = \frac{n_{11}}{n_{10} + n_{01} + n_{11}}$$

$$d_J = \frac{n_{10} + n_{01}}{n_{10} + n_{01} + n_{11}}$$

Many other measures of similarity or dissimilarity for categorical data exist, but the principles are always the same: you calculate some fraction of matches, possibly emphasising one aspect (such as presence or absence of certain values) more than others. Feel free to invent your own — as far as I can see, none of these measures has achieved universal acceptance or is fundamentally better than any other.

String Data

If the data consists of strings, we can use a form of Hamming distance and count the number of mismatches. If the strings in the data set are not all of equal length, we can pad the shorter string and count the number of characters added as mismatches.

If we are dealing with many strings that are rather similar to each other (distorted through typos, for instance), then we can use a more detailed measure of the difference between them, namely the *edit* or *Levenshtein distance*. The Levenshtein distance is the minimum number of single-character operations (insertions, deletions, and substitutions) required to transform one string into the other. (A quick Internet search will give many references to the actual algorithm and available implementations.)

Another approach is to find the length of the *longest common subsequence*. This metric is used in particular in gene sequence analysis in computational biology.

Maybe this is a good place to make a more general point: the distance to choose does not follow from the type of the data in some automated way, but depends on the semantics of the data — or, even more precisely, the semantics that you care about for your current analysis! In some instance, a very simple metric which only calculates the difference in string length may be perfectly sufficient. In another one, you might want to use the Hamming distance. If you really care about the details of otherwise very similar strings, the Levenshtein distance is the way to go. Or you might want to calculate how often each letter appears in a string and base your comparison on that... It all depends on what the data means, and what aspect of it you are interested, right now. (Which may also change as your analysis progresses.) Similar considerations apply everywhere — there are no cookbook rules.

Special Purpose Metrics

A very abstract measure for the similarity of two points is related to the number of neighbors the two points have in common; this metric is known as the *shared nearest neighbor* similarity (SNN). To calculate the SNN for two points, x and y , you find the k nearest neighbors (using any suitable distance function) for both x and y . The number of neighbors shared by both points is their mutual SNN.

The same concept can be extended to situation where there is some property that the two points may have in common. For example, in a social network, we can define the “closeness” of two people by the number of friends they share, or the number of movies they have both seen, and so on. (This application is equivalent to the Hamming distance). Nearest-neighbor based metrics are particularly suitable for high-dimensional data, where other distance measures can give spuriously small results.

Finally, let me remind you that sometimes the solution does not consist of inventing a new metric — instead, the trick is to map the problem to a different space, which has a pre-defined, suitable metric already!

As an example, consider the problem of measuring the degree of similarity between different text documents (we here assume that these documents are long — hundreds or thousands of words). The standard approach to this problem is to count how often each word appears in each document. The resulting data structure is referred to as the *document vector*. You can now form a dot product between two document vectors as a measure of their correspondence.

Technically, what has happened here is that we have mapped each document to a point in a (high-dimensional) vector space. Each distinct word that occurs in any of the documents spans a new dimension, and the frequency with which each word appears in a document provides the position of that document along this axis. But this is very interesting, because we now have transformed highly structured data (text) into numerical, even vector-like data, and can therefore now operate on it much more easily! (Of course, the benefit comes at a price: in doing so we have lost all information about the sequence in which words appeared in the text. It is a separate consideration whether this is relevant for our purpose.)

One last comment: one can overdo it when defining distance and similarity measures. Particularly complicated or sophisticated definitions are usually not necessary, as long as you capture the fundamental semantics. The Hamming distance and the document vector correlation are two good examples of simplified metrics which intentionally discard a lot of information, yet which nevertheless turn out to be highly successful in practice.

13.3 Clustering Methods

In this section, I want to present several very different clustering algorithms. As you will see, the basic ideas behind all three algorithms are rather simple,

and it is straightforward to come up with a perfectly adequate implementation of them yourself. These algorithms are also important as starting points for more sophisticated clustering routines, which usually augment them with various heuristics, or combine ideas from different algorithms.

Different algorithms are suitable for a different kinds of problems, depending for example on the shape and structure of the clusters. Some require vector-like data, whereas others only need the distance function. Different algorithms tend to be misled by different kinds of pitfalls, and finally, they all have different performance (that is: computational complexity) characteristics. It is therefore important to have a variety of different algorithms at your disposal, so that you can choose the one most appropriate, both for your problem *and* for the kind of solution you want to get! (Remember that it is pretty much the choice of algorithm which defines what constitutes a “cluster” in the end.)

13.3.1 Center Seekers

One of the most popular clustering methods is the *k-means* algorithm. The *k*-means algorithm requires the number of expected clusters *k* as input. (We will come back to discuss ways to determine this number later.) The *k*-means algorithm is an iterative scheme. The main idea is to calculate the position of each cluster’s center (or *centroid*) from the positions of the points belonging to the cluster, and then to assign points to their nearest centroid. This process is repeated until sufficient convergence is achieved. The basic algorithm can be summarized as follows:

```

choose initial positions for the cluster centroids

repeat:
    for each point:
        calculate its distance from each cluster centroid
        assign the point to the nearest cluster

    recalculate the positions of the cluster centroids

```

The *k*-means algorithm is non-deterministic: a different choice of starting values will potentially result in a different assignment of points to clusters. For this reason, it is customary to run the *k*-means algorithm several times and to compare the results. If you have previous knowledge of likely positions for the cluster centers, you can use it to precondition the algorithm. Otherwise, choose random data points as initial values.

What makes this algorithm efficient is that I can *construct* a centroid position from the positions of the member points: I don’t have to search the existing data points to find out which one would make a good centroid — instead I am free to construct a *new* one. This is usually done by calculating the center-of-mass of the cluster. In two dimensions, we would have:

$$x_c = \frac{1}{n} \sum_i^n x_i$$

$$y_c = \frac{1}{n} \sum_i^n y_i$$

where the sum is over all points in the cluster. (Equivalently in higher dimensions.) I can only do this for vector-like data, however: only such data allows me to form arbitrary “mixtures” in this way.

For strictly categorical data (such as the strings in figure 13.4), the k -means algorithm can not be used (because I cannot “mix” different points to construct a new centroid). Instead, we have to use the k -medoids algorithm. The k -medoids algorithm works in the same way as the k -means algorithm, except that instead of calculating the new centroid, we instead search through all points in the cluster to find that data point, which has the smallest average distance to all other points in the cluster.

The k -means algorithm is surprisingly modest in its resource consumption. On each iteration, the algorithm evaluates the distance function once for each cluster and each point, hence the computational complexity per iteration is $\mathcal{O}(k \cdot n)$ (where k is the number of clusters and n is the number of points in the data set). This is remarkable, because it means that the algorithm is *linear* in the number of points. The number of iterations is usually pretty small: 10...50 iterations are typical. The k -medoids algorithm is more costly, because the search to find the medoid of each cluster is an $\mathcal{O}(n^2)$ process. For very large data sets this may be prohibitive, but you can try to run the k -medoids algorithm on random *samples* of all data points. The results from these runs can then be used as starting points for a run using the full data set.

Despite its cheap-and-cheerful appearance, the k -means algorithm works surprisingly well. It is pretty fast and relatively robust. Convergence is usually quick. Because the algorithm is simple and highly intuitive, it is easy to augment or extend it — for example, to incorporate points with different weights. Or you might want to experiment with different ways to calculate the centroid, possibly using the median position, rather than the mean, and so on.

That being said, the k -means algorithm can fail — annoyingly in situations that exhibit particularly strong clustering! Because of its iterative nature, the algorithm works best in situations involving gradual density changes. If your data sets consists of very dense, widely separated clusters, then the k -means algorithm can get “stuck” if initially two centroids are assigned to the same cluster: moving one centroid to a different cluster would require a very large move, which is not likely to be found by the mostly local steps taken by the k -means algorithm.

Among variants, a particularly important one is *fuzzy clustering*. In fuzzy clustering, we don’t assign each point to a single cluster; instead, for each point and each cluster, we determine the probability that the point belongs to that cluster. Each point therefore acquires a set of k probabilities or weights (one for

each cluster; the probabilities must sum to 1 for each point). We then use these probabilities as weights when calculating the centroid positions. The probabilities also make it possible to declare certain points as “noise” (low probability to belong to *any* cluster) and can help with data sets that contain unclustered “noise” points, or with ambiguous situations such as the one in figure 13.7.

To summarize:

- The k -means algorithms and its variants work best for globular (at least star-convex) clusters. The results will be meaningless for clusters with complicated shapes or for nested clusters (figures 13.6 and 13.3).
- The expected number of clusters is required as an input. If this number is not known, it is necessary to repeat the algorithm with different values and compare the results.
- The algorithm is iterative and non-deterministic. The specific outcome may depend on choice of starting values.
- The k -means algorithm requires vector data; use k -medoids for categorical data.
- The algorithm can be misled if there are clusters of highly different size or different density.
- The k -means algorithm is linear in the number of data points; k -medoids is quadratic in the number of points.

13.3.2 Tree Builders

Another way to find clusters is to successively combine clusters that are “close” to each other into a larger cluster, and to continue doing so until only a single cluster remains. This approach is known as *agglomerative hierarchical clustering*, and leads to a tree-like hierarchy of clusters, with clusters that are close being joined early (near the leafs of the tree) and more distant cluster being joined late (near the root of the tree). (One can also go the opposite direction, continually splitting the set of points into smaller and smaller clusters. When applied to classification problems, this leads to a *decision tree* — see chapter *prediction*.)

The basic algorithm proceeds exactly as outlined above:

1. examine all pairs of clusters
2. combine the two clusters that are closest to each other into a single cluster
3. repeat

What do we mean by the distance between *clusters*? The distance measures that we have defined are only valid between points! To apply them, we need to select (or construct) a single “representative” point from each cluster. Depending on this choice, hierarchical clustering will have lead to different results. The most important alternatives are:

Minimum or Single-Link We define the distance between two clusters as the distance between the two points (one from each cluster) that are *closest* to each other. This choice leads to extended, thinly connected clusters. Because of this, it can handle clusters of complicated shapes, such as those in figure 13.6, but can be sensitive to noise points.

Maximum or Complete-Link The distance between two clusters is defined as the distance between the two points (one from each cluster) that are *farthest away* from each other. With this choice, two clusters are not joined until all points within each cluster are connected to each other, favoring compact, globular clusters.

Average In this case, we form the average over the distances between all pairs of points (one from each cluster). Its characteristic is between the Single- and Complete-Link approaches.

Centroid For each cluster, we calculate the position of a centroid (as in k -means clustering) and define the distance between clusters as the distance between centroids.

Ward's Method Ward's method measures the distance between clusters by the decrease in coherence that occurs when the two clusters are combined: if we combine clusters that are closer together, the resulting cluster should be more coherent than if we combine clusters that are further apart. We can measure coherence as average distance of all points in the cluster from a centroid, or as their average distance from each other. (We'll come back to the concept of cohesion below.)

The result of hierarchical clustering is not actually a set of clusters. Instead, we obtain a tree-like structure, which contains the individual data points at the leaf nodes. This structure can be represented graphically in a *dendrogram* (see figure 13.10). To extract actual clusters from it, we need to walk the tree, evaluate the cluster properties for each subtree, and cut the tree to obtain clusters.

Tree-builders are expensive: we need at least the full distance matrix for all pairs of points (requiring $\mathcal{O}(n^2)$ operations to evaluate). Building the complete tree takes $\mathcal{O}(n)$ iterations (there are n clusters (initially: points) to start with, and at each iteration the number of clusters is reduced by one, because two clusters are combined). For each iteration, we need to search the distance matrix for the closest pair of clusters — naively implemented, this is an $\mathcal{O}(n^2)$ operation (leading to a total complexity of $\mathcal{O}(n^3)$ operations), however, using indexed lookup, this can be reduced to $\mathcal{O}(n^2 \log n)$.

One outstanding feature of hierarchical clustering is that it does not just produce a flat list of clusters, but also shows up their relationships in an explicit way. You need to decide whether this information is relevant for your needs, but keep in mind that the choice of measure for the cluster distance (single- or complete-link, and so on) can have a significant influence on the appearance of the resulting tree structure!

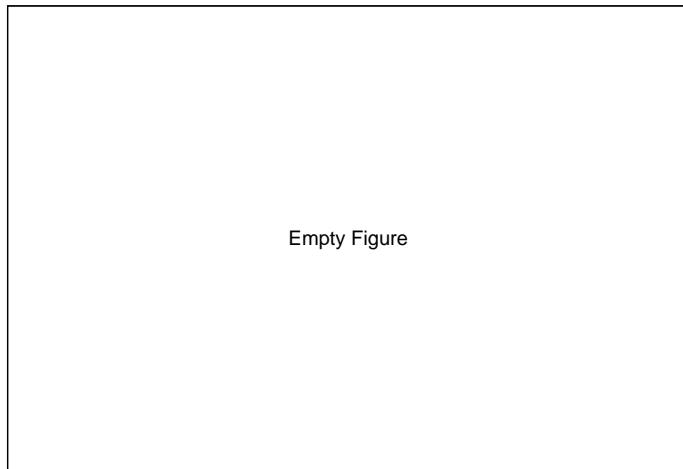


Figure 13.10: TBD

13.3.3 Neighborhood Growers

A third kind of clustering algorithm could be dubbed “neighborhood growers”. They work by connecting points that are “sufficiently close” to each other to form a cluster, and then keep doing so until all points have been classified. This approach makes most direct use of the definition of a cluster as a region of high density, and makes no assumptions about the overall *shape* of the cluster. Therefore, these methods can handle situations involving clusters of complicated shapes (as in figure 13.6), interwoven or even nested clusters (that is: one cluster being entirely surrounded by another one, such as the facial features in the smiley in figure 13.3). In general, neighborhood-based clustering algorithms are more of a special-purpose tool, either for cases that other algorithms don’t handle well (such as the ones just mentioned), or to polish, in a second pass, the features of a cluster found by a general-purpose clustering algorithm such as *k*-means.

The DBSCAN algorithm which we will introduce in this section is one such algorithm and demonstrates some typical concepts. It requires *two* parameters: one is the *minimum density* that we expect to pertain inside of a cluster — points that are less densely packed will not be considered part of any cluster. The other parameter is the *size of the region* over which we expect this density to be maintained: it should be larger than the average distance between neighboring points, but smaller than the entire cluster. The choice of parameters is rather subtle, and clearly requires an appropriate balance.

In a practical implementation, it is easier to work with two slightly different parameters: one is the neighborhood radius r , the other one is the minimum

number of points n that we expect to find within the neighborhood of each point in a cluster. The DBSCAN algorithm distinguishes between three kinds of points: Noise, Edge, or Core points. A noise point is a point which has fewer than n points in its neighborhood of radius r — such a point does not belong to any cluster. A core point of a cluster has more than n neighbors. An edge point, finally, is a point which has fewer neighbors than is required for a core point, but which is itself the neighbor of a core point. The algorithm discards noise points, and concentrates on core points. Whenever it finds a core point, it assigns it a cluster label, then continues to add all its neighbors, and *their* neighbors recursively to the cluster, until all points have been classified.

This description is simple enough, but actually deriving a concrete implementation that is both correct and efficient is less than straightforward. The pseudo-code in the original paper¹ appears needlessly clumsy; on the other hand, I am not convinced that the streamlined version that can be found (for example) on Wikipedia is necessarily correct. Finally, the basic algorithm lends itself to very elegant recursive implementations, but keep in mind that the recursion will not unwind until the current cluster is complete, which means that in the worst case (of a single connected cluster), you will end up putting the entire data set onto the stack!

As pointed out earlier, the main advantage of the DBSCAN algorithm is that it handles clusters of complicated shapes and nested clusters gracefully. On the other hand, it depends sensitively on the appropriate choice of values for its two control parameters, and provides little help in finding them. If a data set contains several clusters with widely varying densities, a single set of parameters may also be insufficient to classify all of the clusters. These problems can be ameliorated by coupling the DBSCAN algorithm with the k -means algorithm: in a first pass, the k -means algorithm is used to identify candidates for clusters. Moreover, statistics on these subsets of points (such as range and density) can be used as input to the DBSCAN algorithm.

The DBSCAN algorithm is dominated by the calculations required to find the neighboring points — basically, for each point in the data set all other points have to be checked, leading to a complexity of $\mathcal{O}(n^2)$. In principle, algorithms and data structures exist to find candidates for neighboring points more efficiently (kd -trees and global grids), but their implementations are subtle and carry their own costs (grids can be very memory-intensive). Coupling the DBSCAN algorithm with a more efficient first-pass algorithm (such as k -means) may therefore be a better strategy.

¹A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96). 1996.

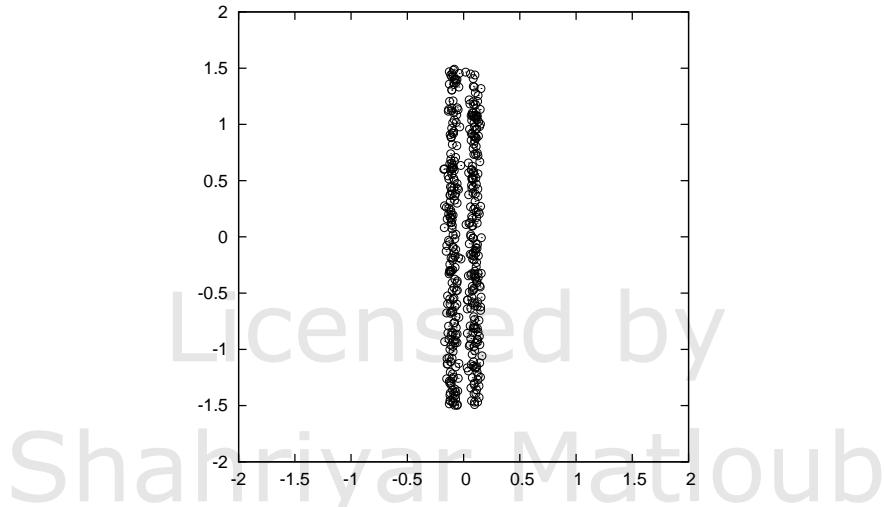


Figure 13.11: TBD

13.4 Pre- and Post-Processing

The core algorithm to group data points into clusters is usually only part (though the most important one) of the whole strategy. Some data sets may require some clean up or normalization to be suitable for clustering: that's the topic of the next section.

Furthermore, we need to inspect the results of every clustering algorithm, to validate and characterize the clusters that have been found. I will introduce some concepts and quantities that are being used to describe clusters and to measure the quality of the clustering.

Finally, several cluster algorithms require certain input parameters (such as the number of clusters to find), and we need to confirm that the values we provided are consistent with the outcome of the clustering process. That will be our last topic in this section.

13.4.1 Scale Normalization

Look at figures 13.11 and 13.12. Wouldn't you agree that the data set in figure 13.12 exhibits two clearly defined, and most of all well-separated clusters, while the one in figure 13.12 does not? Yet, both figures show the *same* data set, just drawn to different scales! In figure 13.11, I have used identical units for both the x- and the y-axis; whereas figure 13.12 has been drawn to maintain a suitable aspect ratio.

This example demonstrates that clustering is not independent from the units in which the data is measured. In fact, in this example, points in two

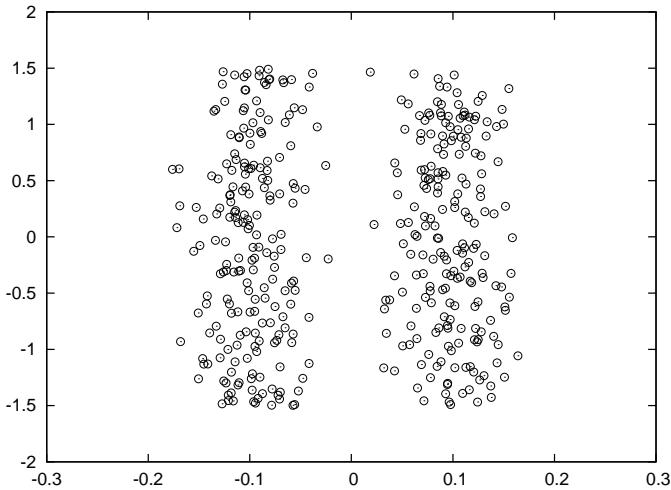


Figure 13.12: TBD

different clusters may be closer to each other than to other points in the same cluster! This is clearly a problem.

If, as in this example, your data spans very different ranges along different dimensions, you need to normalize the data before starting a clustering algorithm. An easy way to achieve this is to divide the data, dimension for dimension, by the range along that dimension. Alternatively, you might want to divide by the standard deviation along that dimension. This process is sometimes called *whitening* or *pre-whitening*, in particular in signal theoretic literature.

You only need to worry about this problem when you are working with vector-like data and using a distance-like metric (such as the Euclidean Distance). It does not affect other distance metrics, such as the correlation coefficient. In fact, there is a special variant of the Euclidean distance, which does the appropriate rescaling per dimension on the fly, the so-called *Mahalanobis Distance*.

Coming soon:
Mahalanobis Distance

13.4.2 Cluster Properties and Evaluation

It is easiest to think about cluster properties in the context of vector-like data and a straightforward clustering algorithm such as k -means. The algorithm already gives us the coordinates of the cluster centroids directly, hence we have the cluster *location*. Two additional quantities are the *mass* of the cluster (that

is, the number of points in the cluster), and its radius. The radius is simply the average deviation of the all points from the cluster center — basically the standard deviation, when using the Euclidean distance:

$$r_c^2 = \sum_{i \text{ in } c} (x_c - x_i)^2 + (y_c - y_i)^2$$

in two dimensions (equivalent in higher dimensions), where x_c and y_c are the coordinates of the center of the cluster, and the sum runs over all points i in the cluster c . Dividing the mass by the radius gives us the *density* of the cluster. (These values can be used to construct input values for the DBSCAN algorithm!)

We can apply the same principles to develop a measure for the overall quality of the clustering. The key concepts are *cohesion* within a cluster and *separation* between clusters. The average distance for all points within one cluster is a measure of the cohesion, the average distance between all points in one cluster from all points in another cluster is a measure of the separation of the two clusters. (If we know the centroids of the clusters, we can use the distance between the centroids as a measure for the separation.) We can go further and form the average (weighted by the cluster mass) of the cohesion for all clusters as a measure for the overall quality.

If a data set can be cleanly grouped into clusters, then we expect the distance between the clusters to be large compared to the radii of the clusters. In other words, we expect the ratio:

$$\frac{\text{separation}}{\text{cohesion}}$$

to be large.

A particular measure based on this concept is the *silhouette coefficient* S . The silhouette coefficient is defined for individual points as follows: Let a_i be the average distance (the cohesion) that point i has from all other points in the cluster it belongs to. Evaluate the average distance that point i has from all points in any cluster it does not belong to; let b_i the smallest such value (that is, b_i is the separation from the “closest” other cluster). Then the silhouette coefficient of point i is defined as:

$$S_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

The numerator is a measure for the “empty space” between clusters (that is: it measures the amount of distance between clusters that is not occupied by the original cluster) and compares it to the greater of the two length scales in the problem (namely the cluster radius and the distance between clusters).

By construction, the silhouette coefficient ranges from -1 to 1 . Negative values indicate that the cluster radius is *greater* than the distance between clusters, so that clusters overlap, and therefore suggest poor clustering. Large values of S suggest good clustering. We can easily extend this definition to silhouette coefficients for entire *clusters*, to develop a measure for the quality of

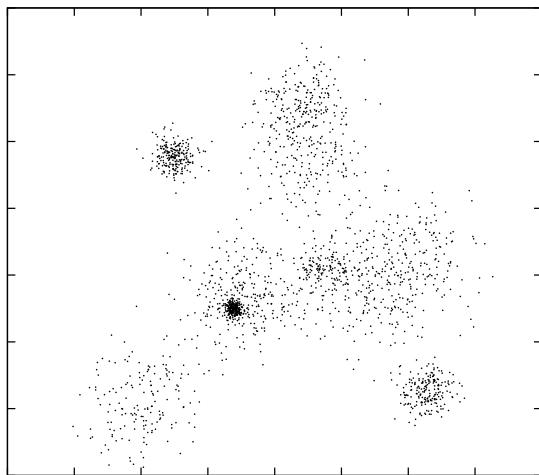


Figure 13.13: TBD

a cluster; and we can define the average over all individual silhouette coefficients as the overall silhouette coefficient for the entire data set, to express the quality of the clustering result.

The overall silhouette coefficient can be useful to determine the number of clusters present in the data set. If we run the k -means algorithm several times for different values of the expected number of clusters and calculate the overall silhouette coefficient every time, it should display a peak near the optimal number of clusters.

Figure 13.14 shows an example. Displayed is the total silhouette coefficient (averaged over all points in the data set) for the two-dimensional data set shown in figure 13.13. This is an interesting data set, because although it does exhibit clear clustering, it is not at all obvious how many *distinct* clusters there really are — any number between six and eight seems plausible. The silhouette coefficient (figure 13.14) confirms this, clearly leaning towards the lower end of this range. (It is interesting to note that the data set was generated, using a random-number generator, to include *ten* distinct clusters, but some of those clusters are overlapping so strongly that it is no longer possible to distinguish them.) This example also serves as a cautionary reminder that it may not always be so easy to determine what actually constitutes a cluster!

Another interesting question concerns the distinction of legitimate clusters from a random (unclustered) background. Of the algorithms that we have seen, only the DBSCAN algorithm explicitly labels some points as background; the k -means and the tree-building algorithm perform what is known as *complete clustering*, by assigning every point to a cluster. We may want to relax this behavior, by trimming those points from each cluster which exceed the average

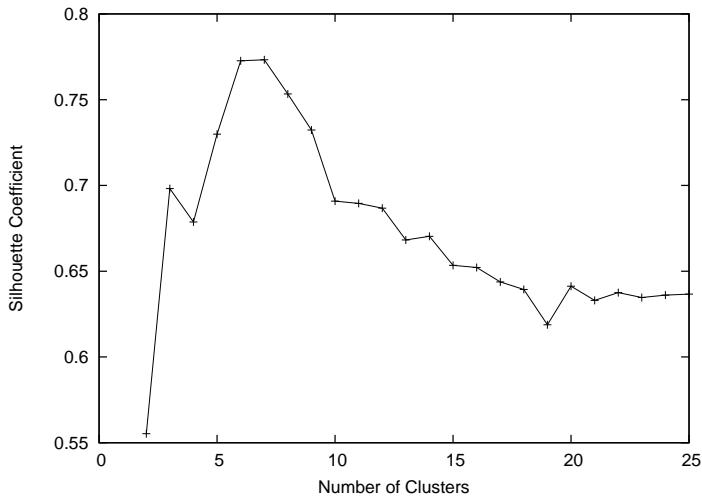


Figure 13.14: TBD

cohesion within the cluster by some amount. This is easiest for fuzzy clustering algorithms, but can be done for other algorithms as well.

13.5 Other Thoughts

Although the three types of clustering algorithms that I introduced in this chapter are probably the most popular and widely used, they certainly don't exhaust the range of possibilities. Here is a brief list of other ideas that can (and have) been used for to develop clustering algorithms.

- We can impose a specific *topology*, such as a *grid* on the data points. Each data point will fall into a single grid cell, and we can use this information to find cells containing unusually many points to guide clustering. Cell-based methods will perform poorly in many dimensions, because most cells will be empty and have few occupied neighbors (the "curse of dimensionality").
- Among grid-based approaches, Kohonen Maps (which we will discuss in chapter 14) have a lot of intuitive appeal.
- To address the challenges posed by high-dimensional feature spaces, some special methods have been suggested. For example, in *subspace clustering*, clustering is performed on only a subset of all available features. These results are then successively extended through inclusion of features that had been ignored in previous iterations.

- Remember Kernel Density Estimates (KDE, chapter 2)? If the dimensionality is not too high, we can generate a KDE for the data set, which is a smooth approximation to the point density and then *maximize the density directly* using standard methods from numerical analysis.
- The QT (“quality threshold”) algorithm is a center-seeking algorithm, which does not require the number of clusters as input — however, we have to fix a maximum *radius* instead. The QT algorithm treats *every* point in the cluster as potential centroid and adds neighboring points (in the order of increasing distance from the centroid) until the maximum radius is exceeded. Once all candidate clusters have been completed in this way, the cluster with the greatest number of points is removed from the data set, and the process starts again with the remaining points.
- There is a well-known correspondence between graphs and distance matrices: Given a set of points, a graph tells us which points are directly connected to each other — but so does a distance matrix! We can exploit this equivalence by treating a distance matrix as the adjacency matrix of a graph. The distance matrix is pruned, by removing connections which are too long, to obtain a sparse graph, which can be interpreted as the backbone of a cluster.
- Finally, *spectral clustering* uses very powerful but abstract methods from Linear Algebra (similar to those used for Principal Component Analysis — see chapter 14) to structure and simplify the distance matrix.

Obviously, much depends on our prior knowledge about the data set: if we expect clusters to be simple and convex, *k*-means suggests itself. On the other hand, if we have a sense for the typical radius of the clusters that we expect to find, then QT clustering might be a natural approach. If we expect clusters of complicated shapes or nested clusters, then an algorithm like DBSCAN will be required. Of course, it might be difficult to develop this kind of intuition, in particular for problems that have significantly more than two or three dimensions!

Besides thinking of different ways to combine points into clusters, we can also think of different ways to define clusters to begin with. All methods we have seen so far relied (directly or indirectly) on the information contained in the distance between any two points. We can extend this concept and start to think of *three point* (or higher) *distance functions*. For example, it is possible to determine the *angle* between any three consecutive points and use this information as measure of the similarity between points. Such an approach might help with situations like the one shown in figure 13.8. Yet another idea would be to measure not the similarity between *points*, but the similarity between a point and a *property of the cluster*. For example, there is a rather straightforward generalization of the *k*-means, where the centroids are no longer pointlike, but are straight lines, representing the “axis” of an elongated cluster. Rather than

measuring the distance for each point from the centroid, this algorithm calculates the distance from this axis to assign points to clusters. This algorithm would be suitable to a situation such as the one in figure 13.7. I don't think any of these ideas that try to generalize beyond pairwise distances have been explored in detail yet.

13.6 A Special Case: Market-Basket Analysis

Which items are frequently bought together? This and similar questions arise in *market-basket analysis* or — more generally — in *association analysis*. Because association analysis is looking for items that occur together, it is in some ways related to clustering. But the specific nature of the problem is sufficiently different to require a separate toolset.

The starting point for association analysis is usually a data set consisting of *transactions*, that is: items that have been bought together (we will often stay with the market-basket metaphor when illustrating a concept). Each transaction corresponds to a single “data point” in regular clustering.

In a transaction, we keep track of all items that have occurred together, but we typically ignore whether any one item was purchased multiple times: all attributes are boolean, indicating only the presence or absence of a certain item. Each item spans a new dimension — if the store sells N different items, then each transaction can have up to N different (boolean) attributes, although each transaction typically contains only a tiny subset of the entire catalogue. (Note that do not necessarily need to know the dimensionality of the catalogue ahead of time: if we don't know it, we can infer an approximation from the number of different items that actually occur in the data set.)

From this description, you can already see how association analysis differs from regular clustering: data points in association analysis are typically very high-dimensional, but also very sparse. Another difference to clustering as we have discussed it so far is that we are not necessarily interested in grouping entire “points” (that is: transactions), but that we would like to identify those dimensions which frequently occur together.

A group of zero or more items occurring together is known as an *item set* (or *itemset*). Each transaction consists of an item set, but every one of its subsets is also an item set. We can construct arbitrary item sets from the overall catalogue. For each such item set, its *support count* is the number of actual transactions that contain the candidate item set as a subset.

Besides simply identifying frequent itemsets, we can also try to derive *association rules*, that is rules of the form: “If items A and B are bought, then item C is also likely to be bought”. Two measures are important when evaluating the strength of an association rule: its *support s* and its *confidence c*. The support of a rule is fraction of transactions in the entire data set, that contain the combined item set (that is: the fraction of transactions that contain all three items A, B, and C). A rule with low support is not very useful, because it is rarely applicable.

Confidence, on the other hand, is a measure for the reliability of an association rule. It is defined as the number of transactions in which the rule is *correct*, divided by the number of transactions in which it is *applicable*. In the example above, it would be the number of times A, B, and C occur together, divided by the number of times A and B occur together.

How do we go about finding frequent item sets (and association rules)? Rather than performing an open-ended search for the “best” association rule, it is customary to set thresholds for the minimum support (such as 10 percent) and confidence (such as 80 percent) required of a rule, and to generate all rules meeting these conditions.

To identify rules, we generate candidate item sets, and then evaluate them against the set of transactions to determine whether they exceed the required thresholds. However, the naive approach, to create and evaluate *all* possible item sets of k elements is infeasible because of the huge number (2^k) of candidate item sets that can be generated — most of which will *not* be frequent! We must find a way to generate candidate item sets more efficiently.

The crucial observation is that *an itemset can only occur frequently if all of its subsets also occur frequently*. This insight is the basis for the so-called *apriori algorithm*, which is the most fundamental algorithm for association analysis.

The apriori algorithm is a two-step algorithm: in the first step, we identify frequent item sets; in the second step, we extract association rules. The first part of the algorithm is the more computationally expensive one. It can be summarized as follows:

Find all 1-item item sets that meet the minimum support threshold.

```
repeat:
    from the current list of k-item item sets, construct (k+1)-item item sets
    eliminate those item sets that do not meet the minimum support threshold
    stop when no (k+1)-item item set meets the minimum support threshold
```

The list of frequent item sets may be all that we require, or we may post-process it to extract explicit association rules. To find association rules, we split each frequent item set into two sets, and evaluate the confidence associated with this pair. From a practical point of view, rules that have a 1-item item set on the “right-hand side” are the easiest to generate and the most important. (In other words, rules of the form “people who bought A and B also bought C”, as opposed to rules of the form “people who bought A and B bought C *and* D”.)

This basic description leaves out many technical details, which are important in actual implementations. For example: how exactly do we create a $(k + 1)$ -item item set, given the list of k -item item sets? For example, We can take every single item that occurs among the k -item item sets and add it, in turn, to every one of the k -item item sets, but this will generate a large number of duplicate item sets, which need to be pruned. Alternatively, we might combine two k -item item sets only if they agree on all but one of their items. Clearly, appropriate data structures are essential to obtain an efficient imple-

mentation. (Similar considerations apply when determining the support count of a candidate item set, and so on.)²

While the apriori algorithm is probably the most popular algorithm for association analysis, very different approaches exist. The *FP-Growth Algorithm*, for example, identifies frequent item sets using something like a string-matching algorithm. Items in transactions are sorted by their support count, and a tree-like data structure is build up, exploiting data sets that agree in the first k items. This tree structure is then searched for frequently occurring item sets.

Association analysis is a relatively complicated problem, involving many technical (as opposed to conceptual) challenges as well. The discussion in this section could only introduce the topic, and attempt to give a flavor for the kinds of approaches that are available. We will see some additional problems of a similar nature in chapter 18.

13.7 A Word of Warning

Clustering can lead you astray, and done carelessly, it can become a huge waste of time. There are at least two reasons for this: although the algorithms are deceptively simple, it can be surprisingly difficult to obtain useful results from them. Many of them depend quite sensitively on several heuristic parameters, and you can spend hours fiddling with the various knobs. Moreover, because the algorithms are simple and the field has a lot of intuitive appeal, it can be a lot of fun to play with implementations and to develop all kinds of modifications and variations.

And that assumes that there actually are any clusters present! (This is the second reason.) In the absence of rigorous, independent results, you will actually spend *more* time on data sets that are totally worthless, perpetually hunting for those clusters that “the stupid algorithm just won’t find”. Perversely, additional domain knowledge does not necessarily make the task any easier: knowing that there should be exactly ten clusters present in figure 13.13 is no help finding the clusters that actually can be identified!

The last question concerns the value that you ultimately derive from clustering (assuming now that at least one of the algorithms has returned something apparently meaningful). It can be difficult to distinguish spurious results from real ones: cluster evaluation methods are not particularly rigorous or unequivocal either (figure 13.14 does not exactly inspire confidence). And that still leaves open the question what you will actually *do* with the results, even assuming that they are significant.

I have found that understanding the actual question that needs to be answered, developing some pertinent hypothesis and models around it, and then verifying them on the data through specific, focussed analysis is usually a far better use of ones time than to go off on a wild-goose clustering search.

²An open-source implementation of the apriori algorithm and many other ones, together with notes on efficient implementation, can be found at: <http://borgelt.net/apriori.html>.

Lastly, I should emphasize that (in keeping with the spirit of this book) the algorithms in this chapter are suitable for moderately sized data sets (a few thousand data points and a dozen dimensions, or so), and for problems that are not particularly pathological. Highly developed algorithms (with names such as CURE or BIRCH) exist for very large or very high-dimensional problems, which usually combine several different cluster finding approaches together with a set of heuristics. You need to evaluate whether such specialized approaches make sense for your situation.

13.8 Toolbox: Pycluster and the C Clustering Library

The C Clustering Library is a mature and relatively efficient clustering library, originally developed to find clusters among gene expressions in microarray experiments. It contains implementations of the k -means and k -medoids algorithms, tree clustering, and even Self-Organized (Kohonen) Maps. It comes with its own GUI frontend, as well as excellent Perl and Python bindings. It is easy to use and very well documented. In the example below, we will use the Python to demonstrate the “center-seeker” algorithms in the library.

Listing 13.1 shows the code used to generate figure 13.14, showing how the silhouette coefficient depends on the number of clusters. Let’s step through it.

We import both the Pycluster library itself as well as the NumPy package. We will use some of the vector manipulation abilities of the latter. The point coordinates are read from the file specified on the command line and then passed to the `kcluster()` function, which performs the actual k -means algorithm. This function takes a number of optional arguments: `nclusters` is the desired number of clusters, and `npass` holds the number of trials that should be performed with *different* starting values. (Remember that k -means clustering is indeterministic with regards to the initial guesses for the positions of the cluster centroids.) The `kcluster()` function will make `npass` different trials and report on the best one.

The function returns three values: the first is an array, which for each point in the original data set holds the index of the cluster it has been assigned to; the second and third provide information about the quality of the clustering (which we ignore in this example). This signature is a reflection of the underlying C API: there, you pass in an array of the same length as the data array and the cluster assignments of each point are communicated via this additional array. This frees the `kcluster()` function from having to do its own resource management, which makes sense in C (and possibly also for very large data sets).

All the information about the result of the clustering are contained in the `clustermap` data structure. The Pycluster library provides several functions to extract this information; here we just demonstrate one: we can pass the `clustermap` to the `clustercentroids()` function to obtain the coordinates of the cluster centroids. (We won’t actually be using these coordinates in the rest of the program.)

You may have noticed that we did not specify the distance function to use in listing 13.1. The C Clustering library does *not* give us the opportunity to use a user-defined distance functions with *k*-means. It includes several standard distance measures (Euclidean, Manhattan, Correlation, and several others), which can be selected through a keyword argument to `kcluster()` (the default is to use the Euclidean distance). Distance calculations can be a rather expensive part of the algorithm, and having them be implemented in C makes the overall program faster. (If we want to define our own distance function, we have to use the `kmedoids()` function, which we will discuss in a moment.)

To evaluate the silhouette coefficient, we need the point-to-point distances and therefore we obtain the distance matrix from the `Pycluster` library. We will also need the number of points in each cluster (the cluster's "mass") later.

Next, we calculate the individual silhouette coefficients for all data points. Remember that the silhouette coefficient involves both the average distance to the all points in the *same* cluster, and the average distance to all points in the *nearest* cluster. Since we don't know ahead of time which one will be the nearest cluster to each point, we simply go ahead and calculate the average distance to *all* clusters. The results are stored in the matrix `sil`.

(In the implementation, we make use of some of the vector manipulation features of NumPy: in the expression `sil[i, :] /= mass`, each entry in row *i* is divided by the corresponding entry in `mass` component-wise. Further down, we make use of "advanced indexing", when looking for the minimum distance between the point *i* and a cluster it does not belong to: in the expression `b = min(sil[i, range(0,c)+range(c+1,n)])`, we construct an indexing vector that includes indices for all clusters, except the one that the point *i* belongs to. See the Workshop in chapter 2 for more details.)

Finally, we form the average over all single-point silhouette coefficients and print the results.

Listing 13.2 demonstrates the `kmedoids()` function, which we have to use if we want to provide our own distance function. As implemented by the `Pycluster` library, the *k*-medoids algorithm does *not* require the data at all — all it needs is the distance matrix!

In the listing, we calculate the distance matrix, using the maximum norm (which is not supplied by `Pycluster`) as distance function. Obviously, we could use any other function here, such as the Levenshtein distance if we wanted to cluster the strings in figure 13.4.

We then call the `kmedoids()` function, which returns a `clustermapper` data structure, similar to the one returned by `kcluster()`. For the `kmedoids()` function, it contains for each data point the index of that data point, which is the centroid of the assigned cluster.

Finally, we calculate the masses of the clusters and print the coordinates of the cluster medoids, together with the coordinates of all points assigned to that cluster.

The C Clustering Library is small and relatively easy to use. You might also want to explore its tree-clustering implementation. It also includes routines for Kohonen maps and Principal Component Analysis — which we will discuss in

chapter 14.

13.9 Further Reading

- *Introduction to Data Mining*. Pang-Ning Tan, Michael Steinbach, Vipin Kumar. Addison-Wesley. 2005.

This is my favourite book on data mining. The presentation is compact and more technical than in most other books on this topic. The section on clustering is particularly strong.

- *Data Clustering: Theory, Algorithms, and Applications*. Guojun Gan, Chaoqun Ma, Jianhong Wu. SIAM. 2007.

This book is a rather recent survey of results from clustering research. The presentation is to be terse, but it provides a good source of concepts and keywords for further investigation.

- *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*. Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96). 1996.

This is the original paper on the DBSCAN algorithm. It can be found on the Web.

- *Algorithms for Clustering Data*. Anil K. Jain and Richard C. Dubes. Prentice Hall. 1988.

An older book on clustering, freely available at http://www.cse.msu.edu/~jain/Clustering_Jain_Dubes.pdf.

- *Metric Spaces: Iteration and Application*. Victor Bryant. Cambridge University Press. 1985.

If you are interested in thinking about distances measures in arbitrary spaces in a more abstract way, then this little (100 page) booklet is a wonderful introduction. It does not require more than some passing familiarity with real analysis, but does a remarkable job demonstrating the power of purely abstract reasoning — both from a conceptual point of view, but also with an eye to real applications.

```

import Pycluster as pc
import numpy as np
import sys

# Read data filename and desired number of clusters from command line
filename, n = sys.argv[1], int( sys.argv[2] )

data = np.loadtxt( filename )

# Perform clustering and find centroids
clustermap, _, _ = pc.kcluster( data, nclusters=n, npass=50 )
centroids, _ = pc.clustercentroids( data, clusterid=clustermap )

# Obtain distance matrix
m = pc.distancematrix( data )

# Find the masses of all clusters
mass = np.zeros( n )
for c in clustermap:
    mass[c] += 1

# Create a matrix for individual silhouette coefficients
sil = np.zeros( n*len(data) )
sil.shape = ( len(data), n )

# Evaluate the distance for all pairs of points
for i in range( 0, len(data) ):
    for j in range( i+1, len(data) ):
        d = m[j][i]

        sil[i, clustermap[j]] += d
        sil[j, clustermap[i]] += d

# Normalize by cluster size (that is: form average over cluster)
for i in range( 0, len(data) ):
    sil[i,:] /= mass

# Evaluate the silhouette coefficient
s = 0
for i in range( 0, len(data) ):
    c = clustermap[i]
    a = sil[i,c]
    b = min( sil[i, range(0,c)+range(c+1,n) ] )
    si = (b-a)/max(b,a) # This is the silhouette coeff of point i
    s += si

# Print overall silhouette coefficient
print n, s/len(data)

```

Example 13.1: TBD

```

import Pycluster as pc
import numpy as np
import sys

# Our own distance function: maximum norm
def dist( a, b ):
    return max( abs( a - b ) )

# Read data filename and desired number of clusters from command line
filename, n = sys.argv[1], int( sys.argv[2] )

data = np.loadtxt( filename )
k = len(data)

# Calculate the distance matrix
m = np.zeros( k*k )
m.shape = ( k, k )

for i in range( 0, k ):
    for j in range( i, k ):
        d = dist( data[i], data[j] )
        m[i][j] = d
        m[j][i] = d

# Perform the actual clustering
clustermapper, _, _ = pc.kmedoids( m, n, npass=20 )

# Find the indices of the points used as medoids, and the cluster masses
medoids = {}
for i in clustermapper:
    medoids[i] = medoids.get(i,0) + 1

# Print points, grouped by cluster
for i in medoids.keys():
    print "Cluster=", i, " Mass=", medoids[i], " Centroid: ", data[i]

    for j in range( 0, len(data) ):
        if clustermapper[j] == i:
            print "\t", data[j]

```

Example 13.2: TBD

Chapter 14

Seeing the Forest for the Trees: Finding Important Attributes

14.1 Introduction

What do you do when you don't know where to start? When you are dealing with a data set that offers no structure that would suggest an angle of attack?

For example, I was once looking through the contracts that a client had with its suppliers for a certain consumable. These contracts were all different in regards to the supplier, the number of units ordered, the duration of the contract and the lead time, the destination location that the items were supposed to be shipped to, the actual shipping date, as well as the actual procurement agent that had authorized the contract — and of course the unit price. What I tried to figure out was which of these quantities had the greatest influence on the unit price.

This kind of problem can be very difficult: there are so many different variables, none of which seems at first glance to be predominant. Furthermore, I have no assurance that the variables are all independent: many of them may be expressing related information. (In the example above, the actual supplier and the shipping destination may be related, because suppliers are chosen to be close to the place where the items are required.)

Because all variables are arising on more or less equal footing, we can't identify a few as the obvious "control" or independent variables and then track the behavior of all the other variables in response to these independent variables. We can try and look at all possible pairings, for example using graphical techniques such as scatter-plot matrices (chapter 5), but that may not really reveal much either — in particular if the number of variables gets truly large. We need some form of computational guidance.

In this chapter, I will introduce a number of different techniques for exactly this purpose. All of them help us select the most important variables or *features* from a multivariate data set in which all variables appear to arise on equal footing. In doing so, we reduce the dimension of the data set from the original number of variables or features to a smaller set, which (hopefully) captures most of the “interesting” behavior of the data. These methods are therefore also known as *feature selection* or *dimensionality reduction* techniques.

A word of warning: the material in this chapter is probably the most advanced and least obvious in the whole book, both conceptually, and also in regards to actual implementations. In particular the following section (on Principal Component Analysis) is very abstract, and may not make much sense if you haven’t had some previous exposure to matrices and linear algebra (including eigen-theory). Other sections are more accessible.

I offer these techniques here nevertheless...because they are of considerable practical importance, but also to give a flavor for the kinds of (more advanced) techniques that are available, and as a possible pointer for further study.

14.2 Principal Component Analysis

Principal Component Analysis (PCA) is the primary tool for dimensionality reduction in multivariate problems. It is a foundational technique that finds applications as part of many other, more advanced procedures.

14.2.1 Introduction

To understand what it can do for us, let’s consider a particularly simple example. Let’s go back to the contract example I gave earlier and now assume that there are only two variables for each contract: its lead time and the number of units to be delivered. What can we say about them? Well, we can draw histograms for each, to understand the distribution of values and to see whether there are “typical” values for either of these quantities. The histograms (in the form of kernel density estimates) are shown in figure 14.1 and don’t reveal anything interesting.

But because there are only two variables in this case, we can also plot one variable against the other in a scatter plot. The resulting graph is shown in figure 14.2 and is very revealing: the lead time of the contract grows with its size. So far, so good.

But we can also look at figure 14.2 in a different way. Recall that the contract data depends on *two* variables (lead time and number of items), so that we would expect the points to fill the two-dimensional space spanned by the two axes (lead time and number of items). But in reality, all the points fall very close to a straight line. A straight line, however, is only one-dimensional, and that means that we only need a *single* variable to describe the position of each point: the distance along the straight line. Put differently, although it appears

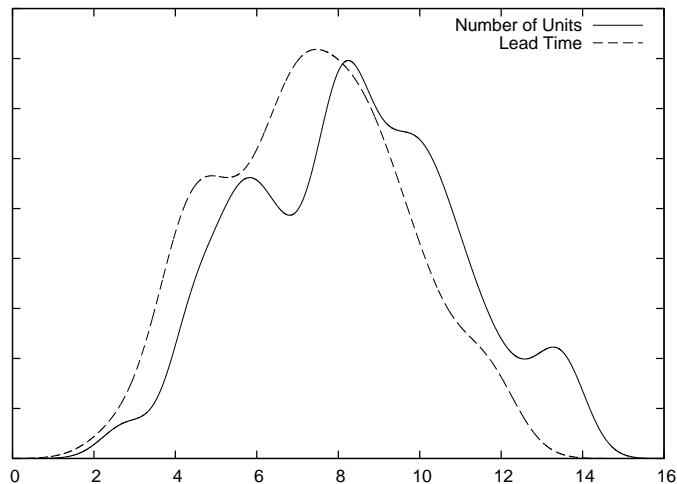


Figure 14.1: TBD

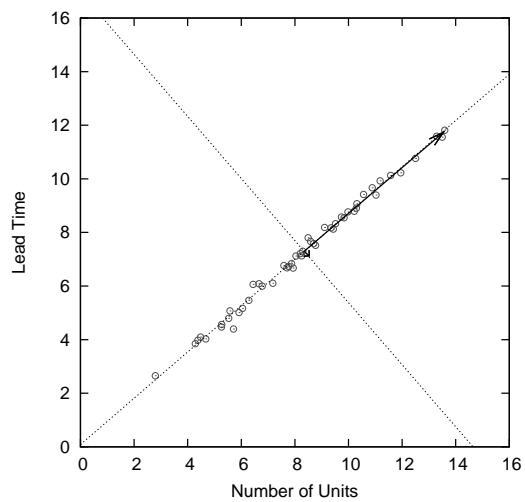


Figure 14.2: TBD

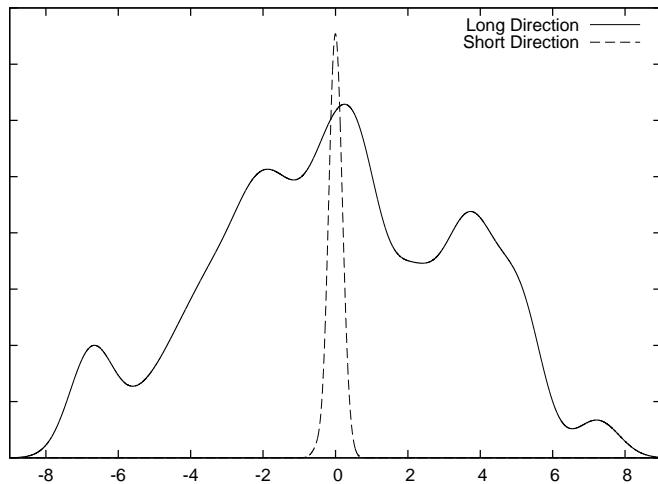


Figure 14.3: TBD - second comp not to scale!

to depend on two variables, the contract data *mostly* depends on a single variable, which lies halfway between the original ones. In that sense, the data is of lower dimensionality than it originally appeared.

Of course, the data still depends on two variables — as it did originally. But most of the *variation* in the data occurs along only one direction. If we were to measure the data only along this direction, we would still capture most of what is “interesting” about the data. In figure 14.3 we see another kernel-density estimate of the same data, but this time not taken along the original variables, but instead showing the distribution of data points along the two “new” directions indicated by the arrows in the scatter plot of figure 14.2. In contrast to the variation occurring along the “long” component, the “short” component is basically irrelevant.

Now, for this simple example, which had only two variables to begin with, it was easy enough to find the lower-dimensional representation, just by looking at it. But that is not going to work once there are significantly more than two variables involved. If there aren’t too many variables, we can generate a scatter-plot matrix (see chapter 5) containing all possible pairs of variables, but even this becomes impractical once there are more than seven or eight variables. Moreover, scatter-plot matrices can never show us more than the combination of any two of the original variables. What if the data in a three-dimensional problem falls onto a straight line that runs along the *space* diagonal of the original three-dimensional data cube? We will not find this by plotting the data against any (two-dimensional!) pair of the original variables.

Luckily, there is a calculational scheme that, given a set of points, will give us the principal directions (basically the arrows in figure 14.2) as combination

of the original variables. That's the topic of the next section.

14.2.2 Theory

We can make progress by using a technique that works for many multi-dimensional problems: if we can summarize the available information regarding the multi-dimensional system in *matrix form*, then we can invoke a large and very powerful body of results from Linear Algebra to transform this matrix into a form that reveals any underlying structure (like the one in figure 14.2).

In the following, I will often appeal to the two-dimensional example from figure 14.2, but the real purpose here is to develop a procedure that will be applicable for any number of dimensions — in particular when the number of dimensions exceeds two or three so that simple visualizations like the ones we discussed so far will no longer work.

To express what we know about the system, we first need to ask ourselves how we can best summarize the way any two variables relate to each other. Looking at figure 14.2, the *correlation coefficient* suggests itself. Remember: the correlation coefficient is a measure for the way the two variables x and y relate to each other and is defined by:

$$\text{corr}(x, y) = \frac{1}{N} \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sigma(x)\sigma(y)}$$

where the sum is over all data points, \bar{x} and \bar{y} are the means of the x_i and the y_i respectively, and $\sigma(x) = \sqrt{\frac{1}{N} \sum_i (x_i - \bar{x})^2}$ is the standard deviation of x (similar for y). The denominator in the expression of the correlation coefficient amounts to a rescaling of the values of both variables to a standard interval — if that is not what we want, we can instead use the *covariance* between the x_i and the y_i :

$$\text{cov}(x, y) = \frac{1}{N} \sum_i (x_i - \bar{x})(y_i - \bar{y})$$

All of these quantities can be defined for *any* two variables (just supply values for x_i and z_i for example), and for a p -dimensional problem, we can find all the $p(p - 1)/2$ different combinations (keep in mind that these coefficients are symmetric: $\text{cov}(x, y) = \text{cov}(y, x)$).

It is now convenient to group the values in a matrix, typically called Σ (not to be confused with the summation sign!), as follows:

$$\Sigma = \begin{pmatrix} \text{cov}(x, x) & \text{cov}(x, y) & \dots \\ \text{cov}(y, x) & \text{cov}(y, y) & \dots \\ \vdots & & \ddots \end{pmatrix}$$

and similarly for the correlation matrix. Because the covariance (or correlation) itself is symmetric under an interchange of its arguments, the matrix Σ itself is also symmetric (meaning that it is the same as its transpose).

We can now invoke an extremely important result from Linear Algebra, known as the *Spectral Decomposition Theorem*, which states the following:

For any real, symmetric $N \times N$ matrix A , there exists an orthogonal matrix U such that

$$B = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_N \end{pmatrix} = U^{-1}AU$$

is a diagonal matrix. (A matrix is *diagonal* if its only non-zero entries are along the main diagonal, from the top left to the bottom right. A matrix is *orthogonal* if its transpose equals its inverse: $U^T = U^{-1}$ or $U^T U = UU^T = I$). Although it may not be obvious, the matrix B contains the same information as A — just packaged differently.

The entries λ_i in the diagonal matrix are called the *eigenvalues* of A and the column vectors of U are the *eigenvectors*. The Spectral Theorem also states that all eigenvectors are mutually orthogonal. Finally, the i -th column vector in U is the eigenvector “associated” with the eigenvalue λ_i : each eigenvalue has an associated eigenvector.

This theorem has a straightforward geometric interpretation. With respect to the original coordinate system (that is, the original variables that the problem was initially expressed in), the matrix is a complicated object, in which the off-diagonal entries are non-zero. However, the eigenvectors span a *new coordinate system*, which is *rotated* with respect to the old one. In this new coordinate system, the matrix takes on a particularly simple, diagonal form, in which all entries that are not on the diagonal vanish. The arrows in figure 14.2 show the directions of the “new” coordinate axes, and the histogram in figure 14.3 measures the distribution of points along these new directions.

Moreover, since the new coordinate axes are found by rotating the original ones, we can express the variables corresponding to these new directions as linear combinations of the original ones. In figure 14.2 for instance, to make a step in the “new” direction (along the diagonal), you take a step along the (old) x -axis, followed by a step along the (old) y -axis. We can therefore express the new direction (call it \hat{x}) in terms of the old ones: $\hat{x} = (x + y)/\sqrt{2}$ (the factor $\sqrt{2}$ is just a normalization factor).

The Spectral Decomposition Theorem applies to *any* symmetric matrix, but the interpretation of the results (what do the eigenvalues and eigenvectors mean?) depends of course on the specific application. In our case, where we diagonalize a covariance or correlation matrix, it can be shown that the eigenvectors point in *the directions of greatest variance*, and that the corresponding eigenvalues are a measure of the amount of variance associated with this direction.

Look again at figure 14.2: the long arrow is aligned with the direction of greatest variance. By construction, the short arrow is perpendicular to the long one, and points into the direction of the second-greatest variance. For this two-dimensional data set, these are the only possible directions, but for a higher-dimensional data set, the sequence would go on like this: each new eigenvector being perpendicular to all other ones, and arranged in the order of decreasing variance.

We can apply the same kind of reasoning to any problem, in any number of dimensions. The process is exactly the same: find the covariance or correlation matrix (if the original problem has p dimensions, this will be a $p \times p$ matrix) and diagonalize it. The eigenvectors will give us the directions of greatest variance, and the associated eigenvalues will measure the amount of variance along the corresponding direction. We will have more to say on how to interpret the results of a Principal Component Analysis in a moment, but first we need to understand how to compute eigenvalues and eigenvectors.

14.2.3 Computation

The theory just described would be of only limited interest if there weren't practical algorithms to calculate both eigenvalues and eigenvectors. These calculations are always numerical — there are algebraic ways to determine both eigenvectors and eigenvalues that you may have encountered in school, but they are impractical for matrices larger than 2×2 and infeasible for matrices larger than about 4×4 .

However, there are several elegant *numerical* algorithms to invert and diagonalize matrices, and they tend to form the foundational part of any numerical library. On the other hand, they are not trivial understand, and developing high-quality implementations (that avoid things like round-off error) is a specialized skill. There are no good reasons to write your own, you should always use an established library. (Every numerical library or package will include the required functionality.)

Matrix operations are relatively expensive, and run-time performance can be a serious concern for large matrices. Matrix operations tend to be of $\mathcal{O}(N^3)$ complexity, meaning that doubling the size of the matrix will increase the time to perform an operation by a factor of $2^3 = 8$. In other words, doubling the problem size will result in an almost *ten-fold* increase in runtime! This is a non-issue for small matrices (up to 100×100 or so), but you will hit a brick wall at a certain size (somewhere between 5000×5000 and 50000×50000). Such large matrices can occur in practice, but usually in application domains that are different from this chapter's topic. For even larger matrices there are alternative algorithms, which however only calculate the most important of the eigenvalues and eigenvectors.

I will not go into details about different algorithms, but I want to mention one of them explicitly, because it is of particular importance in this context. If you read up on Principal Component Analysis (PCA), you will encounter the term *Singular Value Decomposition* or SVD; in fact, many books treat PCA and SVD as equivalent expressions for the same thing. That is not correct, they are really quite different. PCA is the application of spectral methods to covariance or correlation matrices — it is a conceptual technique, not an algorithm. By contrast, the SVD is a specific algorithm, that can be applied to many different problems, one of them being the PCA.

The reason that the SVD features so prominently in discussions of the PCA is that it combines two required steps in one. In our discussion of the PCA, we have so far assumed that you first calculate the covariance or correlation matrix explicitly from the set of data points, and then diagonalize it. The SVD performs these two steps in one fell swoop: you pass the set of data points directly to the SVD and it calculates the eigenvalues and eigenvectors of the correlation matrix directly from the data points.

The SVD is a very interesting and versatile algorithm, which is unfortunately rarely included in introductory classes on Linear Algebra.

14.2.4 Interpretation

What does Principal Component Analysis really tell us? To recap: we start out with a multivariate data set: each data point is a collection of different variables — it is quite literally a *point* in a multi-dimensional coordinate system, where the coordinate axes directly correspond to quantities from the problem domain (such as the lead time or the size per order).

We also suspect that not all of these variables are really needed: some of them may be redundant (expressing more or less the same thing), and others may be irrelevant (carrying little information). Principal Component Analysis is a way to identify those variables (or: combinations of variables) that are redundant and can therefore be neglected, resulting in a simpler problem. (That's the whole point of dimensionality reduction approaches — if we don't expect to eliminate any of the original variables, there is no point in trying such approaches!)

An indicator that some variables may be redundant (that is: express the "same thing") is that they are correlated. That's pretty much the definition of correlation: knowing that if we change one variable, there will be a corresponding change in the other. Principal Component Analysis uses the information contained in the mutual correlations between variables to identify those that are redundant. This has an important consequence: If all of the original variables are uncorrelated, PCA will be of no help!

Given the correlations among the original variables, Principal Component Analysis finds an alternate coordinate system to describe the points that make up the data set. The direction of the axes in this new coordinate system are given by the eigenvectors of the correlation matrix. They provide the new "variables" in which we express the points of the data set. In general, they will not correspond neatly to individual quantities from the problem domain any more; instead they will be given as a *combination* of the original variables.

However, these new variables are uncorrelated with each other! (By construction, their correlation matrix is diagonal, meaning that the correlation between any two different variables is zero.) These new variables are the Principal (uncorrelated) Components, into which the data set can be split.

We may now ask about the distribution of the data along each of these new directions: how wide do the data points spread? If they spread widely, then this direction is "interesting", since it contains a lot of information. On the other hand, if the data falls only into a narrow region along some direction, then the corresponding variable is not important for this data set. (Look again at figure 14.2 and 14.3 — the direction of the "short" arrow simply is not interesting.)

The last bit of the puzzle is that we already know the width of the distributions along all of the new axis! Why is that? Because the usual measure for the width of a distribution is the standard deviation σ , or its square, the variance: $\sigma^2 = \sum_i (x_i - \hat{x})^2$. But these are exactly the diagonal entries of the covariance matrix Σ ! Once we diagonalize Σ , the entries along the diagonal (the "eigenvalues") are the variances along the "new" directions. The width (in terms

of the standard deviation) of the distribution of points along any of the new directions is therefore given simply by the square root of the eigenvalues of Σ .

Now we have everything we need, we just need to put it all together properly. The new directions (the eigenvectors) provide a new set of variables for the data set. These new variables are uncorrelated with each other. To each new variable (each eigenvector) there is a corresponding eigenvalue, which gives the width of the distribution for that variable. Finally, we are interested only in those variables that exhibit an appreciable spread — variables that vary only over a small range do not contain much information. The information about the spread of each of the new variables is contained in the eigenvalues. We therefore inspect the list of eigenvalues: if any of the original variables indeed were redundant, then some of the eigenvalues will be significantly smaller than the rest. We retain only those of the new variables corresponding to the large eigenvalues, and neglect the rest. The result is a data set which depends on *fewer* variables than the original data set, which are also *uncorrelated*, but which in general will not correspond neatly to quantities from the problem domain.

14.2.5 Practical Points

As you can see, Principal Component Analysis is a pretty involved technique — although with the appropriate tools it becomes almost ridiculously easy to perform (see the Workshop in this chapter). But convenient implementations don't make the conceptual difficulties go away, or ensure that the method is applied appropriately.

First, I'd like to emphasize that the mathematical operations underlying Principal Component Analysis (namely the diagonalization of a matrix) are very general: they are a set of formal transformations, which apply to *any* symmetric matrix (and transformations of this nature are used for all kinds of purposes in literally all fields of science and engineering).

In particular, there is nothing specific to data analysis about them. It does not involve any of the concepts that we usually deal with in statistics or analysis: there is no mention of populations, samples, distributions, or models. Instead, Principal Component Analysis is a set of formal transformations, applied to the covariance matrix of a data set. As such, it can be either *exploratory* or *preparatory*.

As an exploratory technique, we may inspect its results (the eigenvalues and eigenvectors) for anything that helps us develop an understanding of the data set. For example, we may look at the contributions to the first few Principal Components, to see whether we can find an intuitive interpretation of them (we will see an example in the Workshop section). Biplots (see below) are a graphical technique that can be useful in this context.

But we should keep in mind that this kind of investigation is exploratory in nature: there is no guarantee that the results of a Principal Component Analysis will turn up anything useful. (In particular we should not expect the Principal Components to have an intuitive interpretation in general.)

On the other hand, Principal Component Analysis may be used as a preparatory technique. Keep in mind that (by construction) the Principal Components are uncorrelated. Therefore, we can transform any multivariate data set into an equivalent form, in which all variables are mutually independent, before performing any subsequent analysis. Identifying a subset of Principal Components which captures most of the variability in the data set for the purpose of reducing the dimensionality of the problem (as we discussed in this chapter) is another preparatory use of Principal Component Analysis.

As a preparatory technique, Principal Component Analysis is always applicable, but may not always be useful. For instance, if the original variables are already uncorrelated, then the PCA cannot do anything for us. Similarly, if none of the eigenvalues are significantly smaller (so that their corresponding Principal Components can be dropped), we again do not gain anything from the PCA.

Finally, let me re-iterate that PCA is just a mathematical transformation, which can be applied to any symmetric matrix. This means that its results are not uniquely determined by the data set, but instead are sensitive to the way the inputs are prepared. In particular, the results of a PCA depend on the actual *numerical values* of the data points, and therefore on the *units* in which the measurements have been recorded. If the numerical values for one of the original variables are consistently larger than the values of the other variables, the former one will unduly dominate the spectrum of eigenvalues. (We will see an example of this problem in the Workshop.) To avoid this kind of problem, all variables should be of comparable scale — a systematic way to achieve this is to work with the correlation matrix (in which all entries are normalized by their auto-correlation) instead of the covariance matrix.

14.2.6 Biplots

Biplots are an interesting way to visualize the results of a Principal Component Analysis.

In a biplot we plot the data points in a coordinate system spanned by the first two Principal Components (that is: those two of the “new” variables corresponding to the largest eigenvalues). In addition, we also plot a representation of the *original* variables, but projected into the space of the new variables. The data points are represented by symbols, whereas the directions of the original variables are represented through arrows. (See figure 14.5 in the Workshop section.)

In a biplot we can immediately see the distribution of points when represented through the new variables (and look for clusters, outliers, or other interesting features). In addition, we can also see how the original variables relate to the first two principal components and to each other: if any of the original variables are approximately aligned with the horizontal (or vertical) axis, then they are approximately aligned with the first (or second) principal component (because in a biplot, the horizontal and vertical axes coincide with

the first and second principal component). We can thus see which of the original variables contribute strongly to the Principal Components, which might help us develop an intuitive interpretation for the Principal Components. Furthermore, any of the original variables that are roughly redundant will show up as more or less parallel to each other in a biplot — which again can help us identify such combinations of variables in the original problem.

Biplots may or may not be helpful. There is a whole complicated set of techniques to interpret biplots and to read off various quantities from them, but they are rarely used and I have not found them helpful. If I do a Principal Component Analysis, I will routinely also draw a biplot — it tells me something worthwhile, that's great; but if not, then I am not going to spend much time on it.

14.3 Visual Techniques

Principal Component Analysis is a rigorous prescription, and what is sometimes known as a “data centric” technique: it transforms the original data in a precisely prescribed way, without ambiguity and without making further assumptions. The results are an expression of properties of the data set. It is up to us to interpret them, but the results are true, independent of whether we find them useful or not.

By contrast, the methods I am going to describe in this section, are convenience methods, which attempt to make multi-dimensional data sets more “palatable” for human consumption. They do not calculate any rigorous properties that are inherent in the data set; instead, they try to transform the data in such a way that it can be plotted, while at the same time trying to be as “faithful” to the data as possible.

I'll not discuss any of these methods in depth, since personally, I do not find them worth the effort: on the one hand, they are (merely) exploratory in nature, on the other hand, they require rather heavy numerical computations and some non-trivial theory. Their primary results are projections (that is: graphs) of data sets, which can be difficult to interpret if the number of data points or their dimensionality becomes large — which is exactly when I expect a computationally intensive method to be helpful! Nevertheless, there are situations where you might find these methods useful, and they provide some interesting concepts for how to *think* about data. This last reason is the most important to me, which is why this section tries to emphasize concepts, while skipping most of the technical details.

14.3.1 Multidimensional Scaling

If we have a set of data points (that is: the *coordinates* of each data point), we can easily find the distance between any pair of points (see also chapter 13 for a discussion of distance measures). Multidimensional scaling (MDS) attempts

to answer the opposite question: given a distance matrix, can we recover the explicit coordinates of the points?

This question has a certain intellectual appeal in its own right, but is of course relevant in situations where our information about a certain system is limited to the differences between data points. For example, in usability studies or surveys, we may ask the respondents to list which of a set of cars (or whiskeys, or pop singers) they find the most or the least alike; and in fact the whole method was first developed for use in psychological studies. The question is: given such a matrix of relative preferences or distances, can we come up with a set of absolute positions for each entry?

First, we must choose the desired number of dimensions of our points. The dimension $d = 2$ is used often, so that the results can be plotted easily, but other values for d are also possible.

If the distance measure is Euclidean, that is if the distance between two points is given by:

$$d(x, y) = \sqrt{\sum_i^d (x_i - y_i)^2}$$

where the sum is running over all dimensions, then it turns out that we can invert this relationship explicitly and find expressions for the coordinates in terms of the distances. (The only additional assumption we need to make is that the center of mass of all data points lies at the origin, but this amounts to no more than an arbitrary translation of all points.) This is known as *classical* or *metric scaling*.

The situation is more complicated if we cannot assume that the distance measure is Euclidean. Now we can no longer invert the relationship exactly, and must resort to iterative approximation schemes. Although the resulting coordinates may not replicate the original distances exactly, they are required to obey their rank order: if the original distances between any three points obeyed the relationship $d_{12} < d_{13}$, then the calculated coordinates of the three points must honor this relationship as well. For this reason, this method is known as *ordinal scaling*.

The basic algorithm makes a guess for the coordinates and calculates a distance matrix based on them. The coordinates are now changed iteratively, to minimize the discrepancy (known as the “stress”) between the new distance matrix and the original one.

In either case, we end up with a set of coordinates in the desired number of dimensions (usually two), which we can use to plot the data points in a form of scatter plot. We can then inspect this plot for clusters, outliers, or other features.

14.3.2 Network Graphs

In passing I’d like to mention *force-based algorithms* for drawing network graphs, because they are similar in spirit to multidimensional scaling.

Imagine we have a network, consisting of nodes, some of which are connected by vertices (or edges), and we would like to find a way to plot this network in a way that is “attractive” or “pleasing”. One way to do this is to treat the edges as springs, in such a way that each spring has a preferred extension and exerts an opposing force — in the direction of the spring — if compressed or extended beyond its preferred length. We can now try to find a configuration (that is, a set of coordinates for all nodes) that will minimize the overall tension of the springs.

There are basically two ways we can go about this: we can write down the total energy due to the distorted springs and minimize it with respect to the node coordinates using a numerical minimization algorithm. Alternatively, we can “simulate” the system, by initializing all nodes with random coordinates, and then iteratively moving each node in response to the spring forces acting on it. For smaller networks, we can update all nodes at the same time; for very large networks we may randomly chose a single node at each iteration step for update and continue until the configuration no longer changes.

It is easy to see how this basic algorithm can be extended to include richer situation — for instance, edges carrying different weights.

Note that this algorithm does not make guarantees regarding the distances that are maintained between the nodes in the final configuration. It is purely a visualization technique.

14.3.3 Grand Tours and Projection Pursuits

In a scatter-plot matrix, we show the projections of a data set on the planes spanned by all possible pairings of the original variables. Using Principal Component Analysis, we show the projection of the same data set on the planes spanned by the first few principal component eigenvectors. In either case, however, we select only a very limited number of all the possible two-dimensional planes that the data can be projected onto.

Grand Tours and *Projection Pursuits* are attempts to enhance our understanding of a data set by presenting many closely related projections in form of an animated “movie”. The concept is straightforward: we begin with some projection, and then continuously move the view point around the data set. (If the data set was three-dimensional, you can imagine the view point moving on a sphere enclosing it.)

In Grand Tours, the viewpoint is allowed to perform essentially a random walk around the data set. In a Projection Pursuit, the viewpoint is moved so that it will improve the value of an “index”, which measures how “interesting” a specific projection will appear. Most indices currently suggested measure properties such as deviation from Gaussian behavior. At each step of a Pursuit, the program evaluates several possible projections and then selects the one that most improves the chosen index. Eventually a Pursuit will reach a local maximum for the index, at which point it needs to be restarted from a different starting point.

Obviously, Tours and Pursuits require specialized tools that are able to perform the required projections, and do so in real-time. You might want to check out the open source tool GGobi if you want to experiment with this kind of approach.

Although I find the approach compelling, I have not found Tours very useful in practice. It can be very confusing to watch a movie of essentially random patterns, and very frustrating to interact with projections when attempting to explore the neighborhood of an interesting view point.

14.4 Kohonen Maps

Self-organizing maps (SOM), often called “Kohonen Maps” after their inventor, are different from the techniques we discussed so far. In principal component analysis and multi-dimensional scaling, we attempted to find a new, more favorable, arrangement of points by moving them about in a continuous fashion. When constructing a Kohonen map, by contrast, we map the original data points to cells in a *lattice*. The presence of a lattice forces a fixed topology on the system; in particular, each point in a lattice has a fixed set of neighbors. (This property is typically and confusingly called “ordering” in most literature on Kohonen maps.)

The basic process to construct a Kohonen map works as follows: We start with a set of k data points in p dimensions, so that each data point consists of a tuple of p numeric values. (I am intentionally avoiding the word “vector” here, because there is no requirement that the data points must fulfill the “mixable” property characteristic of vectors — see chapters 1 and 13.)

Next we prepare a lattice. For simplicity, we consider a two-dimensional, square lattice, consisting of $n \times m$ cells. Each cell contains a p dimensional tuple, similar to a data point, called the “reference tuple”. We initialize this tuple with random values. In other words, our lattice consists of a collection of random data points, but arranged on a regular grid.

Now we perform the following iteration: for each data point, we find that cell in the lattice that has the smallest distance between its contained p -tuple and the data point, and we assign the data point to this cell. Note that multiple data points can be assigned to the same cell if necessary.

Once all the data points have been assigned to cells in the lattice, we update the p -tuples of all cells, based on the values of the data points assigned to each cell *and to those assigned to cells in the neighborhood*.

When all lattice points have been updated, we restart the iteration and begin assigning data points to cells again (after erasing the previous assignments). We stop the iteration if the assignments no longer change or the difference between the original cell values and their updates are sufficiently small.

This is the basic algorithm for the construction of a Kohonen map. It has certain similarities with the k -means algorithm that we discussed in chapter 13: both are iterative procedures, in which data points are assigned to cells or

clusters, and the cell or cluster is updated based on the points assigned to it. However, two things are specific to Kohonen maps:

- Each data point is mapped to a cell in the lattice, and this implies that each data point is being put into a specific neighborhood of other data points (which have been mapped to neighboring cells).
- Because the updating step for each cell does not only rely on the current cell, but also on the neighboring cells, the resulting map will show a “smooth” change of values: changes are averaged or “smeared out” over all cells in the neighborhood. Viewed the other way around, this implies that points that are somehow similar to each other will map to lattice cells that are in close proximity to each other.

While the basic algorithm seems rather simple, we need to decide on a number of technical details if we want to develop a concrete implementation. Most importantly, we still need to give a specific prescription how the reference tuples will be updated by the data points assigned to the current cell and its neighborhood.

In principle, it would be possible to recalculate the values for the reference tuple from scratch every time, by forming a component-wise average of all data points assigned to the cell. In practice, this may lead to instability in iteration, and therefore it is usually recommended to perform an incremental update of the reference value instead, based on the difference between the current value of the reference tuple and the assigned data points. If $y_i(t)$ is the value of the reference tuple at position i and at iteration t , we can write its value at the next iteration step $t + 1$ as:

$$y_i(t+1) = y_i(t) + \sum_k h(i, j; t) [x_k(j; t) - y_i(t)]$$

where $x_k(j; t)$ is the data point k , which has been assigned to lattice point j at iteration step t , and the sum runs over all data points. The weight function $h(i, j; t)$ is now chosen to be a decreasing function of the distance between the lattice cells i and j , and is also made to shrink in value as the iteration progresses. A typical choice is a Gaussian:

$$h(i, j; t) = \alpha(t) \exp\left(-\frac{d_{ij}}{2\sigma(t)}\right)^2$$

where d_{ij} is the Euclidean distance between lattice points i and j , and $\alpha(t)$ and $\sigma(t)$ are decreasing functions of t . Other choices (besides the Gaussian) are also possible — for instance, we may choose a step function to delimit the effective neighborhood.

Even with these definitions, we still need to decide on further details:

- What is the topology of the lattice? Square lattices (like quad ruled paper) are convenient, but very strongly single out two specific directions.

Hexagonal lattices (like a honeycomb) are more isotropic. We also need to fix the boundary conditions: do cells at the edge of the lattice have fewer neighbors than cells in the middle of the lattice, or do we wrap the lattice around and connect the opposite edges to form periodic boundary conditions?

- What is the size of the lattice? Obviously, the number of cells in the lattice should be smaller than the number of data points (otherwise, we end up with unoccupied cells). But how much smaller? Is there a preferred ratio between data points and lattice cells? Also: should the overall lattice be square ($n \times n$) or rectangular ($n \times m$)? In principle, we can even consider lattices of different shape — triangular, for example, or circular. However, if we choose a lattice of higher symmetry (square or circular), the *orientation* of the final result within the lattice is not fixed; for this reason, it is advised that the lattice should always be oblongated (that is, rectangular, rather than square).
- We need to choose a distance or similarity measure to measure the distance between data points and reference tuples.
- We still need to fix the numerical range of $\alpha(t)$ and $\sigma(t)$ and define their behavior with t .

In addition, there are lots of opportunities for low-level tuning, in particular in regards to performance and convergence. For example, we may find it beneficial to initialize the lattice points with values other than random numbers.

Finally, we may ask what we can actually do with the resulting lattice of converged reference tuples? Here are some ideas:

- We can use the lattice to form a smooth “heatmap” style visualization of the original data set. Because cells in the lattice are closely packed, a Kohonen map interpolates smoothly between different points. This is in contrast to the result from either PCA or MDS, which only yield individual, scattered points.
- One problem when plotting a Kohonen map is to decide which feature to show. If the original data set was p dimensional, you may have to plot p different graphs to see the distribution of all features.
- The situation is more favorable if one of the features of interest is categorical with few only a few values. In this case, you can plot the labels on the graph and study their relationships (which labels are close to each other, and so on). In this case, it is also possible to use a “trained” Kohonen map to classify new data points or data points with missing data.
- In particular if the number of cells in the lattice was chosen much smaller than the number of original data points, you can try mapping the reference tuples *back* into the original data space, using them as *prototypes* for clustering purposes.

Kohonen maps are an interesting technique. On the one hand, the whole concept is very ad-hoc and heuristic, with little rigorous theory, and therefore with little guidance on the choice of specific details. On the other hand, their hands-on, intuitive quality can make them a versatile tool in situations calling for experimentation and exploration.

14.5 Perspective

Coming soon

14.6 Toolbox: PCA with R

Principal Component Analysis is a complicated technique, and therefore it makes sense to use specialized tools that hide most of the complexity. Here we are going to use R, which is the best-known open-source package for statistical calculations. (We already covered some of the basics of R in the Workshop section of chapter 10; in this chapter I want to demonstrate the kind of advanced functionality that is built into R.)

We are going to study a non-trivial example: for a collection of almost 5000 wines, almost a dozen physico-chemical quantities were measured, as well as the results of a subjective “quality” or taste test.¹ The properties are:

- 1 - fixed acidity
- 2 - volatile acidity
- 3 - citric acid
- 4 - residual sugar
- 5 - chlorides
- 6 - free sulfur dioxide
- 7 - total sulfur dioxide
- 8 - density
- 9 - pH
- 10 - sulphates
- 11 - alcohol
- 12 - quality (score between 0 and 10)

That's a complicated data set, and having to handle eleven input variables is not comfortable. Can we find a way to make sense of them, and possibly even find out which are most important in determining the overall quality of the wine?

This is a problem made for an application of the PCA. And as we will see, R makes it really easy for us.

¹This example is taken from the “Wine Quality” data set, available at the UCI Machine Learning repository at <http://www.ics.uci.edu/~mlearn/MLRepository.html>.

For this example, I'll take you on a slightly roundabout route. Be prepared that our initial attempt will lead to an incorrect conclusion! I am including this detour here for a number of reasons:

- I want to remind you that real data analysis, with real and interesting data sets, usually does not progress linearly.
- Instead, it is very important that, as we work with a data set, we constantly keep checking and questioning our results as we go along: Do they make sense? Might we be missing something?
- I also want to demonstrate how R's interactive programming model facilitates the required exploratory work style: try something and look at the results; if they look wrong, go back and make sure you are on the right track, and so on.

Although it can be scripted for batch operations, R is primarily intended for interactive use, and that is what we will be doing. We first load the data set into a heterogeneous “data frame”, and then invoke the desired functions on it. Functions in turn may return data structures themselves that can be used as input to other functions, that can be printed in a human readable format to the screen, or that can be plotted.

R includes many statistical functions as built-in functions. In our specific case, we can perform an entire Principal Component Analysis in a single command:

```
wine <- read.csv( "winequality-white.csv", sep=';', header=TRUE )
pc <- prcomp( wine )
plot( pc )
```

This snippet of code reads the data from file and assigns the resulting data frame to the variable `wine`. The `prcomp()` function performs the actual Principal Component Analysis, returning a data structure containing the results, which we assign to the variable `pc`. We can now examine this returned data structure in various ways.

R makes heavy use of function overloading — a function such as `plot()` will accept different forms of input and try to find the most useful action to perform given the input. For the data structure returned by `prcomp()`, it constructs a so-called *scree plot* (figure 14.4), showing the magnitudes of the variances for the various principal components, from the greatest to the smallest.

We see that the first eigenvalue entirely dominates the whole spectrum, suggesting that the corresponding new variable is all that matters (which of course would be great). To understand in more detail what is going on, we look at the corresponding eigenvector. The `print()` function is another overloaded function, which for this particular data structure prints out the eigenvalues and eigenvectors.



Figure 14.4: TBD

```

print( pc )

(some output omitted...)

          PC1           PC2           PC3
fixed.acidity -1.544402e-03 -9.163498e-03 -1.290026e-02
volatile.acidity -1.690037e-04 -1.545470e-03 -9.288874e-04
citric.acid    -3.386506e-04  1.403069e-04 -1.258444e-03
residual.sugar -4.732753e-02  1.494318e-02 -9.951917e-01
chlorides       -9.757405e-05 -7.182998e-05 -7.849881e-05
free.sulfur.dioxide -2.618770e-01  9.646854e-01  2.639318e-02
total.sulfur.dioxide -9.638576e-01 -2.627369e-01  4.278881e-02
density         -3.596983e-05 -1.836319e-05 -4.468979e-04
pH              -3.384655e-06 -4.169856e-05  7.017342e-03
sulphates        -3.409028e-04 -3.611112e-04  2.142053e-03
alcohol          1.250375e-02  6.455196e-03  8.272268e-02

(some output omitted...)

```

That is now disturbing: if you look closely you will see that both the first and the second eigenvector are dominated by the sulfur dioxide concentration — and by a wide margin at that! That does not seem right. I don't understand much about wine, but I would not think that the sulfur dioxide content is all that matters in the end.

Maybe we went ahead a little too fast. What do we actually know about the

data in the data set? Right — absolutely nothing! Time to find out. One quick way to do so is to use the `summary()` function on the *original* data:

```
summary(wine)
fixed.acidity  volatile.acidity  citric.acid  residual.sugar
Min. : 3.800  Min. :0.0800  Min. :0.0000  Min. : 0.600
1st Qu.: 6.300 1st Qu.:0.2100 1st Qu.:0.2700 1st Qu.: 1.700
Median : 6.800 Median :0.2600 Median :0.3200 Median : 5.200
Mean   : 6.855 Mean  :0.2782 Mean  :0.3342 Mean  : 6.391
3rd Qu.: 7.300 3rd Qu.:0.3200 3rd Qu.:0.3900 3rd Qu.: 9.900
Max.   :14.200 Max.  :1.1000 Max.  :1.6600 Max.  :65.800
chlorides    free.sulfur.dioxide total.sulfur.dioxide density
Min. :0.00900  Min. : 2.00      Min. : 9.0      Min. :0.9871
1st Qu.:0.03600 1st Qu.:23.00     1st Qu.:108.0    1st Qu.:0.9917
Median :0.04300 Median :34.00     Median :134.0    Median :0.9937
Mean   :0.04577 Mean  :35.31     Mean  :138.4    Mean  :0.9940
3rd Qu.:0.05000 3rd Qu.:46.00     3rd Qu.:167.0    3rd Qu.:0.9961
Max.   :0.34600 Max.  :289.00    Max.  :440.0    Max.  :1.0390
pH      sulphates    alcohol    quality
Min. :2.720  Min. :0.2200  Min. : 8.00  Min. :3.000
1st Qu.:3.090 1st Qu.:0.4100 1st Qu.: 9.50 1st Qu.:5.000
Median :3.180 Median :0.4700 Median :10.40 Median :6.000
Mean   :3.188 Mean  :0.4898 Mean  :10.51 Mean  :5.878
3rd Qu.:3.280 3rd Qu.:0.5500 3rd Qu.:11.40 3rd Qu.:6.000
Max.   :3.820 Max.  :1.0800 Max.  :14.20 Max.  :9.000
```

I am showing the command and its output in their entire length, to give you a sense for the kind of output generated by R. If you look through this carefully, you will notice that the two sulfur dioxide columns have values in the tens to hundreds, whereas all other columns have values between 0.01 and about 10.0. That explains a lot — the two sulfur dioxide columns dominate the eigenvalue spectrum simply because they have been measured in units that make the numerical values come out much larger than the other quantities. As I explained earlier, if this is the situation, we need to *scale* the input variables before performing the PCA. We can achieve this, by passing the `scale` option to the `prcomp()` command, like so:

```
pcx <- prcomp( wine, scale=TRUE )
```

Before we examine the result of this operation, I'd like to point out something else. If you look really closely, you will notice that the quality column is not what it claims to be: in the description of the original data set, it was said that quality was graded on a scale from 1 to 10. But looking at the data summary, we see that only grades between 3 and 9 have actually been assigned. Worse, the first quartile is 5 and the third quartile is 6, meaning that at least half of all entries in the data set have a quality ranking of either 5 or 6 — in other

words, the actual range of qualities is much narrower than we might have expected, and strongly dominated by the center. Which makes sense (there are more mediocre wines than tremendous or awful ones), but also makes this data set much less interesting, because whether a wine is ranked 5 or 6 is likely to be a toss-up during the sensory testing.

We can use the `table()` function to see how often each quality ranking occurs in the data set (remember that the dollar-sign is used to select a single column from the data frame):

```
table( wine$quality )
```

	3	4	5	6	7	8	9
20	163	1457	2198	880	175	5	

As we suspected, the middling ranks totally dominate the distribution. We might therefore want to change our goal and instead try to predict the outliers, either good or bad, rather than spending too much effort on the undifferentiated middle.

Coming back to the results of the scaled PCA, we can look at the spectrum of eigenvalues for the scaled version, by using the `summary()` function (again, overloaded!), on the return value of `prcomp()`:

```
summary( ppx )
Importance of components:
```

	PC1	PC2	PC3	PC4	PC5	PC6
Standard deviation	1.829	1.259	1.171	1.0416	0.9876	0.9689
Proportion of Variance	0.279	0.132	0.114	0.0904	0.0813	0.0782
Cumulative Proportion	0.279	0.411	0.525	0.6157	0.6970	0.7752
	PC7	PC8	PC9	PC10	PC11	PC12
Standard deviation	0.8771	0.8508	0.7460	0.5856	0.5330	0.14307
Proportion of Variance	0.0641	0.0603	0.0464	0.0286	0.0237	0.00171
Cumulative Proportion	0.8393	0.8997	0.9460	0.9746	0.9983	1.00000

No single eigenvalue dominates now, and the first five (out of twelve!) eigenvalues only amount to 70% of the total variance. That's not encouraging — it does not seem as if we can significantly reduce the number of variables this way.

As a last attempt, we can create a biplot. This is again very simple, all we have to do is execute (figure 14.5):

```
biplot( ppx )
```

This is actually a fascinating graph! We see that three of the original variables are parallel to the first principal component (the horizontal axis), namely alcohol content, sugar content, and density. Moreover, alcohol content is aligned in the opposite direction to the other two quantities.

But this makes utmost sense! If you recall from chemistry class, alcohol has lower density than water, and sugar syrup has a higher density. So this is

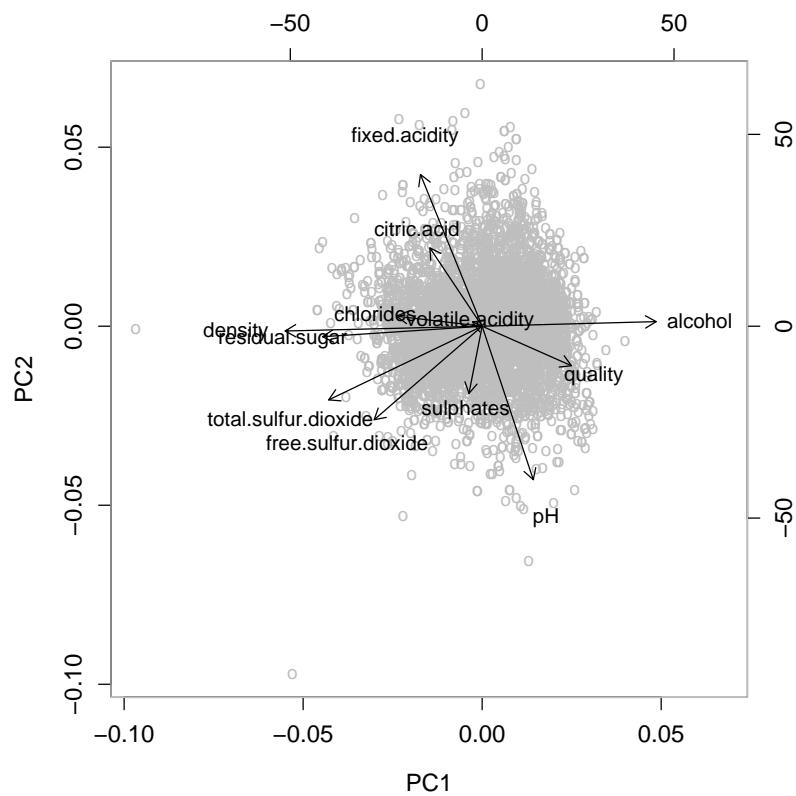


Figure 14.5: TBD

reminding us that density, sugar concentration, and alcohol content are not independent: if you change one, the others will change accordingly. And because these variables are parallel to the first principal component, we can conclude that the overall density of the wine is an important quantity.

The next set of variables that we can read off are the fixed acidity, the citric acid concentration, and the pH value. Again, this makes total sense: the pH is a measure of the acidity of a solution, and it is measured in such a way that increasing the pH indicates less acidity. In other words, these three variables are also at least partially redundant.

The odd one out then is the overall sulfur content, as combination of sulfur dioxide and sulphate concentration.

And finally, I find it interesting to see that the quality seems to be a mix of the alcohol content and the acidity. This suggests that the more alcoholic and the less sour the wine, the more higher it is ranked. Quite a reasonable conclusion!

We could have inferred all of this from the original description of the data set, but I must say that I at least failed to see these connections when initially scanning the list of columns. In this sense, the PCA has been tremendously helpful to me in interpreting and understanding the content of the data set.

Finally, I'd like to reflect one more time on our use of R in this example. This little application demonstrates both the power and the shortcomings of R. On the one hand, R comes with lots of high-level, powerful functions built in, often for quite advanced statistical techniques (even an unusual and specialized graph like a biplot can be done in a single command!). On the other hand, the heavy reliance on high-level functions with implicit behavior leads to opaque programs that make it hard to understand what exactly is going on. For example, such a critical question as deciding whether to rescale the input data or not is handled as a rather obscure option to `prcomp()` command. In particular the frequent use of overloaded functions, that can exhibit widely differing functionality depending on their input, makes it difficult to predict the precise outcome of an operation, and makes discovering ways to perform a specific action uncommonly difficult.

14.7 Further Reading

- *Introduction to Multivariate Analysis*. Chris Chatfield, Alexander Collins. Chapman & Hall/CRC. 1981.
A bit dated, but still one of the most practical, hands-on introductions to the mathematical theory of multivariate analysis. The section on PCA is particularly clear and practical, but entirely skips computational issues and makes no mention of the SVD.
- *Principal Component Analysis*. I. T. Jolliffe. Springer Verlag. 2nd ed., 2002.
The definitive reference on Principal Component Analysis. Not an easy read.

- *Introduction to Data Mining.* Pang-Ning Tan, Michael Steinbach, Vipin Kumar. Addison Wesley. 2005.

My favourite reference on data mining. The presentation is compact and more technical than in most other books on this topic.

14.7.1 Linear Algebra

Linear Algebra is a foundational topic. It is here that one encounters for the first time abstract concepts such as spaces and mappings treated as objects of interest in their own right. It takes time and some real mental effort to get used to these notions, but one gains a whole different perspective on things.

The material is also of immense practical value, in particular the central result, which is the Spectral Decomposition Theorem. Its importance cannot be overstated: it is being used in *every* multi-dimensional problem, in maths, science, engineering.

However, the material is abstract and unfamiliar, and that makes it hard for the beginner. Most introductory books on Linear Algebra try to make the topic more palatable by emphasizing applications, but that only ends up being even more confusing, because it never becomes clear why you need all that abstract machinery when looking at elementary examples. The abstract notions which are at the heart of Linear Algebra are best appreciated, and also most easily understood, when treated in their own right.

The books below are the ones that I have found to be most helpful in this regard.

- *Linear Algebra Done Right.* Sheldon Axler. 2nd ed., Springer. 2004.

The book lives up to its grandiose title. Treats linear algebra as abstract theory of mappings, but on a very accessible, advanced undergraduate level. Highly recommended, but probably not as first book on the topic.

- *Matrix Methods in Data Mining and Pattern Recognition.* Lars Eldén. SIAM. 2007.

This short book is an introduction to linear algebra, with a particular eye to applications in data mining. The pace is fast, and probably requires at least some previous familiarity.

- *Understanding Complex Datasets: Data Mining with Matrix Decompositions.* David Skillicorn. Chapman & Hall/CRC. 2007.

An advanced book, mostly on applications of the SVD and its variants.

- *A Singularly Valuable Decomposition: The SVD of a Matrix.* Dan Kalman. The College Mathematics Journal, vol. 27; p2-23. 1996.

This article, which can be found on the web, is a nice introduction to the SVD. Not for beginners, though.

Chapter 15

Intermezzo: When More is Different

When dealing with some of the more computationally intensive data analysis or mining algorithms, you may encounter an unexpected obstacle: *the brickwall*. Programs or algorithms that seemed to work just fine turn out to not work once in production. And I don't mean: work slower than expected. I mean: not work at all!

Now, performance or scalability problems are familiar to most enterprise developers, but the kinds of problems that arise in data-centric or compute-intensive applications are different, and most enterprise programmers (and, in fact, most Computer Science graduates) are badly prepared for them.

Let's try an example — table 15.1 shows the time (in seconds) to perform 10 matrix multiplications for square matrices of different size. (The details of matrix multiplication don't really concern us here, suffice it to say that it is the basic operation in almost all problems involving matrices and is at the heart of operator decomposition problems, including the Principal Component Analysis we saw in chapter 14.)

Size n	Time [seconds]
100	0.00
200	0.06
500	2.12
1000	22.44
2000	176.22

Table 15.1: TBD

Would you agree that the data in table 15.1 does not look too threatening? For a 2000×2000 matrix, the time required is a shade under three minutes — how long might it take to perform the same operation for a 10000×10000 matrix? Five, maybe ten minutes? Yeah, right. It takes *five HOURS!* And if you

need to go a little bit bigger still — say 30000×30000 , the computation will take five *days*.

What we observe here is typical of many computationally intensive algorithms: they consume disproportionately more time as the problem size gets larger. Of course we have all heard about this in school, but our intuition for the reality of this phenomenon is usually not very good. Even if we run a few tests on small data sets, we fail to spot the trouble: yeah, sure, the program takes longer as the data sets get larger, but it all seems quite reasonable. Nevertheless, we tend to be unprepared for what appears to be a huge *jump* in the required time as we increase the data set by a seemingly not very large factor (remember: what took us from three minutes to five hours was an increase in the problem size by a factor of 5 — not even an order of magnitude!).

The problem is that, unless you have explicitly worked on either a numerical or a combinatorial problem in the past, you probably have never encountered the kind of scaling behavior exhibited by computational or combinatorial problems. That skews our perception.

What is the most likely place that one encounters perceptible performance problems in an enterprise environment? Answer: Slow database queries! I think we all have encountered the frustration resulting from queries that perform a full table scan, rather than using an indexed lookup (regardless whether no index is available or the query optimizer fails to use it). Yet, a query that performs a full table scan rather than using an index exhibits one of the most benign forms of scaling: from $\mathcal{O}(\log n)$ (meaning that the response time is essentially insensitive to the size of the table) to $\mathcal{O}(n)$ (meaning that doubling the table size will double the response time).

By contrast, matrix operations such as the matrix multiplication we encountered in the example earlier, scale as $\mathcal{O}(n^3)$, meaning that if the problem doubles in size, the time required grows by a factor of *eight* (because $2^3 = 8$). In other words, as you go from a 2000×2000 matrix to a 4000×4000 matrix, the problem will take almost ten times as long, and if you go to a 10000×10000 matrix, it will take $5^3 = 125$ times as long. Ooops.

And this is the good news. Many combinatorial problems (such as the Traveling Salesman Problem and similar problems) don't scale according to a power law (such as $\mathcal{O}(n^3)$), but instead scale exponentially ($\mathcal{O}(e^n)$). In these cases, you will reach the brickwall *much* faster and much more brutally. For such problems, an incremental increase in the size of the problem (that is, from n to $n + 1$) will typically at least *double* the runtime: in other words, the last element to calculate takes as much time as all the previous elements *taken together*. System sizes of around $n = 50$ are frequently the end of the line. With extreme effort you might be able to push it to $n = 55$, but $n = 100$ will be entirely out of reach.

The reason I stress this kind of problem so much is that in my experience not only are most enterprise developers unprepared for the reality of it, but also that the standard set of software engineering practices and attitudes is entirely inadequate to deal with them. I once heard a programmer say: "It's all just engineering" in response to challenges about the likely performance problems

of a computational system he was working on. Nothing could be further from the truth: no amount of low-level performance tuning will save a program of this nature that is algorithmically hosed — and no amount of faster hardware, either. Moreover, “standard software engineering practices” are either of no help, or are even entirely inapplicable (we’ll see an example in a moment).

Most disturbing to me was his casual, almost blissful ignorance — coming from a guy who definitely *should* have known better.

15.1 A Horror Story

I was once called into a project in its thirteenth hour — they had far exceeded both their budget and their schedule, and were about to be shut down for good, because they could not make their system work. They had been trying to build an internal tool, which was intended to solve what was essentially a combinatorial problem. The tool was supposed to be used interactively: the user supplies some inputs, and gets back an answer within a few minutes, at the most. By the time I was brought in, the team had labored for over a year, but the minimum response time their system achieved exceeded *twelve hours*, despite the fact that it ran on a very expensive (and very expensive to operate) super-computer.

It took me two weeks to come up with an improved algorithm, which calculated answers in real time and was able to run on a laptop. Ahem...

No amount of “engineering” will be able to deliver that kind of speed-up.

(How did I do it? By attacking the problem on many different levels. First of all, I made sure I *understood the problem domain*. The original project team had always been a little vague about what specifically the program was trying to calculate, with the result that their “domain model” was not truly logically consistent, so I first put the whole problem on sound mathematical footing. Second, I *redefined the problem*: the original program had attempted to calculate a certain quantity by explicit enumeration of all possible combinations. My solution calculated an approximation instead — which was warranted, since the input data was not known very precisely, anyway, and I was able to show that the uncertainty introduced through my approximation was less than the uncertainty already present in the data. Third, I *treated hot spots differently than the happy case*: the new algorithm contained a facility to calculate the result to higher accuracy, but did so only if the added accuracy was needed. Fourth, I worked together with the principal engineer to make sure we used efficient data structures. We also implemented some core pieces ourselves, rather than relying on general-purpose libraries; and judiciously pre-calculated and cached some frequently-used, intermediate results. Besides putting the whole effort on a conceptually consistent footing, the most important contribution was the change in the problem definition — dropping the exact approach, which was unnecessary and infeasible, and instead adopting an approximate solution, which was all that was required and cheap.)

15.2 Some Suggestions

Computational and combinatorial programming is really different. It runs into different limits and requires different techniques. Most important is the appropriate choice of algorithm at the outset, since no amount of low-level tuning or “engineering” will save a program that is algorithmically flawed.

Here is a list of recommendations, in case you find yourself setting out on a project that involves heavy computation or that deals with combinatorial complexity issues:

Do your homework Understand computational complexity, know the complexity of the algorithm you intend to use, research the choice of algorithms (and their trade-offs) available for your kind of problem. Read broadly — although the exact problem as specified may turn out to be intractable, you may find that a small change to the requirements may lead to a much simpler problem. It is definitely worth re-negotiating the problem with the customer or end-users, rather than setting out on a project that is infeasible from the outset. (Skiena’s “Algorithm Design Manual” is a particularly good resource for algorithms grouped by problems.)

Run a few numbers Do a few tests with small programs and evaluate their scaling performance. Don’t just look at the actual numbers themselves — also consider the scaling behavior as you vary the problem size! If the program is not showing the scaling behavior you would have expected theoretically, it has a bug. If so, fix the bug before proceeding! (In general, algorithms follow the theoretical scaling prediction very closely, for all but the very smallest of problem sizes.) Extrapolate to real-sized problems: can you live the expected run-time predictions?

Forget standard software engineering practices It is a standard assumption in current software engineering that developer time is the scarcest resource, and that programs should be designed and implemented accordingly. Computationally intensive programs are one situation where this is not true: if you are likely to max out the machine, it is worth having the developer go the extra mile, rather than the computer — additional developer time may very well make the difference between an “infeasible” problem and a solved one.

In situations where you are pressed for space, for instance, it might very well make sense to write your own container implementations, rather than to rely on the system-provided hashmap. Watch out for the trap of conditioned thinking, though: in one project that I worked on, we knew that we would have a memory-size problem, and that we therefore would have to keep the size of individual elements small. On the other hand, it was not clear at the outset, whether the 4-byte Java `int` data type would be sufficient to represent all required values, or whether we would have to use the 8-byte Java `long` type, whereupon someone suggested to

wrap the atomic data type in an object, so that we could swap out the implementation in case the 4-byte `int` would turn out to be insufficient. That's a fine answer in a standard software engineering scenario ("encapsulation" and all that), but in the current situation, where space was at a premium, it was entirely missing the point: the space that the Java wrapper would have consumed (in addition to its data members) would have been larger than the payload!

Remember: standard software engineering practices are typically intended to trade machine resources for developer resources. However, for computationally intensive problems, machine resources (not developer time) are the limiting factor!

Don't assume that parallelization will be possible Don't assume that you will be able to partition the problem in such a way that simultaneous execution on multiple machines (that is: parallelization) will be possible, until you have developed an actual, concrete, implementable algorithm — many computational problems don't parallelize well. Even if you can come up with a parallel algorithm, performance may be disappointing: hidden costs (such as communication overhead) often lead to a performance that is much poorer predicted — a cluster consisting of twice as many nodes often exhibits a behavior *much* less than double the original one! Running realistic tests (on realistically sized data sets and on realistically sized clusters) are harder to do than for single processor implementations — but even more important!

Leave yourself some margin Assume the problem size will be larger by a factor of 3 and that hardware will only deliver 50% of theoretical performance.

If the results are not wholly reassuring, explore alternatives Take the results for the expected run-time and memory requirements that you obtained from theoretical predictions and the tests that you have performed seriously. Unless you seem to be able to meet your required benchmarks *comfortably*, explore alternatives: consider better algorithms, research whether the problem can be simplified or whether the problem can be approached in an entirely different manner, look into approximate or heuristic solutions. If you feel yourself stuck, get help!

If you can't make it work on paper, STOP It won't work in practice either. It is a surprisingly common anti-pattern to see the warning signs early, but to press on regardless, based on hopeful optimism that "things will work themselves out during implementation". This is entirely misguided: nothing is going to work out better as you proceed with an implementation; instead, everything is going to be a bit worse than expected.

Unless you can make it work on paper, and make it work *comfortably*, there is no point in proceeding!

The recurring recommendation here is that nobody is helped by a project that ultimately fails — because it was impossible (or at least: infeasible) from the get-go. Unless you can demonstrate at least the feasibility of a solution (at an acceptable price point!), there is no use in proceeding. And everybody is much better off knowing this ahead of time.

15.3 What about map/reduce?

Won't the map/reduce family of techniques make most of these considerations obsolete? The answer, in general, is: *No*.

It is important to understand that map/reduce is not per se a clever algorithm, or actually an algorithm at all. It is a piece of *infrastructure* that makes naive algorithms convenient.

That's a whole different ball game. The map/reduce approach does not speed up any particular algorithm at all. Instead, it makes the parallel execution of many sub-problems convenient. For map/reduce to be applicable, therefore, it must be possible to *partition* the problem, in such a way that individual partitions don't need to talk to each other. Search is such an application that is trivially parallelizable, and many (if not all) successfull current applications of map/reduce that I am aware seem to be related to generalized forms of search.

This is not to say that map/reduce is not a very important advance. (Any device that makes an existing technique orders of magnitudes more convenient is an important innovation!) But right now, we are still in the process of figuring how which problems are most amenable to the map/reduce approach, and how best to adapt them. I suspect that the algorithms that will work best on map/reduce will *not* be straightforward generalizations of serial algorithms, but will instead be algorithms that would be entirely unattractive on a serial computer.

It is also worth remembering that parallel computation is not new. What has killed it in the past was the need for different partitions of the problem to communicate with each other: very quickly, the associated communication overhead annihilated the benefit from parallelization. This problem has not gone away, it is merely masked by a current emphasis on search and search-like problems, which allow trivial parallelization without any need for communication among partitions. I worry that more strictly computational applications (such as the matrix multiplication problem we discussed earlier, or the simulation of large physical systems) will require so much sharing of information among nodes that the map/reduce approach will appear unattractive.

Finally, over the excitement currently generated by map/reduce, it should not be forgotten that its total cost of ownership (including the long-term *operational* cost of maintaining the required clusters and the associated network and storage infrastructure) is not yet known. Although map/reduce installations make distributed computing "freely" available to the individual programmer, the required hardware installations and their operations are any-

thing but “free”.

In the end, I expect map/reduce to have an effect similar to the one that compilers had when they came out: yes, the code that they produced was less efficient than hand-coded assembler. Yet the overall efficiency gain far outweighed this local disadvantage.

But keep in mind that even the best compilers have not neither made Quicksort obsolete, nor indexed lookup.

15.4 Workshop: Permutations

Sometimes, one has to see it to believe it. In this spirit, let’s write a program that calculates all permutations (all re-arrangements) of a set. (That is, if the set is $[1, 2, 3]$, the program will generate $[1, 2, 3]$, $[1, 3, 2]$, $[2, 1, 3]$, $[2, 3, 1]$, $[3, 1, 2]$, $[3, 2, 1]$.) You can imagine this routine to be part of a larger program: for example, to solve the Traveling Salesman problem exactly, one needs to generate all possible trips (that is: all permutations of the cities to visit) and evaluate the associated distance.

Of course, we all “know” that the number of permutations grows as $n! = 1 \cdot 2 \cdot 3 \cdots n$, where n is the number of elements in the set, and that the factorial function grows “quickly”. Nevertheless, you have to see it to believe it. (Even I was shocked by what I found when I developed and ran the program below!)

The program in listing 15.1 reads a positive integer n from the command line, and then generates all permutations of a list of n elements, using a recursive algorithm. (It successively removes one element of the list, generates all permutations of the remainder, then tacks the removed element back on to the results.) The time required is measured and printed.

(You may object to the use of recursion here, pointing out that Python does not allow infinite depth of recursion. That is true, but no factor: we will run into trouble long before this particular constraint comes into play.)

I highly recommend that you try it. Because we know (or suspect) that this program might take a while if the number of elements is large, we probably want to start out with 3 elements. Or with 4. Then maybe we try 5, 6, or 7. In all cases, the program finishes almost *instantaneously*. Then go ahead and run it with $n=10$. Just 10. Go ahead, do it. (But I suggest you save all files and clean up your login session first, so that you can reboot without loosing too much work if you have to...)

Go ahead. You have to *see* it to believe it!¹

¹Anybody who scoffs that this example is silly, because “you should not store all the intermediate results — use a generator”, or because “everyone knows you can’t find all permutations exhaustively — use heuristics” is absolutely correct — and entirely missing my point. I know that my implementation is naive, but — cross your heart — would you truly have assumed that the naive implementation would be in trouble for $n = 10$? In particular, after it didn’t even blink for $n = 7$?

```

import sys, time

def permutations( v ):
    if len(v) == 1: return [ [v[0]] ]

    res = []
    for i in range( 0, len(v) ):
        w = permutations( v[:i] + v[i+1:] )
        for k in w:
            k.append( v[i] )
        res += w

    return res

n = int(sys.argv[1])
v = range(n)

t0 = time.clock()
z = permutations( v )
t1 = time.clock();

print n, t1-t0

```

Example 15.1: TBD

15.5 Further Reading

- *The Algorithm Design Manual*. Steven S. Skiena. 2nd ed., Springer. 2008. This is an amazing book, because it presents algorithms not as abstract entities to be studied for their own beauty, but as potential solutions to real problems. In its second half, it provides a self-titled “hitch-hiker’s guide to algorithms”, which is a catalog of different algorithms for common problems. It helps you find an appropriate algorithm by asking detailed questions about your specific problem instance and includes pointers to existing implementations. In addition, the author’s “War Stories” of past successes and failures in the real world provide a vivid reminder that algorithms are *real*.

Part IV

Applications: Using Data

Chapter 16

Reporting, Business Intelligence, and Dashboards

Data analysis does not just consist of crunching numbers. It also includes navigating the context and environment in which the need for data analysis arises. In this chapter and the next, I am going to look at two areas that often have a demand for data analysis and analytical modeling, but that tend to be unfamiliar if you come from a technical background: in the present chapter, we are going to talk about Business Intelligence and Corporate Metrics, and in the next chapter we will discuss financial calculations and business plans.

This material may seem a little out of place, because it is largely not technical. But that is precisely why it is important to be included here: to a person with a technical background, this material is often totally new. On the other hand, it is precisely in these areas that sound technical and analytical advice is often required: the primary consumers of these services are “business people”, who may not have the necessary background and skills to make appropriate decisions without help. This puts a additional responsibility on the person working with the data to understand the problem domain thoroughly, in order to make suitable recommendations.

This is no joke. I have seen otherwise very smart people at high-quality companies completely botch business metrics programs, simply because they lacked basic software engineering and math skills. As the person who (supposedly!) “understands data”, I see it as part of my responsibility to understand what my clients actually want to *do* with the data — and advise them accordingly on the things they *should* be doing. But to do so effectively, it is not enough to understand the data — I also need to understand my clients.

That’s the spirit in which these chapters are intended: to describe some of the ways demand for data that arises in a business environment, to highlight some of the traps for the unwary, and hopefully to give some advice on using data more successfully.

16.1 Business Intelligence

Businesses have been trying to make use of the data that they collect for years, and in the process accumulated a fair share of disappointments. I think we need to accept that the problem is hard: you need to find a way to represent, store, and make accessible a comprehensive view of all available data, in such a way that is useful to anybody and for any purpose. That's just hard. In addition, to be comprehensive, such an initiative has to span the entire company (or at least a very large part of it), which brings with it a whole set of administrative and political problems.

This frustrating state of affairs has brought forth a number of attempts to solve what is essentially a conceptual and political problem using *technical* means. In particular the large enterprise tool vendors saw (and see) this problem space as an opportunity!

The last iteration on this theme was data warehouses — that is, long term, comprehensive data stores, in which data is represented in a denormalized schema, which is (hopefully) more general than the schema of the transactional databases and easier to use for the non-technical user. Data is imported into the data warehouse from the transactional databases using *ETL* (extraction, transformation, and load) processes.

Overall, there seems to be a feeling that data warehouses fell short of expectations, for three reasons. First of all, since data warehouses are enterprise-wide, they respond slowly to changes in any particular business unit. In particular, changes to the transactional data schema tend to propagate into the data warehouse at a glacial pace, if at all. The second reason is that accessing the data in the data warehouse never seems to be as convenient as it should be. The third and final reason is that doing something useful with the data (once obtained) turns out to be hard — partially (but only partially), because the typical query interface is often clumsy and not designed for analytic work.

While data warehouses were the most recent iteration in the quest for making company data available and useful, the current iteration comes under the name *Business Intelligence* or BI. The term is not new (Wikipedia tells me that it was first used in the 1950s), but it has only been in the last one or two years that I have seen the term used with regularity.

The way I see it, Business Intelligence is an accessibility layer sitting on top of a data warehouse or similar data store, trying to make the underlying data more useful through better reporting, improved support for ad-hoc data analysis, and even some attempts at canned predictive analytics.

Because it sits atop a database, all Business Intelligence stays squarely within the database camp, and what it aspires to do is constrained by what a database (or: a database developer!) can do. The “analytics” capabilities mostly consist of various aggregate operations (sums, averages, etc), typically supported by *OLAP cubes* (Online Analytical Processing). OLAP cubes are multi-dimensional contingency tables (that is: more than two dimensions), which are pre-computed and stored in the database, and which allow for (relatively) quick summaries or projections along any of the axes. In a way, such “cubes” are spreadsheets-on-

steroids, which makes them very familiar and accessible to the large number of people comfortable with spreadsheets and pivot tables!

In my experience, the database heritage (in contrast to a software engineering heritage) has another consequence, in the way the people involved with Business Intelligence relate to it. While almost all software development has an element of *product* development to it, Business Intelligence often feels like *infrastructure* maintenance. And while the purpose of the former typically involves innovation and the development of new ways to please the customer, the latter tends to be more reactive and largely concerned with keeping the trains on time. This is not necessarily a bad thing, as long as one pays attention to the difference in cultures.

What is the take-away here? First of all, I think it is important to have realistic expectations: when it comes right down to it, Business Intelligence initiatives are mostly about better reporting. That is fine as far as it goes, but does not require (or provide) much data analysis per se. The business users who are the typical customers of such projects usually don't need much help in defining the numbers they would like to see. There may be a need for help with visualization and overall user interface design, but the possibilities here tend to be mostly defined (that is: limited) by the set of tools that is being used.

More care needs to be taken when any of the "pre-canned" analysis routines are being used that come bundled with many BI packages. Most (if not all) of these tools are freebies, thrown in by the vendor to pad the list of supported features, but are likely to lack production strength, and instead are going to emphasize "ease of use". They will produce results alright — but it will be up to us to decide how *significant* and how *relevant* these results are likely to be.

We should first ask what these routines really do under the hood. For example, a "clustering" package may be employing any one from a whole range of clustering algorithms, as we have seen in chapter 13, or even use a combination of algorithms together with various heuristics. Once we understand what the package does, we can then begin asking questions concerning the quality and in particular the significance of the results. Given that the routine is largely a black-box to us, we are not going to have an intuitive sense regarding the extend of the region of validity of its results, for example. And because it is intended as an easy-to-use give-away, it is not likely to have support for or report at length about nasty details such as confidence limits on the results. Finally, we should ask how relevant these results are going to be. Was there an original question that is being addressed or was the answer mostly motivated by the ease with which it could be obtained?

One final observation: when there are no commercial tool vendors around, there is not much momentum to develop Business Intelligence implementations. Neither of the two major open-source databases (MySQL and Postgres) has developed BI functionality or the kinds of ad-hoc analytics interfaces that are typical for BI tools. (There are a few open-source projects, which provide reporting and OLAP functionality, though.)

16.1.1 Reporting

The primary means by which data is used for “analysis” purposes in an enterprise environment is via reports. Much of “Business Intelligence” (whether we like it or not), revolves around reporting, and in general “reporting” is usually a pretty big part of what companies do with their data.

It is also one of the greatest sources of frustration. Given the ubiquity of reporting, and the resources spent on it, one would think that the whole area would be pretty well figured out by now. But this is not so: in my experience, nobody seems to like what the reporting team is putting out — including the reporting team itself.

I have come to the conclusion that reporting, as currently understood and practiced, has it all wrong. Reporting is the one area of the software universe which so far has not been touched in any way by the notion of “agility” and “agile development”. Reporting solutions are invariably big, bulky, and bureaucratic; slow to change and awkward to use. Moreover, I think with regards to two specific issues, they get it *exactly* wrong:

- In an attempt to conserve resources, reporting solutions are often built generically: a single reporting system, which supports all needs by all users. The reality, of course, is that the system does not serve the need of *any* user (certainly not well), but that on the other hand the overhead from the general-purpose architecture drives the cost through the roof.
- Most reporting that I have seen confuses “up-to-date” with “real-time”: data for reports is typically pulled in immediate response to a user’s query — ensuring that the data is up to date, but also (for many reports) that it will take a while for the report to be available. Often quite a while! I believe that this delay is the single greatest source of frustration with all reports, anywhere. For a user it typically matters much more to get the data *right this minute* rather than to get it *up to this minute*!

Can we conceive of an alternative to the current style of reporting? One that actually delivers on its promise and is easy and fun to use? I think so (in fact, I have seen it in action!), but first we need to slaughter a sacred cow: namely that *one* reporting system should be able to handle all kinds of *different* requirements. In particular, I think it will be helpful to distinguish very clearly between *operational* and *representative* reports.

Representative reports are those that are intended for external users. Quarterly filings certainly fall into this category, but also reports the company may provide to its customers on various metrics: anything that gets published.

By contrast, operational reports are those that are used by managers within the company to actually run the business. Here you find things like the number of orders shipped today, or the size of the backlog, or the CPU loads of various servers.

These two kinds of reports have almost nothing in common! Operational reports need to be fast and convenient — little else matters. Representative reports need to be definitive and optically impressive. It is not realistic to expect

a single reporting system to support both requirements simultaneously! I'd go further and say that the preparation of representative reports is always somewhat of special operation and therefore should be treated as such: "making it look good". If you have to do this a lot (for example because you regularly send invoices to a large number of customers), then by all means automate the process — but don't kid yourself into thinking that this is still merely "reporting". (Billing is a core business activity for all service businesses!)

When it comes to operational reports, here are some ideas to consider.

Think "simple, fast, convenient" Reports should be simple to understand, quick (instantaneous) to run, and convenient to use. Convenience dictates that the users must not be required to fill in an input mask with various parameters — the most the user can be expected to do is to select one specific report from a fixed list of available ones.

Don't waste real estate... The whole point of having a report is the *data*. Don't waste space on other things, in particular if they never change! I have seen reports in which fully one third of the entire screen was taken up with a header, showing the company logo! In another situation, a similar amount of space was taken up by an input mask. Column headers and explanations are another culprit: once people have seen the report twice, they will know what the columns are! (You will still need headers, but they can be short.) Move explanatory material to a different location and provide a link to it. Remember: the reason people ran the report is the *data*.

...but make reports easy to read In particular, this means putting lots of data onto a single page, which can be read by scrolling (rather than forcing the user to page through the report). Use a large enough font! Possibly consider (gently!) highlighting every second line. Less is more.

Provide raw data, and leave filtering and aggregation to the user This is a potentially radical idea — instead of providing a complicated input mask, in which the user can specify a bunch of selection criteria and the set of columns to return, a report can simply return *everything* (within reason, of course) and leave it to the user to do any amount of filtering and aggregation required. This idea is based on the realization that most people who actually look at reports are going to be comfortable with Excel. (I said: "most people" — and for the majority of people in the business world, "spreadsheet" means: Excel.) Hence, we can regard a report not as an end product, but as a data feed for Excel.

This has a number of advantages: it is simple, cheap, and flexible (because users are free to design their own reports themselves). It also implies that the report needs to include additional columns, which are required for user-level filtering and aggregation.

Consider cached reports, rather than real-time queries Having removed the input mask, the content that a report shows is basically fixed. But once

it is fixed, it can be run ahead of time and cached — which means that we can return the data to the user instantaneously. It also means that the database is hit only once, no matter how often the report is viewed.

Find out what your users are doing with your reports — and then try to provide it for them

The can't tell how often I have witnessed the following scenario: The reporting team spends significant time and effort worrying about the details and layout of its reports. But a couple of doors down the hall, the first thing that the actual users of a report do with the report is to cut-and-paste the results out of the reporting system and import it into, yes, Excel. And then they often spend a lot of time (like: an hour, or so), manually editing and formatting the results, so that they actually show the information that the users actually need. They do this *every day* (or every week, or every hour — every time the need to access the report).

These edits are often painfully simple: the users need the report sorted on some numerical column, but they can't just sort on it, because the entry in this column is text: "Quantity 17". Or they need the difference between two columns, rather than the raw values. Whatever it is — it is often the kind of thing that would take less than 30 minutes to implement for the reporting team, and solve the problem once and for all. (These things don't tend to come up in formal "requirements" meetings, but become immediately apparent if you spend a couple of hours with a user, going through his or her daily routine.)

Reports are for consumers, not producers A common response to the previous item is that every user seems to have his or her own unique set of needs, and trying to meet all of them would lead to a proliferation of different reports.

There is of course some truth to that. But in my experience, certain reports are used by work groups in a pretty standard fashion. It is those situations that the time wasted on repetitive, routine editing tasks like the ones I just described are particularly painful — and avoidable. In these cases, it might also be worthwhile to work with the group (or its management) to standardize their processes, so that in the end a single report meets everybody's need.

But there is a bigger question here, too: Whose convenience is actually more important? The producers', or the users'? Or, even more broadly: whom are reports *for*? For the reporting team, or for the people looking at them?

Think about the proper metrics to show For reports that show some form of summary statistics (as opposed to raw counts), consider which quantities precisely to show. Will a mean (such as the "average time spent in queue") be sufficient, or is the distribution of values skewed, so that you should consider a median? Do you need to include a measure for the width of the distribution (standard deviation or inter-quartile range)?

(Answer: probably!). Also, don't neglect cumulative information! (See chapter 2.)

Don't mix drill-down functionality with reporting This is possibly a controversial item. In my opinion, reports are exactly that: standard overviews of the status of the system. Every time I run a report, I expect to find the same picture. (The numbers will change, of course, but not the overall view.) Drill-downs, on the other hand, are always different. After all, they are usually conducted in response to something out of the ordinary. As such, I don't think it makes sense to try and provide a general-purpose framework for them: ad-hoc work is best done using ad-hoc tools.

Consider this: general-purpose frameworks are always clumsy and expensive, yet rarely deliver the functionality required. Would it be more cost effective to forget about maintaining drill-down functionality in the reporting system itself, and instead deploy the resources (that is: developers) liberated this way to work on drill-down tasks on an ad-hoc basis?

Don't let your toolset strangle you Don't let your toolset limit the amount of value you can deliver. Many reporting solutions that I have seen can be awfully limiting — in regards to the kind of information that you can display and in terms of the formatting that is possible. As with any tool: if it gets in the way, evaluate again whether it is a net gain!

This is the list. I think the picture I am trying to paint is pretty clear: fast, *simple*, and convenient reports, showing lots of data, but little else. Minimal overhead, and a preference for cheap one-offs as opposed to expensive general-purpose solutions. It's not all roses — in particular the objection that a large number of cheap one-off reports can incur a significant total cost of ownership is well taken. On the other hand, every general-purpose reporting solution that I have seen incurred a similar cost of ownership — but did not deliver the same level of flexibility and convenience.

I think it is time that we rethink reporting. The agile movement (whether right or wrong in all detail) has brought fresh life to software development. We should start applying its lessons to reporting.

Finally, a word about reporting tools. The promise of reporting tools that I have seen is to take in data from "many sources" and to deliver reports to "many formats" (such as HTML, PDF, and Excel).

I have already stated why I believe this to be largely an imaginary problem: I cannot conceive of a situation where you really need to deliver the same report as both HTML and PDF. The HTML report is likely to be an operational report, whereas the PDF report is going to be representational. Chances are, there are additional differences between them, besides the sheer output format (in terms of content, life cycle, and audience)!

Similarly for the need to pull data from many sources. Although that *does* occur, does it occur sufficiently frequently that it needs to form the basis for the entire reporting architecture? Or does in reality most of the data come from

relational databases, and the odd case where some information comes from an XML document, or an LDAP server, or a proprietary data store is best handled as a special case? (And if you seriously need to pull data from very different sources, you are better off implementing a proper intermediate layer, which extracts and stores data from all sources in a solid, common format. Reporting requires a solid and reliable data model: in other words, you want to isolate your reporting solution from the vagaries of the data sources — in particular if these sources are “weird”.)

The kinds of problems that reporting tools promise to solve strike me as classic examples for situations where a framework *seems* like a much better idea than it actually is. Sure, a lot of the tasks involved in reporting are lame and repetitive. However, designing a framework that really has the flexibility required to function as a general-purpose tool is difficult, leading to frameworks that are hard to use for everyone (and you still have to work around their limitations). The alternative is to write some boring, but straightforward, and most of all *simple* boilerplate code that solves *your* specific problems simply and well.

16.2 Corporate Metrics and Dashboards

It is always surprising when a company doesn’t have good, real-time, and consistent visibility into some of its own fundamental processes. It can be amazingly difficult to get insight into data such as: orders fulfilled today, orders still pending, revenue by item type, and so on.

But one should not be so surprised, because up close the problem is harder than it appears. Any business of sufficient size will have complex business rules, which in addition may be inconsistent across divisions or include special exceptions for major customers. The IT infrastructure which provides the data will have gone through several iterations over the years and be a mixture of “legacy” and more current systems — none of which primarily designed for our current purposes! The difficulties in presenting the desired data are nothing more than a reflection of the complexity of the business.

You may encounter two concepts, which try to address specifically the visibility problem just described: *dashboards* and more general *metrics programs*. The goal of a metrics program is to *define* those quantities that are most relevant and should be tracked (and might possibly be used for performance evaluation as well), and to design and develop the infrastructure required to collect the appropriate data and to make it accessible. A dashboard might be the visible outcome of a metrics program. The purpose dashboard is to provide a high-level view of all relevant metrics on a single report (as opposed to a dozen individual, more detailed reports). Dashboards often also include information whether any given metric is within the desired range.

Dashboard implementations can get arbitrarily fancy, with various forms of graphical displays for individual quantities. A particularly unfortunate misunderstanding results from taking the word “dashboard” too seriously and popu-

lating the report with graphical images of dials, like the one in a car. Of course, this is beside the point and actually takes away from what is a legitimately useful idea: namely to have a comprehensive, unified view of the whole set of relevant metrics.

I think it is important to keep dashboards simple. Stick to the original idea of all the relevant data on a single page — together with a clear indication of whether some specific value is within the desired range or outside of it.

As I already explained when talking about reports, I do not believe that drill-down functionality should be part of the overall infrastructure: the purpose of the dashboard is highlight areas that need further attention, but the resulting research is better done using individual, detailed research.

16.2.1 Recommendations for a Metrics Program

If you find yourself on a project team to implement a metrics program, including the definition of metrics to track and the design the required infrastructure, here are some concrete recommendations that you might want to consider.

Understand the cost of metrics programs Metrics aren't free. They require development effort and deployment infrastructure of production-level strength, both of which have costs and overhead. Once in production, they will also require regular maintenance. None of this is free.

I think the single biggest mistake is to assume that a successful metrics program can be run as an add-on project, without additional resources. It can't.

Have realistic expectations for the achievable benefit The short-term effect of any sort of metrics program is likely to be small, possibly non-detectable: metrics provide visibility, and *only* visibility, but they don't improve performance. Only the decisions based on these metrics will (may!) improve performance. But here the *marginal* gain can be quite small, because how many of the same decisions would have been taken anyway, based on routine and gut-feeling?

The more important effect of a metrics program lies in the long-term effect it has on the organizational culture: possibly a greater sense of accountability, or even the realization that there *are* different levels of performance, can change the way the business runs. But they take time to materialize.

Start with the actions that the metrics should drive When setting out to define a set of metrics to collect, make sure you ask yourself: what decision would I make differently, based on the value of this metric? If none comes to mind, you don't need to collect it!

Don't define what you can't measure This is a good one. I once participated in a metrics program, where the set of metrics to track had been decided

at the executive management level, based on what would be “useful” to see. Problem was, for a significant fraction of those quantities, no data was being collected, and could not be collected, because of limitations in the physical processes.

Build appropriate infrastructure For a metrics program to be successful, it must be technically reliable and the data must be credible. In other words, the systems to support it must be of *production-level quality* in regards to robustness, uptime, and reliability. For a company of any size, this probably requires databases, network infrastructure, monitoring, the whole nine yards. Plan on them! It will be very difficult to be successful with flat files and a CGI script (or Excel sheets on a SharePoint, for that matter).

There is an important difference here between a more comprehensive *program*, which purports to be normative and widely available, and an ad-hoc report. Ad-hoc reports can be extremely effective, precisely because they do not require any infrastructure beyond a CGI script (or an Excel sheet), but they *do not scale*: they won’t scale to more metrics, larger groups of users, more facilities, longer historical time frames, whatever it is.

On the flip side, if all you need is an ad-hoc report, by all means go for it.

Steer clear of manually collected metrics Two problems: First of all, manually collected metrics are neither reliable nor credible (people will forget to enter numbers, and if pressed, will make them up!). Secondly, most people will resist to enter numbers (in particular at any level of detail — think timesheets!), which will destroy the acceptance and credibility of the program. Avoid manually collected metrics at all cost.

Beware of aggregates It can be very appealing to aggregate values as much as possible: “Just give me *one number* so that I see how my business is going”. The problem is that every aggregation step loses information, which is impossible to regain: you can’t unscramble the egg. And *actionable* information is typically *detailed* information. Knowing that my aggregated performance score has tanked is not actionable, but knowing which *specific* system has failed is!

This leads to the question about user-interface design, roll-ups and drill-downs. I think most of this is unnecessary. All that is required is a simple, high-level report. If details are required, one can always go and dig deeper in an ad-hoc fashion.

Think about the math involved The math required for corporate metrics is rarely very advanced, but it will still contain some opportunity for mistakes. A very common example occurs whenever we are forming a ratio, for example to calculate the defect rate: number of defects, divided by the number of items produced. The problem is that the denominator can become zero (no items produced during the observation time frame), so

that it becomes impossible to calculate a defect rate. There are different ways you can handle this (report as “not available”, treat zero items produced as a special case; particularly slick: add a small number to the denominator in your definition of the defect rate, so that it can never become zero).

There are other problems where careful thinking about the best mathematical representation can be helpful: to compare different metrics, they need to be normalized through rescaling by an appropriate scaling factor. For quantities that vary over many orders of magnitude, it might be more insightful to track the logarithm, rather than the bare quantity. Consider getting expert help: a person with sufficient analytical background will be able to recognize trouble spots *and* make recommendations for the best ways of dealing with them that may be not at all obvious.

Be careful with statistical methods that might now apply Mean and standard deviation are only good representations for the typical value and the typical spread if the distribution of data points is roughly symmetrical. In many practical situations, this is *not* the case — waiting times, for instance, can never be negative, and although the “typical” waiting time may be quite short, there is likely to be a tail of events that took a very long time to complete. This tail will corrupt both mean and standard deviation. In such a case, median-based statistics are a better bet. (See chapter 2 and chapter 9.)

In general it is necessary to study the nature of the data before settling on an appropriate way to summarize it. Again, consider expert help if you don’t have the competency in-house.

Don’t buy what you don’t need It is tempting to ask for a lot of detail that is not really required. Generally, it is not necessary to track sales numbers on a milli-second basis, because we can not respond to changes at that speed. And even if we could, it would not be very meaningful, since sales will fluctuate normally over the course of the day.

Establish a meaningful time scale or the frequency over which to track changes. This time scale will be similar to the time scale in which we can make decisions, and also similar to the time scale after which we will see the results of our decisions. Note that this might vary drastically: daily is probably pretty good for sales, but for the reactor temperature, a much shorter time scale is certainly appropriate!

Don’t oversteer This item is the logical consequence from the previous one. Every “system” has a certain response time with which it reacts to changes. Applying changes more frequently than this response time is useless at best, and quite possibly harmful (because it prevents the system from reaching a steady state).

Learn to distinguish trend and variation Most metrics will be tracked over time, so what we learned in chapter 4 on time-series analysis applies. The most

important skill is to develop an understanding for the duration and magnitude of typical “noise” fluctuations, and to distinguish them from significant changes (trends) in the data. If sales dipped today by 20 percent, then this is no cause for alarm if we know that sales fluctuate by plus or minus 25 percent from day to day. But if sales fell by 5 percent for 5 days in a row, then that’s possibly a warning sign.

Don’t forget the power of perverted incentives... When metrics are used to manage staff performance, this often means going from a vague, yet broad sense of “performance”, to a much narrower focus on specifically those quantities that are being measured. This can set up perverted incentives.

For instance, what is the primary performance metric in a customer-service call-center? The number of calls each worker handles per hour, or “calls-per-hour”. What is the best way for a call-center worker who is evaluated solely on calls-per-hour to improve his or her standing? By picking up the phone when it rings and hanging up immediately! By making calls-per-hour the dominant metric, we have implicitly de-emphasized other important aspects, such as customer satisfaction (that is: quality).

...in particular when combined with availability bias Some quantities are easier to measure than others and therefore tend to receive greater attention. In my observation, productivity is generally easier to measure than quality, with all the unfortunate consequences this entails.

Just because you can’t measure it does not mean it does not exist Soft factors: culture, commitment, fun. But also some very “hard” factors, like customer satisfaction. You can’t measure that — all you can measure directly are proxies (such as return rate). An alternative are surveys, but because people choose themselves whether they reply, the results may be highly misleading. (This is known as *self-selection bias*.)

Above all, keep in mind that a metrics program is intended to help the business by providing visibility. It should never become an end in itself, and it is important to keep in mind that it is an effort to support others, not the other way around.

16.3 Data Quality Issues

All reporting and metrics efforts depend on the availability and quality of the underlying data: if the required data is improperly captured (or not captured at all!), there is nothing to work with!

The truth of the matter is that if a company wants to have a successful business intelligence or metrics program, its data model and storage solution *must*

be designed with reporting needs in mind. By the time the demand for data analysis services rolls around, it is of course a bit late to worry about data modeling!

In my experience, two problems in particular occur frequently when trying to prepare reports or metrics: data may not be *available* or it may not be *consistent*.

16.3.1 Data Availability

Data may not be collected at all, often with the innocent argument that “nobody wanted to use it”. That’s silly: data that’s directly related to a company’s business is always relevant — whether anybody is looking at it right now or not (yet).

If data is not available, that does not necessarily mean that it is not being collected. Data may be collected, but not at the required level of granularity. Or it is collected, but immediately aggregated in a way that loses the details that would be required for later analysis. (For instance, if server logs are aggregated daily into hits per page, we will have lost the ability to associate a specific user to a page, and also have lost information about the order in which pages were visited.)

Obviously, there is a trade-off between the amount of data that can be stored and the level of detail that we can achieve in an analysis. My recommendation: try to keep as much detail as you can, even if you have to spool it out to tape (or whatever off-line storage mechanism is available). Keep in mind that operational data, once lost, can *never* be restored. Furthermore, gathering new data takes *time* and can not be sped up. If you know that you will need data for some planned analysis project, start collecting it *today*. Don’t wait for the “proper” extraction and storage solution to be in place (which can easily take weeks or months to come to completion). If necessary, I don’t hesitate to pull daily snapshots of relevant data to my local desktop, to preserve it temporarily, while a long-term storage solution is being worked on. Keep in mind that every day that data is not collected is another day by which your results will be delayed.

Even if data is in principle collected at the right level of detail, it may still not be available in a practical sense, if the storage schema has not been designed with reporting needs in mind. (I assume here that the data in question comes out of a corporate database — certainly the most likely case by far.) Three problems stand out to me in this regard: lack of revision history, business logic commingled with data, and awkward encodings.

Some entities have a non-trivial lifecycle: orders will go through several status updates, contracts have revisions, and so on. In such cases, it is usually important to preserve the full revision history, that is all lifecycle events. The best way to do this is to model the time-varying state as a *separate entity*. For instance, you might have the `Order` entity (which contains things such as the order ID and the customer ID), and the `OrderStatus`, containing the actual status of the order (placed, accepted, shipped, paid, completed, ...), as well as a timestamp for the time that the status change took place. The current status

is the one with the most recent status change. (A good way to do this is to have two timestamps: `ValidFrom` and `ValidTo`, with the latter one being `NULL` for the current status.) Such a model preserves all the information necessary to study such things as the typical time orders stay in any one state, and so on. (By contrast, the presence of history tables with `OldValue`, `NewValue` columns suggests improper relational modeling.)

The important principle is that data is never *updated* — we only append to the revision history. Keep in mind that every time a database field is updated, the previous value is being lost. Try to avoid whenever you can! (I'd go so far to say that CRUD (create, read, update, delete) is indeed a four letter word. The only two operations that should ever be used are create and read. There may be valid operational reasons to move very old data to offline storage, but the data model should be designed in such a way that we never clobber existing data. In my experience, this point is far too little understood, much less heeded.)

Another common problem is business logic commingled with data, in such a way that by looking at the *data* alone, you can't get an accurate picture of the business. A sure sign of this situation is the statement: "Don't try to read from the database directly — you have to go through the access layer API to get all the business rules." What this is saying is that the DB schema was not designed so that the data can stand by itself: the business rules in the access layer are required to interpret the data correctly. (Another indicator is the presence of long, complicated stored procedures — this is worse, in fact, because it suggests that the situation came about inadvertently, whereas the presence of an access layer is at least proof of some degree of foreplanning.)

The difficulty from a reporting point of view is that a reporting system typically has to consume the data in bulk, whereas application-oriented access layers tend to access individual records or small collections of items. The problem is not the access layer as such — in fact, an abstraction layer between the database and the application (or applications) often makes sense. But it should be exactly that: an abstraction and access layer, without embedded business logic.

The final problem that sometimes arises are weird data representations, which (although complete), make bulk reporting excessively difficult. As an example, think of a database that stores only updates (to inventory levels, for example) but not the grand total. To get a view of the current state, it is now necessary to replay the entire transaction history, since the beginning of time. (This is why your bank statement lists both a transaction history *and* an account balance!) In such situations, it may actually make sense to invest in the required infrastructure to pull out the data and store it in a more manageable fashion. Chances are plenty of uses for the sanitized data will appear over time. (Build it, and they will come.)

16.3.2 Data Consistency

Problems of data consistency (as opposed to data availability) occur in every company of sufficient size and are simply an expression of the complexity of

the underlying business. Here are some typical examples that I have encountered:

- Different parts of the company use different definitions for the same metric. Operations, for example, may consider an order to be completed when it has left the warehouse (more precisely: when it has passed the last scan station), whereas the finance department will consider an order to be complete when the payment for it has been received.
- Reporting time frames may not be aligned with operational process flows. A seemingly simple question such as: "How many orders did we complete yesterday?" can quickly become complicated, depending on whose definition of "yesterday" we are going to use. For example, in a warehouse, we may only be able to obtain a total for the number of orders completed per shift — but then how do we account for the shift that stretches from 10pm at night to 6am the next morning? How do we deal with timezones? Simply stating that "yesterday" refers to the local time at the corporate headquarters sounds simple, but is probably not going to be practical, since all the facilities will naturally be doing their bookkeeping and reporting according to their local time.
- Time flows backwards. How does one account for an order that was later returned? If we want to recognize revenue in the quarter in which the order was completed, and it is later returned, we are going to have a problem. We can still report on the revenue accurately — but not in a timely manner. (In other words, final quarterly revenue reports can not be produced until the time allowed to return an item has elapsed. Keep in mind that this may be a *long* time in the case of extended warranties or similar arrangements.)

Additional difficulties will arise if information has been lost, for instance because the revision history of a contract has not been kept (recall our earlier discussion). You can probably think of further scenarios in which problems of data or metric inconsistency occur.

The answer to this set of problems is not technical, but administrative or political. Basically it comes down to agreeing on a common definition of all metrics. An even more drastic recommendation to deal with conflicting metrics is to declare one source of data as the "normative" one — this does not make the data any more accurate, but it can help to stop fruitless efforts to reconcile different sources at all cost. At least that's the theory. Unfortunately, if the manager of an off-site facility can expect to have his feet held to the fire by the CEO over why the facility missed the daily goal of 2 million produced units by seven units last Friday, he will look for ways to pass the blame. And pointing to inconsistencies in the reports is an easy way out. (In my experience, one of the big drawbacks of all metrics programs is the amount of work generated to reconcile minute inconsistencies between different versions of the same data. The cost in terms of wasted developer-time and -frustration can be stunning.)

As practical advice, I would recommend to strive as much as possible for clear definitions of all metrics, so that at least we know what it is we are talking about. Furthermore, wherever possible, try to make those metrics normative which are *practical* to gather, rather than “correct” from a theoretical point of view (local versus global time coordinates come to mind). If necessary, apply conversion factors behind the scenes, but try to have humans deal with quantities that are meaningful and familiar to them as much as possible.

16.4 Workshop: Berkeley DB and SQLite

For analysis purposes, the most suitable data format is usually the flat file: most of the time, we will want all (or almost all) of the records in the data set for our analysis. It therefore makes more sense to slurp in the whole file, possibly filter out the unneeded records, and process the rest, rather than to do an indexed lookup of only the records that we want.

Common as this scenario is, it does not always apply, and in particular when it comes to reporting, it can be highly desirable to have access to data storage solution that supports structured data, indexed lookup, and even the ability to merge and aggregate data. In other words, a database.

The problem is that most databases are *expensive* — and I don’t (just) mean in terms of money. They require their own process (or processes!), they require care and feeding, they require network access (so that people and processes can actually get to them). They must be designed, installed, and provisioned, and very often, they require architectural approval before anything else. (At least in one shop I worked at, the latter point was such an ordeal that it made anything requiring changes to the database environment basically impossible; one simply had to invent solutions not requiring them.) In short, most databases are expensive: both technically and politically.

Luckily, other people have recognized this, and developed database solutions that are *cheap*: so-called *embedded databases*. Their distinguishing feature is that they do not run in a separate process. Instead, they store their data in a regular file, which is accessed through a library linked into the application. This means that most of the overhead for provisioning and administration goes away — moreover, we can replicate the entire database by simply copying the data file! (Occasionally very useful to “deploy” databases.)

Let’s take a look at the two most outstanding examples of (open-source) embedded databases: the Berkeley DB, which is a key/value hashmap stored on disk, and SQLite, which is a complete relational database “in a box”. Both of them have bindings to basically any programming language — here, we demonstrate them from Python. (Both ship with the Python Standard Library, and therefore should already be available wherever Python is.)

16.4.1 Berkeley DB

The Berkeley DB is a key/value hashmap (a “dictionary”), persisted to disk. (The notion of a persistent key/value database originated on Unix; the first implementation being the Unix `dbm` facility. Various reimplementations (`ndbm`, `gdbm`, and so on) exist. The original “Berkeley DB” was just one specific implementation, which added some additional capabilities (mostly multi-user concurrency support). It was developed and distributed by a commercial company (Sleepycat), which was acquired by Oracle in 2008. However, the name “Berkeley DB” is being used for this kind of key/value database in general.)

Moreover, through the magic of operator overloading, a Berkeley DB also *looks* like a dictionary to the programmer¹ (with the provision that keys and values *must* be strings):

```
import dbm

db = dbm.open( "data.db", 'c' )

db[ 'abc' ] = "123"
db[ 'xyz' ] = "Hello, World!"
db[ '42' ] = "42"

print db[ 'abc' ]

del db[ 'xyz' ]

for k in db.keys():
    print db[k]

db.close()
```

That’s all there is to it. In particular, notice that the overhead (“boilerplate”) required is precisely zero. You can’t do much better than that.

I used to be a great fan of the Berkeley DB, but over time I have become more aware of their limitations. Berkeley DBs store single-key/single-value pairs — period. If that’s what you want to do, they indeed are great. But the moment that’s not *exactly* what you want to do, they simply are the wrong solution. Here are a few things you *cannot* do with a Berkeley DB:

- Range searches: `3 < x < 17`
- Regular expression searches: `x like 'Hello%'`
- Aggregation: `count(*)`
- Duplicate keys

¹In Perl, you use a “tied hash” to the same effect.

- Result sets consisting of multiple records and iteration over result sets
- Structured data values
- Joins

In fairness, you can achieve some of these features, but you have to build them yourself (for example, provide your own serialization and deserialization to support structured data values), or be willing to lose almost all of the benefit provided by the Berkeley DB (you can have range or regular expression searches, as long as you are willing to suck in *all* the keys and process them sequentially in a loop).

Another area in which Berkeley DBs are weak concerns administrative tasks. There are no standard tools to browse and (possibly) edit entries, with the consequence that you have to write your own tools to do so. (Not hard, but annoying.) Furthermore, Berkeley DBs don't maintain administrative information about themselves (such as the number of records, most recent access times, all that). The solution of course (which I have seen implemented in about every project using a Berkeley DB) is to maintain this information explicitly, and to store it in the DB under a special, synthetic key. All of this is easy to do, but it does add back some of the "boilerplate" code, that we hoped to avoid by using a Berkeley DB in the first place.

16.4.2 SQLite

In contrast to the Berkeley DB, SQLite is a full-fledged relational database: with tables, keys, joins, and WHERE clauses. You talk to it in the familiar fashion through SQL. (In Python, you can use the DB-API 2.0, or one of the higher-level frameworks built on top of it.)

SQLite supports almost all features found in standard SQL, with very few exceptions. The price you have to pay is that you have to design and define a schema. SQLite therefore has a bit more overhead than a Berkeley DB: both in terms of the upfront design that is required, and in regards to the need for a certain amount of boilerplate code.

A simple example exercising many features of SQLite is shown in listing 16.1, and should pose few (if any) surprises, but it does demonstrate some interesting features of SQLite.

Initially, we "connect" to the database — if it does not yet exist, it will be created. We specify autocommit mode, so that each statement is executed immediately. (SQLite also does support concurrency control through explicit transaction.)

Then we create two tables. Two things should be noticed: the first column is specified as a primary key (meaning that it will be indexed automatically), with an autoincrement feature. All other columns do not have a data type associated with them, because basically all values are stored in SQLite as strings. (It is also possible to declare certain type conversions that should be applied to the values, either in the database or in the Python interface.)

We then insert two orders and some associated lineitems. In doing so, we make use of a convenience feature provided by the `sqlite3` module: the last value of an autoincremented primary key is available through the `lastrowid` attribute (data member) of the current cursor object.

Finally, we run two queries — the first one demonstrates a join and the use of SQL wildcards, the second uses an aggregate function and also sorts the result set. As you can see, basically everything you know about relational databases carries over directly to SQLite!

SQLite supports some additional features that I have not mentioned, such as an “in-memory” mode, in which the entire database is kept entirely in memory: this can be very useful if you want to use SQLite as a part of a performance critical application. Also part of SQLite is the command-line utility `sqlite3`, which allows you to examine a database file and run ad-hoc queries against it.

I have found SQLite to be very useful — basically everything you expect from a relational database, but without most of the pain. I recommend it highly.

```

import sqlite3

# Connect and obtain a cursor
conn = sqlite3.connect( 'data.db' )
conn.isolation_level = None           # use autocommit!
c = conn.cursor()

# Create tables
c.execute( """CREATE TABLE orders
              ( id INTEGER PRIMARY KEY AUTOINCREMENT,
                customer )""")
c.execute( """CREATE TABLE lineitems
              ( id INTEGER PRIMARY KEY AUTOINCREMENT,
                orderid, description, quantity )""")

# Insert values
c.execute( "INSERT INTO orders ( customer ) VALUES ( 'Joe Blo' )")
id = str( c.lastrowid )
c.execute( """INSERT INTO lineitems ( orderid, description, quantity )
              VALUES ( ?, 'Widget 1', '2' )""", ( id ) )
c.execute( """INSERT INTO lineitems ( orderid, description, quantity )
              VALUES ( ?, 'Fidget 2', '1' )""", ( id ) )
c.execute( """INSERT INTO lineitems ( orderid, description, quantity )
              VALUES ( ?, 'Part 17', '5' )""", ( id ) )

c.execute( "INSERT INTO orders ( customer ) VALUES ( 'Jane Doe' )")
id = str( c.lastrowid )
c.execute( """INSERT INTO lineitems ( orderid, description, quantity )
              VALUES ( ?, 'Fidget 2', '3' )""", ( id ) )
c.execute( """INSERT INTO lineitems ( orderid, description, quantity )
              VALUES ( ?, 'Part 9', '2' )""", ( id ) )

# Query
c.execute( """SELECT li.description FROM orders o, lineitems li
              WHERE o.id = li.orderid AND o.customer LIKE '%Blo'""")
for r in c.fetchall():
    print r[0]

c.execute( """SELECT orderid, sum(quantity) FROM lineitems
              GROUP BY orderid ORDER BY orderid desc""")
for r in c.fetchall():
    print "OrderID: ", r[0], "\tItems: ", r[1]

# Disconnect
conn.close()

```

Example 16.1: TBD

Chapter 17

Financial Calculations and Modeling

I recently got a notice from a magazine I subscribe to, reminding me that my subscription was running out. It's a relatively expensive weekly magazine, and they offered me three different plans to renew my subscription: one year (52 issues) at \$130, two years at \$220, or three years at \$275. Table 17.1 summarizes these options and also shows the respective cost per issue.

Subscription	Total Price	Price per Issue
single issue	n/a	6.00
1 year	130	2.50
2 years	220	2.12
3 years	275	1.76

Table 17.1: Pricing plans for a magazine subscription

Assuming that I want to continue the subscription, which of these three options makes the most sense? From table 17.1 we can see that each issue of the magazine becomes cheaper as I commit myself to a longer subscription period, but — is this a good deal? In fact, what does it mean for a proposal like this to be a “good deal”? Somehow, stomping up nearly three hundred dollars right now seems like a stretch, even if I remind myself that it saves me more than half of the price of a single issue.

This little story demonstrates the central topic of this chapter: the *time-value of money*: a hundred dollars today are worth more than a hundred dollars a year from now. In this chapter, I will introduce some standard concepts and calculational tools that are required whenever we need to make a choice between different investment decisions, be it in our own personal finances, or when evaluating the business cases for different corporate projects.

I find the material in this chapter fascinating — not because it is rocket sci-

ence (it isn't), but because it is so fundamental to the way the economy works. Yet, very few people, and in particular very few tech people have any understanding of it. (I certainly didn't.) This is a real shame, not just because of the topic's obvious importance, but also because it is actually not all that mystical. A little familiarity with the basic concepts goes a long way to removing most of the confusion (and, let's face it, the intimidation) that many (most?) of us experience when reading the Wall Street pages.

More important in the context of this book is that a lot of data analysis is done specifically to evaluate different business proposals and to support decisions among them. And to be able to give effective, appropriate advice, you want to understand the concepts and the terminology of this particular problem domain.

17.1 The Time-Value of Money

Coming back to the subscription question: the essential insight is that instead of paying for the second and third year of the subscription *now*, I could instead invest that money, reap the investment benefit, and pay for the subsequent years of the subscription later. In other words, the discount that the magazine offers me has to be *greater* than the investment income I could expect if I were to invest the sum instead.

It is this ability of gaining an investment benefit which makes having money *now* more valuable than having the same amount of money *later*. Note well that it has nothing to do with the concept of *inflation*, which is the process by which a certain amount of money tends to buy a lesser amount of goods as time passes. For our purposes, inflation is an external influence, that we don't have control over. By contrast, investment and purchasing decisions (such as the magazine subscription problem with which I opened this chapter) are under our control, and time-value of money calculations can help us to make the best possible decisions in this regard.

17.1.1 A single payment: Future and Present Value

Things are easiest when there is only a single payment involved. Imagine we are given the following choice: receive \$1000 today, or receive \$1050 a year from now. Which one should we choose?

Well, that depends on what we could do with \$1000 right now. For this kind of analysis, it is customary to assume that we would put the money in a "totally safe form of investment" and use the returns generated in this way as a benchmark for comparison.¹ Now we compare the alternatives against the interest that would be generated by this "safe" investment. For example, let's assume that the current interest rate that we could gain on a "safe" investment

¹This used to mean investing in US Treasury Bonds or the equivalent, but at the time of this writing, even these are no longer considered sacrosanct. But that's leaving the scope of this discussion!

is 5 percent annually. So if we invest \$1000 for a full year, we will at the end of the year receive back our principal (\$1000), and in addition the accrued interest ($0.05 \cdot \$1000 = \50), for a total of \$1050.

In this example, both options lead to the same amount of money after one year; we say, they are *equivalent*. In other words, receiving \$1000 now is *equivalent* to receiving \$1050 a year from now, *given* that the current interest rate on a safe form of investment is 5 percent annually. Equivalence always refers to a specific time frame and interest rate.

Clearly, any amount of money that we currently have will have a *future value* (or *future worth*) at any point in the future; and a payment we are going to receive at some point in the future has a *present value* (or *present worth*) now. Both are dependent on the interest rate that we could achieve by investing in a safe alternative investment instead. The present or future values must be equivalent at equal times.

There is a little bit of math behind this, which is really not complicated, but often a little messy. To calculate the future value V_f of some base amount M (the *principal*) after a single time period, during which the amount earns p percent of interest, is:

$$\begin{aligned} V_f &= M + \frac{p}{100} M \\ &= \left(1 + \frac{p}{100}\right) M \end{aligned}$$

The first term on the right-hand side expresses that we get our principal back, the second term is the amount of interest we will receive in addition. Here and in the following, I am also showing the denominator 100 explicitly that is used to translate a statement such as “ p percent” into the equivalent numerical factor $p/100$.

In contrast, if we want to know how much a certain amount of money in the future is worth today, we have to *discount* that amount to its present value. To find the present value, we work the equation above backwards: the present value V_p is unknown, but we do know the amount of money M we will have at some point in the future, hence the equation above becomes:

$$M = \left(1 + \frac{p}{100}\right) V_p$$

Or, after solving for V_p :

$$V_p = \frac{M}{1 + \frac{p}{100}}$$

Note how we find the future or present value by multiplying the base amount with an appropriate *equivalencing factor*, namely the future-worth factor $(1 + p/100)$ and the present-worth factor $1/(1 + p/100)$. And because we mostly discount a future payment to the present value, the percentage rate p used in these formulas is usually referred to as the *discount rate*.

The example we just looked at was the simplest possible, because there was only a single payment involved — either at the beginning or at the end

of the period under consideration. Next, we look at scenarios where there are multiple payments throughout the period of interest.

17.1.2 Multiple payments: compounding

Things get a bit more complicated if there is not just a single payment involved as in the example above, but a series of payments over time. Each of these payments must be discounted by the appropriate, time-dependent factor, which leads to *cashflows analysis*. And payments made or received may alter the base amount that we operate with — this leads to the concept of *compounding*.

Let's consider compounding first, since it is so fundamental. Again, the idea is very simple: if we put a sum of money into an interest-bearing investment, and then *re-invest* the generated interest, we will start receiving interest on the interest itself. In other words, we will start receiving *compound interest*.

Here is how it works: we start with principal M and invest it at interest rate p . After one year, we have:

$$V(1) = \left(1 + \frac{p}{100}\right) M$$

In the second year, we will receive interest on the combined sum of the principal and the interest from the first year:

$$\begin{aligned} V(2) &= \left(1 + \frac{p}{100}\right) V(1) \\ &= \left(1 + \frac{p}{100}\right)^2 M \end{aligned}$$

and so on. After n years, we will have:

$$V(n) = \left(1 + \frac{p}{100}\right)^n M$$

These equations tell us the future worth of our investment at any point in time. It works the other way around, too: we can determine the present value of a payment we expect to receive n years from now by working the equations backwards in much the same way we did earlier for a single payment and find:

$$V(\text{present}) = \frac{M}{\left(1 + \frac{p}{100}\right)^n}$$

As we can see from these equations, if we continue re-investing our earnings, the total amount of money grows exponentially with time (that is, like a^t for some constant a) — in other words, *fast*. The growth law that applies to compound interest is the same that describes the growth of bacteria cultures or similar systems, where at each time step new members are added to the population and start producing offspring themselves. In such systems, not only does the population grow, but the rate at which it grows is constantly increasing as well.

On the other hand, if you take out a loan without making payments, and have the lender add the accruing interest back onto your principal, not only do you get deeper in debt every month, but you are also getting *faster* into debt as time goes by.

17.1.3 Calculational Tricks with Compounding

I wanted to point out a simple trick that is very convenient when making approximate calculations of future and present worth. The single-payment formula for future worth: $V = (1 + p/100)M$, is simple and intuitive: the principal *plus* the interest after one period. By contrast, the corresponding formula for present worth: $V = \frac{M}{1+p/100}$, seems to make less intuitive sense and is harder to work with (how much is \$1000 divided by 1.05?). But this is again one of those situations where the guesstimation techniques we learned in chapter 7 can be brought to bear. We can approximate the discounting factor as follows:

$$\frac{1}{1 + \frac{p}{100}} \approx 1 - \frac{p}{100} + \left(\frac{p}{100}\right)^2 \mp \dots$$

Since p is typically small (single digits), $p/100$ is very small and we can terminate the expansion after the first term. Using this approximation, the discounting equation for the present worth becomes $V = (1 - p/100)M$, which has an intuitive interpretation: the present value is the value of the future value, less the interest that we will have received by then.

We can use similar formulas even in the case of compounding, because:

$$\begin{aligned} \left(1 + \frac{p}{100}\right)^n &\approx 1 + n \frac{p}{100} + \dots \\ \left(1 + \frac{p}{100}\right)^{-n} &\approx 1 - n \frac{p}{100} + \dots \end{aligned}$$

However, keep in mind that the overall perturbation must be small for the approximation to be valid. In particular as the number of years n gets larger, the perturbation term $np/100$ may no longer be small. Still, even for five percent over five years, the approximation gives $1 \pm 25/100 = 1.25$ or 0.75 respectively. Compare this with the exact values of 1.28 and 0.79. On the other hand, for ten percent over ten years, the approximation starts to break down, yielding 2 and 0 respectively, compared to the exact values of 2.59 and 0.39.

Similar logic is behind “Einstein’s Rule of 72” that you may have heard of. This rule of thumb, states that if you divide 72 by the applicable interest rate, you obtain the number of years that it would take for your investment to double. So, if you earn 7 percent interest, your money will double in ten years, but if you only earn 3.5 percent, it will take 20 years to get there.

What’s the basis for this rule? By now, you can probably figure it out yourself, but here is the solution in a nutshell: for your investment to double, the compounding factor must equal 2. Therefore we need to solve $(1 + p/100)^n =$

2 for n . Applying logarithms on both sides we find $n = \log(2) / \log(1 + p/100)$. In a second step we expand the logarithm in the denominator (remember that $p/100$ is a small perturbation!) and end up with $n = \log(2)100/p = 69/p$ because the value of $\log(2)$ is approximately 0.69. The number 69 is awkward to work with, so it is usually replaced by the number 72, which has the advantage of being evenly divisible by 2, 3, 4, 6, 8, and 9 (you can replace 72 by 70 for interest rates of 5 or 7 percent).

Here is another calculational tool that you may find useful. Strictly speaking, an expression such as x^n is only defined for integer n . For general exponents, the power function is defined as $x^n = \exp(n \log x)$. We can use this when calculating compounding factors as follows:

$$\begin{aligned} \left(1 + \frac{p}{100}\right)^n &= e^{n \log\left(1 + \frac{p}{100}\right)} \\ &\approx e^{n \frac{p}{100}} \end{aligned}$$

where in the second step we have expanded the logarithm again and truncated the expansion after the first term. This form of the compounding factor is often convenient (for example, it allows arbitrary values for the time period n , not just full years). It becomes exact in the limit of continuous compounding (see below).

Interest rates are conventionally quoted “per year”: “5 percent annually” or something like this. But payments may occur more frequently than that. Savings accounts, for example, pay out any accrued interest on a monthly basis. That means that (as long as we don’t withdraw anything) the amount of money that earns us interest grows every month, we say: it is *compounded monthly*. (This is in contrast to other investments, which only pay out interest or dividends on a quarterly or even annual basis.) To take advantage of the additional compounding, it is of course in our interest (pun intended) to receive payments as early as possible.

This monthly compounding is the reason for the difference between the *nominal* interest rate and the annual *yield* that you will find stated on your bank’s website: the nominal interest rate is the rate p that is used to determine the amount of interest that is paid out to you each month. The yield tells you by how much your money will grow over the course of the year, once the monthly compounding has been factored in. With our knowledge we can now calculate the yield from the nominal rate:

$$\left(1 + \frac{p_{\text{yield}}}{100}\right) = \left(1 + \frac{\frac{p_{\text{nominal}}}{12}}{100}\right)^{12}$$

One more bit of terminology: the interest rate $p/12$ that is used to determine the value of the monthly payout is known as the *effective* interest rate.

Of course, other payment periods are possible. Many mutual funds pay out quarterly. By contrast, many credit cards compound daily. Theoretically,

we can imagine payments to be made constantly (but at an appropriately reduced effective interest rate): this is the case of *continuous compounding* mentioned earlier. In this case, the compounding factor is given by the exponential function. (Mathematically, you replace the 12 in the formula above by n and then let n go to infinity and use the identity $\lim_{n \rightarrow \infty} (1 + x/n)^n = \exp(x)$ — see appendix B.)

17.1.4 The whole picture: cashflow analysis and net present value

We now have all the tools at our disposal to evaluate any investment decision, no matter how complicated. Imagine we are running a manufacturing plant (or, maybe, an operation like Amazon's where books and other goods are put into boxes and mailed to customers — that's how I learned about all these things). We may consider buying some piece of automated equipment for some part of the process (for example, a sorting machine that sorts boxes according to their destination). Alternatively, we can have people do the same job manually. Which of these two alternatives is better, from an economic point of view?

The manual solution has a simple structure: we simply have to pay out the required wages every year. If we decide to buy the machine, we have to pay the purchasing price right now (this is also known as the *first cost*), and then have to pay a small maintenance fee each year. For the sake of the argument, let's also assume that we expect to use the machine for ten years and then sell it on for scrap value.

In economics text you will often find the sequence of payments visualized through so called *cashflow diagrams* (see figure 17.1). Time progresses left to right, inflows are indicated through upward pointing arrows, outflows through downward pointing arrows.

To decide between different alternatives, we now proceed as follows:

1. Determine all individual net cashflows (*net cashflows*, because we offset annual costs versus revenues).
2. Discount each cashflow to its present value.
3. Add up all contributions.

The quantity obtained in the last step is known either as the *net present value* or the *discounted net cashflow*. It is the total value of all cashflows, each properly discounted to its present value. In other words, our financial situation will be the same, whether we execute the entire series of cashflows *or* receive the net present value today. Because the net present value it contains all inflows and outflows, properly discounted to the present value, it is a comprehensive single measure which can be used to compare the financial outcomes of different investment strategies.

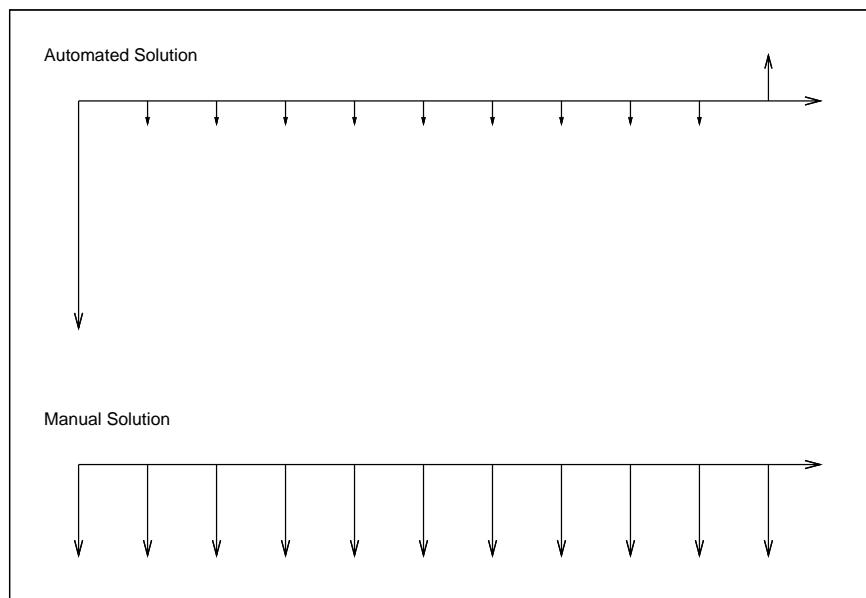


Figure 17.1: Examples of cashflow diagrams

We can write down an expression for the net present value as follows:

$$\text{NPV} = \sum_i \frac{c(i)}{(1 + p/100)^i}$$

where $c(i)$ is the net cashflow at payment period i , and $1/(1 + p/100)^i$ is the associated discounting factor.

There is one more concept that is interesting in this context: what should we use for the discount rate p in the second step above? Rather than supplying a value, we can instead ask how much interest we would have to receive elsewhere (on a “safe” investment) to obtain the same (or higher) payoff than we expect from the planned project. Let’s consider an example: assume that we are evaluating a project that would require us to purchase some piece of equipment at the beginning, but that would result in a series of positive cashflows over the next so many years. Is this a “good” investment? It is, if its net present value is positive! (That’s pretty much the definition of “net present value”.) But the net present value depends on the discount rate p , so we need to find that value of p for which the NPV first becomes zero: if we can get a higher interest rate elsewhere, then the project does not make sense and we should instead take our money to the bank. But if the bank would pay us less than the *rate of return* that we just calculated, then the project is financially the better option.

The reason that the net present value is so important in making investment decisions is that it provides us with a single number, which summarizes the financial results of any planned project. It gives us an objective (financial) quantity to decide among different investment alternatives.

Up to a point, that is. The process described above is only as good as its inputs. In particular, we have assumed that we know all inputs perfectly — possibly for years out in the future. Of course we don’t, and we better accommodate for that uncertainty somehow. That will be the topic of the next section.

There is another, more subtle problem when evaluating different options solely based on net present value: different investment alternatives may have non-financial benefits or drawbacks, which are not captured by the net present value. For example, using manual labor may lead to greater flexibility: if business grows more strongly than expected, the company can hire additional workers, and conversely, if business slows down, we can reduce the number of workers. In contrast, any piece of equipment has a maximum capacity, which may be a limiting factor if business grows more strongly than expected. The distinction that comes up here is the difference between fixed and variable cost, and we will come back to it towards the end of the chapter.

17.2 Uncertainty in Planning and Opportunity Costs

Now we are ready to come back to the magazine subscription problem from the beginning of this chapter. Let’s just consider two alternatives: paying the

entire amount up front or making two single-year payments. The net present value for the second option is: $(1 + 1/(1 + p/100)) C_{1\text{yr}}$, where we have left the discount rate p undetermined for the moment. We can now ask: what interest rate would we have to get elsewhere, to make the the second option worthwhile? In other words, we are asking what discount rate we would have to apply to make the net present value of the multiple payment option equal to the cost of the single payment plan:

$$\left(1 + \frac{1}{1 + \frac{p}{100}}\right) C_{1\text{yr}} = C_{2\text{yr}}$$

This equation can be solved for p . The result is $p = 30\%$! In other words, the two-year subscription is so much cheaper that we would have to find an investment yielding 30 percent per year before it would be worthwhile to pay the subscription year-by-year and invest the saved money elsewhere. No investment (and certainly no “save” investment) yields anywhere close to that much. Clearly something is amiss. (Exercise for the reader: find the net present value for the three year plan and verify that it leads to the same value for p !)

17.2.1 Using Expectation Values to Account for Uncertainty

The two- and three-year plans carry a hidden cost for us: once we signed up, we can no longer freely decide over our money — we commit ourselves for the long-haul. In contrast, if we pay on a yearly basis, we can re-evaluate whether we want to continue the subscription every year. We will have to pay for this freedom through a higher subscription fee. However, we will probably not find it easy to determine the exact dollar-value that this freedom is worth to us.

From the magazine’s perspective, the situation is simpler. They can simply ask how much money they expect to make from an individual subscriber under either option. If I sign up for the two-year subscription, they make $C_{2\text{yr}}$ with certainty, whereas if I sign up for the one-year subscription, they make $C_{1\text{yr}}$ with certainty now, and another $C_{1\text{yr}}$ later — if I will continue my subscription in a year, that is! So, in this case, the amount of money they expect to make on me is $C_{1\text{yr}} + \gamma C_{1\text{yr}}$, where γ is the probability that I will continue my subscription. From the magazine’s perspective both options must be equally favorable (otherwise they would adjust the price of the two-year subscription to make them equal), so we can equate the expected revenues and solve for γ . The result comes out to about $\gamma = 0.7$ — in other words, the magazine expects (based on past experience, and so on) that about 70 percent of its current subscribers will renew their subscription once it has expired. For three years, the equation becomes: $(1 + \gamma + \gamma^2)C_{1\text{yr}} = C_{3\text{yr}}$, because to sign up for three years a subscriber must decide *twice* to renew the subscription. If you work through the algebra, you will find that γ again comes out to about $\gamma = 0.7$, providing a nice consistency check.

There are two take-aways here that I would like to emphasize: when making economic decisions that are subject to uncertainty, you may want to take this uncertainty into account by replacing the absolute cash-flows with their expected values. Very often, a very simple probability model for the likely payout is sufficient. In the magazine example, I used just two outcomes: the subscriber renews with probability $\gamma = 0.7$ and value $C_{1\text{yr}}$, or the subscriber does not renew with probability $\gamma = 0.3$ and value 0, so that the expected value is: $0.3 \cdot 0 + 0.7 \cdot C_{1\text{yr}}$. If your situation warrants it, and you can specify the probability distribution for various payout alternatives in more detail, you can calculate the expected value accordingly. (See chapter 8 and chapter 9 for more information on how to build models to support this kind of conclusion.)

Working with expectation values is convenient, because once you have determined the expected value of the payout, you don't have to worry about the probabilities for the various outcomes anymore: they have been entirely absorbed into the expectation values. What you lose is insight into the probable spread of outcomes. For a quick order-of-magnitude check that's acceptable, but for a more serious study it should be included. There are two ways to do this: repeat your calculation multiple times using different values (low, medium, high) for the expected payouts at every step to get a sense for the range of possible outcomes. (If there are many different options, you may want to do this through simulation — see chapter 12.) Alternatively, you might want to evaluate both the expectation value and the spread directly from the probability distribution to obtain a range for each estimated value: $\mu \pm \sigma$. Now you can use this sum in your calculations, treating σ as a small perturbation and evaluate the effect of this perturbation on your model (see chapter 7).

17.2.2 Opportunity Costs

The second point that I would like to emphasize is the concept of *opportunity cost*. Opportunity costs arise if we miss out on some income (the “opportunity”) because we are not in the position to take advantage of it. Opportunity costs formalizes the notion that resources are finite and that if we apply them to one purpose, they are not available for other uses. In particular, if we commit resources to some project, we want this project to generate greater benefit than the opportunity cost that arise because those resources are now no longer available for other engagements.

I find it easiest to think about opportunity cost in the context of certain business situations. For instance, imagine a company takes on a project which pays \$15,000. While this contract is going on, somebody else offers this company another project that would pay \$20,000. Assuming that the company cannot break its initial engagement, it is now incurring an opportunity cost of \$5,000.

I find the *concept* of opportunity cost useful as a way to put a price on alternatives, in particular in situations where no money changes hands. In textbooks, this is often demonstrated by ways of the student who takes a trip around the world instead of working at a summer job. This student does not only have to pay the actual expenses for the trip, but also incurs an opportu-

nity cost equal to the amount of foregone wages. In this example, the concept of opportunity cost allows us to account for these foregone wages, which would otherwise be difficult, since they do not show up on any account statement because they have never actually been paid.

On the other hand, I often find opportunity cost a somewhat shadowy concept, because it totally hinges on a competing opportunity actually arising! Imagine you try to decide between two opportunities: an offer for a project that would pay \$15,000 and the prospect of a project paying \$20,000. If you take the first job and then the second opportunity comes through as well, you are incurring an opportunity cost of \$5,000. But if the second project falls through, your opportunity cost just dropped to zero! (The rational way to make this decision would be to calculate the total revenue expected from each prospect, but *weighted by the probability* that the contract will actually be signed. Which brings us back to operations with expected payout, as we discussed just a moment ago.)

To be clear: the concept of opportunity cost per se has nothing to do with uncertainty in planning. It is merely a way to evaluate the relative costs of competing opportunities. However, very often when evaluating competing deals, we must decide between plans which have a different likelihood to come to fruition, and therefore opportunity cost and planning for uncertainty often arise together.

17.3 Costs Concepts and Depreciation

The methods of the previous sections might suggest that the net present value is all there is to financial considerations. This is not so — other factors may influence our decision. Some are entirely outside the financial realm (for instance, ethical or strategic considerations), while others yet may very well have direct business implications, but are not sufficiently captured by the quantities we have discussed so far.

For example, let's go back to the situation we considered earlier, where we considered the choice between two alternatives: buying a sorting machine, or having the same work be done through manual labor. Once we take all arising costs into account and discount them properly to the present value, it would seem that we have accounted for all financial implications. But that would be wrong: the solution employing manual labor is more flexible, for instance. If the pace of the business varies over the course of the year, then we need to buy a sorting machine that is large enough to handle the busiest season, while being under-utilized during the rest of the year. If we rely on manual labor, then we can more flexibly scale capacity up through temporary labor or overtime. Similarly, we can respond to unexpectedly strong (or weak!) growth of the overall business more flexibly, again by adjusting the number of workers. (This in turn, may have further consequences, in regards to labor relations, for example.) In short, we need to look at the costs and the way they arise in more detail.

To understand the cost structure of a business or an operation better, it is often useful to discuss it in terms of three pairs of complementary concepts:

- Direct versus Indirect Cost
- Fixed versus Variable Cost
- Capital Expenditure versus Operating Cost

And for good measure, I'll also throw in the idea of *depreciation*, although it is not a cost in the strict sense of the word.

17.3.1 Direct and Indirect Cost

Labor and materials that are applied in the creation of the *product* (that is, in the creation of something the company will *sell*) are considered direct labor or direct materials cost. Indirect costs, on the other hand, arise out of activities that the company undertakes to maintain *itself*: management, maintenance, and administrative tasks (payroll and accounting), but also training, for example. Another term for such indirect costs is *overhead*.

I should point out that this is a slightly different definition of direct and indirect costs than the one you will find in the literature. Most textbooks define direct cost as the cost that is "easily attributable" to the production process, whereas indirect cost is "not easily attributable". This definition makes it seem as if the distinction between direct and indirect cost is mostly one of convenience. Furthermore, the textbook definition provides no reason why for example maintenance and repair activities are usually considered indirect costs: surely, we can keep track which machine needed how much repair, and therefore assign the associated cost to the product made on that specific machine? On the other hand, using the definition I gave earlier, it is clear that maintenance should be considered an indirect cost, because it is an activity that the company undertakes to keep *itself* in good order instead of generating value for the customer.

Earlier, I used the term "product" for whatever the company is selling. For manufacturing or retail industries this is a straightforward concept, but for a service industry the "product" may be intangible. Nevertheless, in probably all businesses we can introduce the concept of a single produced unit or *unit of production*. In manufacturing and retail there are actual "units", but in other industries the notion of a produced unit is a bit more artificial: in service industries for example, one often uses "billable hours" as a measure of production. Other industries have specialized conventions — the airline industry uses "passenger miles", for example.

The unit is an important concept because it is the basis for the most common measure of productivity, namely the unit cost or *cost per unit* (CPU). The cost per unit is obtained by dividing the total (dollar) amount spent during a time period (per month, for instance) by the total number of units produced during that time. If we include not only the direct cost, but also the indirect

cost in this calculation, we obtain what is called the *loaded* or *burdened* cost per unit.

We can go further and break out the various contributions to the unit cost. For example, if there are multiple production steps, we can determine how much each step contributes to the total cost. We can also study how much indirect costs contribute to the overall cost, and how material costs relate to labor. Understanding the different contributions to the total cost per unit is often a very worthwhile exercise, because it points directly to where the money is spent. And appearances can be deceiving. I have seen situations where literally hundreds of people were required for a certain processing step, while next door a single person was all that was needed to oversee a comparable, but highly automated process. Yet, once you calculated the cost per unit, it all looked very different: because the number of units that went through the automated process was low, the total cost per unit was actually higher than for the manual process. And because so many units were processed manually, the labor cost *per unit* for the manually processed units turned out to be very small.

In general, it is desirable to have low overhead relative to the direct cost: a business should spend relatively less time and money managing itself, compared to generating value for the customer. In this way, the ratio of direct to indirect cost can be an interesting indicator for “top-heavy” organizations that mostly seem to manage themselves. On the other hand, over-eager attempts to improve the direct-cost ratio can lead to pretty unsanitary manipulations. For example, one company I know considers software engineers *direct* labor, while any form of management (team leads, dev managers, project managers) is considered *indirect*. The natural consequence is that such management responsibilities are being pushed onto developers — which of course does not make the tasks go away, they just become invisible. (It also leads to an inefficient use of a scarce resource: developers are always in short supply — and they are expensive.) In short: beware the danger of perverted incentives!

17.3.2 Fixed and Variable Cost

Compared to the previous section (on direct and indirect cost), the distinction between fixed and variable cost is clearer. The *variable* costs are those costs that change in response to changing demand, while *fixed* costs don’t. For a car manufacturer, the cost of steel is a variable cost: if fewer cars are being built, less steel is consumed. Whether labor costs are fixed or variable depends on the type of labor and the employment contracts. But the capital cost for the machines in the production line are a fixed cost, because they have to be paid, whether they machines are idle or busy.

It is important to distinguish between direct and variable costs. Although direct costs are more likely to be variable (and overhead in general is fixed), these are unrelated concepts and one can easily come up with examples of fixed, yet direct costs. For example for a consultancy with salaried employees, their staff of consultants constitutes *direct* cost, yet it is also a *fixed* cost, since

the consultants expect their wages regardless of whether the consultancy has projects for them or not. (We'll see another example in a moment.)

In general, having high fixed costs relative to variable ones makes a business or industry less flexible and more susceptible to downturns. An extreme example is the airline industry, with a cost structure that is almost exclusively fixed (pretty much the only variable cost is the price of the inflight meal), yet with a demand pattern that is subject to extreme cyclical swings.

(The numbers are interesting. Let's do a calculation in the spirit of chapter 7. A modern jet airplane costs about \$100M new and has a useful life of about 10 years. The cost attributable to a single 10 hour trans-atlantic flight (the depreciation — see below) comes to about \$30k (or half that, if the plane is turned around immediately, completing a full round-trip within 24 hours). Fuel consumption is about 6 gallons/mile — for a 4000mile (6000km) flight between New York and Frankfurt (Germany), and assuming \$2 per gallon, this comes to \$50k for fuel. Let's say there are 10 members of the cabin crew at \$50k yearly salary, and two people in the cockpit at \$150k each. Double that number for miscellaneous benefits and we end up with about \$2M yearly labor cost, or \$10k attributable to this one flight. By contrast, the cost of an inflight meal (wholesale) is probably less than \$10 per person. For a flight with 200 passengers, that comes to \$1000-2000 dollars total. I think it is interesting to see that — all things considered — the influence of the inflight meal on the overall cost structure of the flight is as high as it is...about 2 percent of the total. In a business with thin margins, improving profitability by 2 percent is usually seen as worthwhile. In other words, we should be grateful that we get *anything* at all! A final cross-check: the cost per passenger for the entire flight from the airline's point of view is \$375 — and at the time of this writing, the cheapest fare I could find was \$600 roundtrip, equivalent to \$300 for a single leg. As is well-known, airlines break even on economy class passengers, but don't make any profits.)

17.3.3 Capital Expenditure and Operating Cost

A final distinction is the one between *capital expenditure* (CapEx) and *operating expense* (OpEx — the abbreviation is rarely used). Capital expenses is money spent to purchase long-lived, and typically tangible assets: equipment, installations, real estate. Operating expenses are everything else: payments for rents, raw materials, fees, salaries. In most companies, separate budgets exist for both forms of expenses and the availability of funds from each bucket may be quite different. For example, in a company that is financially strapped, but does have a revenue stream, it may be quite acceptable to hire and "throw people" at a problem (even at great cost), but it may very well be impossible to buy a piece of equipment that would take care of the problem for good. Conversely, in companies that do have money in the bank, it is often *easier* to get a lump sum approved for a specific purchase, rather than to hire more people or to perform maintenance. Decision makers often are more inclined to approve funding for an identifiable and visible purchase, rather than spending money on "business

as usual". Political and vanity considerations may play a role as well.

The distinction between CapEx and operating costs is important because depending on the availability of funds from either source, different solutions will be seen as feasible. (I refer to such considerations as "Color of Money" issues — although all dollars are green, some are greener than others!)

In the context of capital expenditure, there is one more concept that I'd like to introduce because I think it provides an interesting, and often useful way of thinking about money, and that is the notion of *depreciation*.² The idea is this: any piece of equipment that we purchase will have a useful service life. We can now distribute the total cost of the purchase across the entire life of that asset. For example, if I purchase a car for \$24,000 and expect to drive it for 10 years, then I can say that this car costs me \$200 per month "in depreciation", and before taking into account any operating costs (such as gas and insurance). I may want to compare this number with monthly lease payment options on the same kind of vehicle.

In other words, depreciation is a formalized way to capture how an asset loses value over time, and there are different standard ways to calculate it: "straight-line" distributes the purchase cost (less any *salvage value* that we might expect to obtain for the asset at the end of its life) evenly over the service life. The "declining balance" method assumes that the asset loses a certain fraction of its value every year. And so on. (Interestingly, land is never depreciated — because it does not wear out in the way a machine does, and therefore does not have a finite service life.)

I find depreciation a useful concept, because it provides a good way to think about large capital expenses: namely as an ongoing cost, rather than as an occasional lump sum. But it is just that: a way of thinking. It is important to understand that depreciation is *not* a cashflow and therefore does not show up in any sort of financial accounting. What's in the books is the money actually spent, when it is spent.

The one place where depreciation is being treated as a cashflow is when it comes to taxes. The IRS (the US tax authority) requires that certain long-lived assets purchased for business purposes are depreciated over a number of years, with the annual depreciation being counted as a business expense for that year. For this reason, depreciation is usually introduced in conjunction with tax considerations. But I find the concept more generally useful, as a way to think about and account for the cost of assets and their declining value over time.

²Distinguish between *to depreciate*, which is the process by which an asset loses value over time, and *to deprecate*, which is an expression of disapproval. The latter word is used most often to mark certain parts of a software program or library as *deprecated*, meaning that they should no longer be used in future work.

17.4 Should You Care?

What does all this talk about money, and business plans, and investment decisions have to do with data analysis? Why should you even care?

That depends. If you take a purely technical stance, then all of these questions are outside your area of competence and responsibility. That's a valid position to take, and many practitioners will make exactly that decision.

Personally, I disagree. I don't see it as my job to provide *answers* to *questions*. I see it as my responsibility to provide *solutions* to *problems*, and to do that effectively, I need to understand the context in which questions arise and I need to understand how answers will be evaluated and used. Furthermore, when it comes to questions having to do with abstract topics like data and mathematical modeling, I have found that few clients are in a good position to ask meaningful questions. *Coaching* the client on what makes a good question (one that is operational for me, and actionable for the client) is therefore a large part of what I do — and to do that, again I must understand and speak the client's language.

There are two more reasons why I find it important to understand issues such as those discussed in this (and the previous) chapter: establishing my own *credibility* and providing advice and counsel on the *math* involved.

The decision makers, that is, the people who request and use the results of a data analysis study, are "business men" (or women). They tend to see decisions as *investment* decisions, and will evaluate them using the methods and the terminology introduced in this chapter. Unless I understand how they will look at my results, and can defend them in those terms, I will be on weak ground, in particular since I am supposed to be "the expert". I learned this the hard way — I remember one situation, where I was presenting the results of a rather sophisticated and involved analysis, when some MBA bully fresh out of Business School challenged me with: "Ok now, which of these options has the best discounted net cash flow?" I had no idea what he was talking about. I looked like an idiot. This did *not* help my credibility! (No matter how right I was in everything else I was presenting.)

Another reason why I think it is important to understand the concepts in this chapter is that the math can get a little tricky, which is why the standard textbooks take recourse to large collections of pre-cooked scenarios — which is not only confusing, but can get downright misleading, if none of them fit exactly, and people start combining several of the standard solutions in ad-hoc (and probably incorrect) ways. Often, the most important skill I bring to the table is basic Calculus. In one place I worked for, which was actually staffed by some of the smartest people in the industry, I discovered a problem because people did not fully understand the difference between $1/x$ and $-x$. Of course, if you put it like this, everybody understands the difference. But if you muddy the waters a little bit and present the problem in the business domain setting in which they arose, it's no longer so easy to see the difference. (And I virtually guarantee you that nobody will understand why $1/(1-x)$ is actually close to $1-x$ for small x , when $1/x$ is not equal $-x$.)

In my experience, correct and meaningful application of basic math outside a purely mathematical environment pose an almost impossible challenge even for otherwise very bright people. Understanding what it is people are trying to do (for instance, in calculating a total rate of return) allows me to help them avoid serious mistakes.

But in the end, I think the most important reason is to be able to understand the *context* in which questions arise, and to be able to answer those questions appropriately with a sense for the *purpose* that was driving the original request.

17.5 Further Reading

If you want to read up on some of the details I have (quite intentionally) skipped, you should be looking for books on “Engineering Economics” or “Engineering Economic Analysis”. Some that I have found useful include:

- *Industrial Mathematics: Modeling in Industry, Science and Government*. Charles R. MacCluer. Prentice Hall. 1999.
In his preface, MacCluer points out that most engineers leaving school “will have no experience with problems incorporating the unit \$”. This observation was part of the inspiration for this chapter. MacCluer’s book contains an overview over many more advanced mathematical techniques that are relevant in practical applications. His choice of topics is excellent, but the presentation often seems a bit aloof and too terse for the uninitiated. (For instance, he covers the material in this chapter on only three pages.)
- *Schaum’s Outline of Engineering Economics*. Jose Sepulveda, William Souder, Byron Gottfried. McGraw-Hill. 1984.
If you want a quick introduction to the details left out in my presentation, then this inexpensive book is a good choice. Includes many worked examples.
- *Engineering Economy*. William G. Sullivan, Elin M. Wicks, C. Patrick Koelling. 14th ed., Prentice Hall. 2008.

Engineering Economic Analysis. Donald Newnan, Jerome Lavelle, Ted Eschenbach. 10th ed., Oxford University Press. 2009.

Principles of Engineering Economic Analysis. John A. White, Kenneth E. Case, David B.] Pratt. 5th ed., Wiley. 2000.

Three standard college-level textbooks, which treat the same material on many more pages.

Chapter 18

Predictive Analytics

18.1 Introduction

Data analysis can take many different forms — not only in the techniques that we apply, but also in the *kind* of results that we ultimately achieve. If we look back over the material that we have covered so far, we find that the results we obtained in part I were mostly *descriptive*: we tried to figure out what the data was telling us and to describe it. In contrast, the results in part II were primarily *prescriptive*: data was used as a guide for building models, and we then could infer or prescribe not-yet-observed phenomena from those models. In this form of analysis, data is not used directly, but only indirectly to guide (and verify!) our intuition when building models. Additionally, as I have tried to stress in those chapters, we don't just follow data blindly, but instead try to develop an understanding for the processes that generate the data and capture this understanding in the models that we develop. The predictive power of such models derives from this *understanding* that we developed by studying data and the circumstances in which it is generated.¹

In this chapter, we take a look at yet a different way to use data — we can call it *predictive*, since the purpose will be to make predictions about future events. What is different is that now we try to make predictions *directly from the data*, without forming the kind of conceptual model (and the associated deeper understanding of the problem domain) that we discussed in part II. That is obviously both a strength and a weakness: it is a strength, in that it enables us to deal with problems for which we have no hope of developing a conceptual model (because of the complexity of the situation). It is also a weakness, because we do not end up with deeper understanding, but only with a black-box solution. There are technical difficulties as well: the huge data sets required by this form of analysis are necessary because we are lacking the consistency and continuity guarantees made by a conceptual model. (We will

¹The techniques we discussed in part III belong neither here nor there: for the most part they were strictly computational and can be applied to any purpose, depending on the context.

come back to this point.)

The phrase *Predictive Analytics* is somewhat of an umbrella (others might say: marketing) term for different tasks, all of which share the intent of deriving predictive information directly from data. Three different specific application areas stand out:

Classification or Supervised Learning Assign each record to exactly one of a set of pre-defined classes. For example, classify credit card transactions as “valid” or “fraudulent”. Spam filtering is another example. Classification is considered “supervised”, because the classes are known and don’t need to be inferred from the data. Algorithms are judged on their ability to assign records to the correct class.

Clustering or Unsupervised Learning Group records into clusters, where the size and shape, and often even the number, of clusters is unknown. Clustering is considered “unsupervised”, because no information about the clusters is known ahead of time.

Recommendation Recommend a suitable item, based on past interest or behavior.

A fourth topic that is sometimes included is time-series forecasting, but I find that it does not share many characteristics with the other three, so I usually do not consider it part of predictive analytics itself. (See chapter 4 for more information on time series analysis.)

Of the three application areas, classification is arguably the most important and the best developed — in the rest of this chapter, I will try to give an overview over the most important algorithms and techniques for it. We discussed unsupervised learning in chapter 13 on clustering techniques, and I’ll repeat here what I stated there already, namely that I consider clustering more of an exploratory, and less of a predictive, technique. Recommendations are the youngest branch of predictive analytics, and quite different from the other two. (At least two major differences: on the technical side, many recommendation techniques boil down to network or graph algorithms, which have little in common with the statistical techniques used for classification and clustering. On the other hand, recommendations are *explicitly* about predicting human behavior, which poses additional difficulties not shared by systems which are ruled purely by natural laws.) I won’t have much to say about recommendation techniques in this book.

Let me emphasize that this chapter can only serve as an overview to classification. Entire books could (and have!) been written about it. But we can outline the problem, introduce some terminology, and give a flavor of different solution approaches.

18.2 Some Classification Terminology

We begin with a data set containing multiple elements, records, or *instances*. Each instance consists of several *attributes* or *features*. One of the features is special: it denotes the *class* and is known as the *class label*. Each record belongs to exactly one class.

A large number of classification problems are binary, consisting only of two classes (Valid/Fraudulent, Spam/Not Spam), but multi-class scenarios are also common. Many classification algorithms can only deal with binary problems, but this is no real limitation, since any multi-class problem can be transformed into a *set* of binary problems (does belong to the target class/does belong to any other class).

A *classifier* takes a record (that is: a set of attribute values) and produces a class label for this record. Building and using such a classifier generally follows a three-step process, of training, testing, and actual application.

We first split the existing data set into a *training* and a *test* set. In the training phase, we present each record from the training set to the classification algorithm. We then compare the class label produced by the algorithm to the true class label of the record in question, and adjust the “parameters” of the algorithm to achieve the greatest possible accuracy, or equivalently, the lowest possible error rate. (The details of this “fitting” process of course vary greatly from one algorithm to the next — we will look at the different ways in which this is done in the next section.)

The results can be summarized in a so-called *confusion matrix*, the entries of which are the number of records in each category (here for a binary classification problem with the class labels 0 and 1):

actual 0, predicted 0 : correct	actual 0, predicted 1 : error
actual 1, predicted 0 : error	actual 1, predicted 1 : correct

Unfortunately, the error rate that we get from the training set (the *training error*) is typically hopelessly optimistic as a indicator for the error rate that the classifier would achieve on new data, that is on data that was not used during the learning phase. This is what the test set is for: after we have optimized the algorithm using *only* the training data, we let the classifier operate on the elements from the test set to see how it does on them. The error rate obtained in this way is the *generalization error* and is a much more reliable indicator for the accuracy of the classifier.

To understand the need for a separate testing phase (using a separate data set!), keep in mind that as long as we use enough parameters (that is, making the classifier more and more complex), we can always tweak a classifier until it works very well for the given training set. But in doing so, we train the classifier to memorize every aspect of the training set, even those that are atypical for the system in general. We therefore need to find the right level of complexity for the classifier: if it is too simple, it cannot represent the desired behavior very well, and both its training and generalization error will be poor. This is known as *under-fitting*. On the other hand, if we make the classifier too complex,

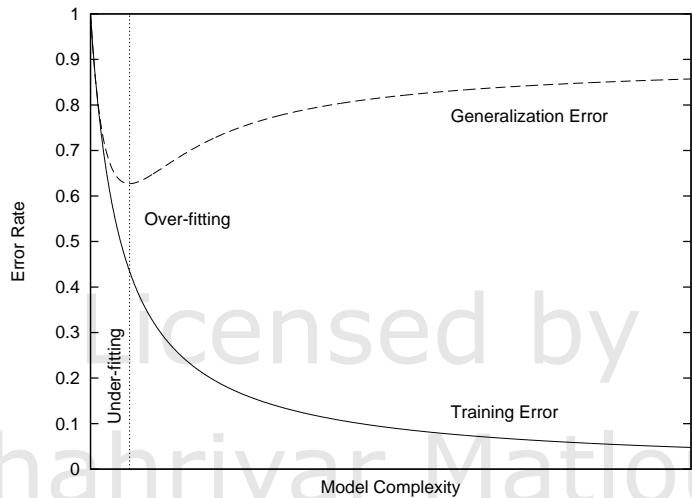


Figure 18.1: TBD

it will perform very well on the training set (low training error), but generalize poorly to unknown data points (high generalization error). This scenario is known as *over-fitting*. Figure 18.1 summarizes all these concepts.

Once a classifier has been developed and tested, it can be used to classify truly new and unknown data points — that is, data points for which the correct class label is not known (in contrast to the test set, where they were known, just not used by the classifier).

18.3 Algorithms for Classification

At least half a dozen different families of algorithms are being used for classification. In this section, I give a brief characterization of the basic idea underlying each algorithm, emphasizing how it differs from competing methods. The first two algorithms (Nearest Neighbor and Bayesian Classifiers) are both technically and conceptually simpler than the remaining ones; I discuss them in more detail since it is not unreasonable that you might want to implement them yourself. For the remaining algorithms, you probably should resort to existing libraries instead!

18.3.1 Instance-Based Classifiers and Nearest-Neighbor Methods

The idea behind instance-based classifiers is dead-simple: to classify an unknown instance, find an existing instance that is “most similar” to the new

instance and assign the class label of the known instance to the new one!

This basic idea can be generalized in a variety of ways. First of all, the notion of “most similar” brings us back to the notion of distance and similarity measures we discussed back in chapter 13: obviously there is considerable flexibility in the choice of distance measure to use. Furthermore, we don’t have to stop at a single “most similar” existing instance, instead we can take the nearest k neighbors and use them to classify the new instance, typically by using a majority rule (that is, we assign the new instance to the class that occurs most often among the k neighbors). Or we can even employ a weighted majority rule, where “more similar” neighbors contributed more strongly than those further away.

Instance-based classifiers are atypical in that they don’t have a separate “training” phase; for this reason they are also known as “lazy learners”. (The only adjustable parameter is the extent k of the neighborhood used for classification.) On the other hand, a possibly large set of known instances must be kept available during the final application phase. In the same spirit, classification can be relatively expensive, because the set of existing instances must be searched for appropriate neighbors.

Instance-based classifiers are *local*: they do not take the overall distribution of points into account. Additionally, they impose no particular shape or geometry on the decision boundaries that they generate. In this sense, they are particularly flexible. On the other hand, they are also susceptible to noise.

Lastly, they depend on the proper choice of distance measure, in the same way that clustering algorithms do. We have encountered this situation before, when we talked about the need for scale normalization in chapters 13 and 14 — the same points apply here as well.

18.3.2 Bayesian Classifiers

A Bayesian classifier takes a probabilistic (that is: non-deterministic) view of classification. Given a set of attributes, it calculates the probability for the instance to belong to this or that class. An instance is then assigned the class label with the highest probability.

A Bayesian classifier gives a *conditional* probability: the probability for the instance to belong to a specific class, *given* the set of attribute values:

$$P(\text{ class } C \mid \{x_1, x_2, x_3, \dots, x_n\})$$

Here, C is the class label, and the set of attribute values is $\{x_1, x_2, x_3, \dots, x_n\}$. Note that we don’t know the value of the probability yet — if we did, we would be finished.

To make progress, we invoke Bayes Theorem (hence the name of the classifier), to invert this probability expression (I am collapsing the set of n features into $\{x_i\}$ for brevity):

$$P(\text{ class } C \mid \{x_i\}) = \frac{P(\{x_i\} \mid \text{ class } C) P(\text{ class } C)}{P(\{x_i\})}$$

The first term in the numerator (the likelihood) is the probability to observe a set of features $\{x_i\}$, if the instance belongs to class C (in the language of conditional probability: *given* the class label C). We can find an empirical value for this probability from the set of training instances: it is nothing more than the frequency with which we observe the set of specific attribute values $\{x_i\}$ among instances belonging to class C . The second term in the numerator, $P(\text{class } C)$, is the prior probability for any instance to belong to class C . We can estimate it from the fraction of instances in the training set that belong to class C . The denominator does not depend on the class label, and as usual in Bayesian computations, is ignored until the end, when the probabilities are normalized.

Through the use of Bayes Theorem, we have now been able to express the probability for an instance to belong to class C , given a set of features, entirely through expressions which we can determine from the training set.

At least in theory. In practice, it will be very difficult to evaluate the expression above directly. Look closely at the expression (now written again in its long form): $P(\{x_1, x_2, x_3, \dots, x_n\} | \text{class } C)$. For each possible combination of attribute values, we must have a sufficient number of examples in our training set to be able to evaluate their frequency with some degree of reliability. This is a combinatorial nightmare! Assume that each feature is binary, that is, can take on only two values. The number of possible combinations is then 2^n — for $n = 5$, we already have 32 different combinations. Let's say that we need about 20 example instances for each possible combination to evaluate the frequency, you can see that we need a training set of at least 600 instances. In practice, the problem tends to be worse, because features frequently can take on more than 2 values, the number of features is easily larger than five, and — most importantly — some combinations of features occur much less frequently than others: we therefore need to have a training set large enough to guarantee that even the least frequent attribute combination occurs sufficiently often.

In short, the “brute force” approach of evaluating the likelihood function for all possible feature combinations is not feasible for problems of realistic size. Instead, one uses one of two simplifications.

The *Naive Bayesian Classifier* assumes that all features are independent of each other, so that we can write:

$$P(\{x_1, x_2, x_3, \dots, x_n\} | C) = P(x_1 | C)P(x_2 | C)P(x_3 | C) \dots P(x_n | C)$$

This simplifies the problem greatly, because now we only need to determine the frequencies for each attribute value for a *single* attribute at a time. In other words, each probability distribution $P(x_i | C)$ is given as the histogram of feature x_i , separately for each class label. Despite their simplicity, Naive Bayesians are often surprisingly effective. (Many Spam filters work this way.)

Another idea is to use a so-called *Bayesian Network*. Here we utilize the fact that there are often causal relationships between different attributes.

Coming soon:
Example for Bayesian Network

There are some practical issues that need to be addressed when building Bayesian classifiers. The description given above silently assumes that all attributes are categorical (that is: take on only a discrete set of values). Attributes that take on continuous numerical values either need to be discretized, or we need to find the probability $P(\{x_i\} | C)$ through a kernel density estimate (see chapter 2) of all the points in class C in the training set. If the training set is large, the latter process may be expensive.

Another tricky detail concerns attribute values that do not occur in the training set. The corresponding probability is therefore zero. But since a naive Bayesian classifier consists of a product of probabilities, it is zero as soon as a single one of the terms becomes zero! In particular for small training sets, this is a problem to watch out for. On the other hand, naive Bayesian classifiers are robust towards *missing* features, that is instances for which information about an attribute value is unknown for some of the instances: the corresponding probability simply evaluates to one and does not affect the final result.

18.3.3 Regression

Sometimes we have reason to believe that there is a functional relationship between the class label and the set of features. For example, we might assume that there is some relationship between an employee's salary and his or her status (Employee or Manager). See figure 18.2.

If it is reasonable to assume a functional relationship, we can try to build a classifier based on this relationship by "fitting" an appropriate function to the data. This turns the classification problem into a *regression* problem.

However, as we can see in figure 18.2, a linear function is typically not very appropriate, since it takes on all values, whereas class labels are discrete. Instead of fitting a straight line, we therefore want to fit something like a step function: a function that is zero for points belonging to the one class, and one for points belonging to the other class. Because of its discontinuity, the step function is hard to work with and one therefore typically uses the logistic function (B) as a smooth approximation to the step function. The logistic function gives the technique its name: *logistic regression*. Like all regression methods, it is a global technique, which tries to optimize a fit over all points, not just a particularly relevant subset.

Logistic regression is not only important in practical applications, but has deep roots in theoretical statistics as well. Until the arrival of Support Vector Machines, it has been the method of choice for many classification problems.

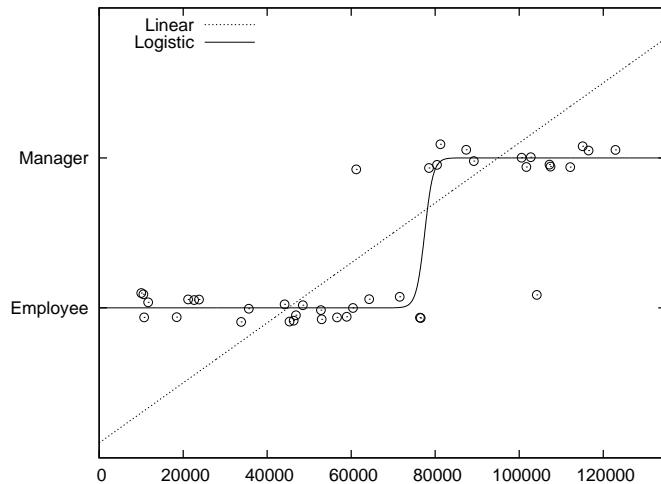


Figure 18.2: TBD

18.3.4 Support Vector Machines

Support Vector Machines are a relative new-comer among classification methods. The name is a bit unfortunate: there is nothing particularly “machiny” about them. Instead, they are based on a simple geometrical construction.

Consider training instances in a two-dimensional feature space, like the one in figure 18.3. Now we are looking for the “best” dividing line (or *decision boundary*) that separates instances belonging to one class from instances belonging to the other.

We need to decide what we mean by “best”. The answer given by Support Vector Machines is that the “best” dividing line is one that has the largest *margin*, where the margin is the space, parallel to the decision boundary, that is free of any training instances. Figure 18.3 shows two possible decision boundaries and their relative margins. Although this example is only two-dimensional, the reasoning generalizes directly to higher dimensions, in which case the decision boundary becomes a hyper-plane, and Support Vector Machines therefore find the *maximum margin hyper-planes*, which is a term you might find in the literature.

I will not go through the geometry and algebra to actually construct a decision boundary from a data set, since you probably don’t want to implement it yourself, anyway. (The construction is not hard, and if you have had some analytic geometry, you will be able to do it yourself, or look it up elsewhere.) The important insight is that Support Vector Machines turn the task of finding a decision boundary first into the geometric task of constructing a line from a set of points (which is an elementary task in analytic geometry). The next

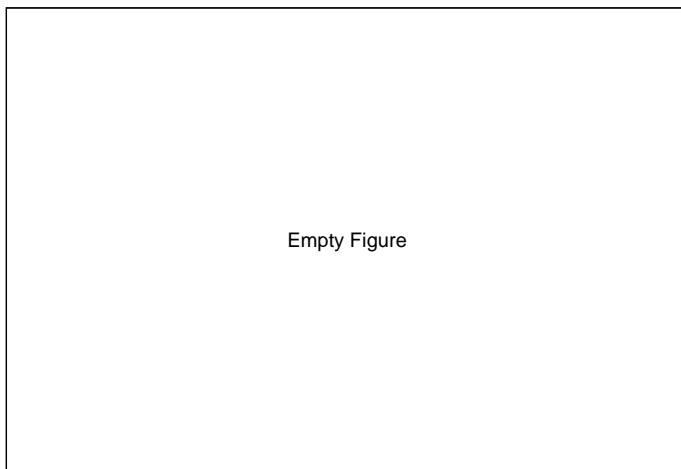


Figure 18.3: TBD

step, to find the decision boundary with the largest margin, is then merely a multi-dimensional optimization problem, with a particularly simple and well-behaved objective function (namely the square of the distance of each point to the dividing line), for which good numerical algorithms exist.

One important property of Support Vector Machines is that they perform a strict global optimization, without having to rely on heuristics. Because of the nature of the objective function, the algorithm is guaranteed to find the global optimum (not merely a local one). On the other hand, the final solution does not depend on all points, but only on those closest to the decision boundary, which lie right on the edge of the margin. (These are the so-called *support vectors*.) This means that the decision boundary does only depend on instances close to it, and is not influenced by the behavior of the system far from the decision boundary. However, the global nature of the algorithm implies that for those support vectors, the optimal hyper-plane will be found!

Two generalizations of this basic concept are of great practical importance. First consider figure 18.3 again. We were fortunate that we could find a straight line (in fact: more than one) to separate the data points exactly into two classes, so that both decision boundaries shown have zero training error. In reality, this is not guaranteed, and we may have some stray instances that cannot be classified correctly by any straight-line decision boundary. More generally, it may be advantageous to have a few mis-classified training instances, in return for a much wider margin, because it is reasonable to assume that a larger margin will lead to a lower generalization error later on. In other words, we may want to find a balance between low training error and large margin size. This can be done through the introduction of so-called *slack variables*. Basically, we as-

sociate a cost with each mis-classified instance and now try to optimize this extended optimization problem, in which we try to minimize the cost of mis-classified instances while at the same time trying to maximize the margins.

The other very important generalization allows us to use curves other than straight lines as decision boundaries. This is usually achieved through *kernelization* or the “kernel trick”. The basic idea is that we can replace the dot product between two vectors (which is central to the geometric construction required to find the maximum margin hyper-plane) by a more general function of the two vectors. As long as this function meets certain requirements (you may find references to “Mercer’s Theorem” in the literature), it can be shown that all the previous arguments continue to hold.

One disadvantage of Support Vector Machines is that they lead to particularly opaque results: they truly are black-boxes. The final classifier may work very well in practice, but it does not shed much light on the nature of the problem. This is in contrast to techniques such as regression or decision trees (see below), which often lead to results that can be interpreted in some form (in regression problems, for instance, one can often see which attributes are the most influential ones, and which are less relevant).

18.3.5 Decision Trees and Rule-Based Classifiers

Decision trees and rule-based classifiers are different from the classifiers discussed before, because they do not require a distance measure — for this reason, they are sometimes referred to as *non-metric classifiers*.

Decision trees consist of a hierarchy of decision points (the nodes of the tree). To classify an unknown instance using a decision tree, a single feature is examined on each node of the tree. Based on the value of that feature, the next node is selected. Leaf nodes on the tree correspond to classes; once we have reached a leaf node, the instance in question is assigned the corresponding class label. See figure 18.4 for an example of a simple decision tree.

The primary algorithm (*Hunt’s Algorithm*) to derive a decision tree from a training set employs a greedy approach. The algorithm is easiest to describe when all features are categorical and can take on only two values (binary attributes). If this is the case, then the algorithm proceeds as follows:

1. For each instance in the training set, examine each feature in turn.
2. Split the training instances into two subsets, based on the value of that feature.
3. Select the feature which results in the “purest” subsets: the value of this attribute will be the decision condition employed by the current node.
4. Repeat this algorithm recursively on the two subsets, until the resulting subsets are sufficiently pure.

To make this concrete, we need to be able to measure the purity of a set. Let f_C be the fraction of instances in the set belonging to class C . Obviously,

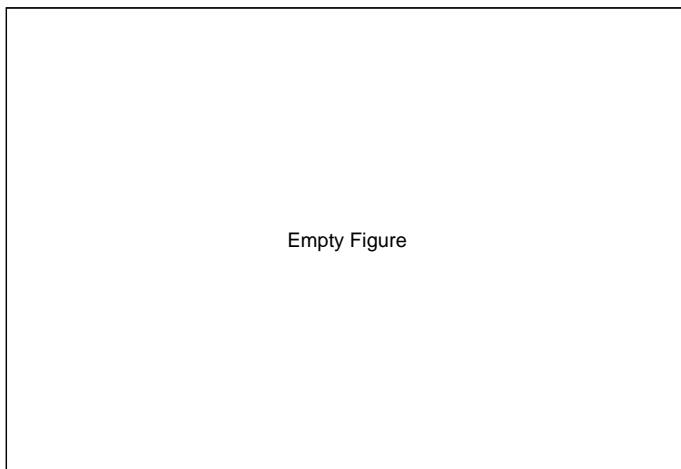


Figure 18.4: TBD

if $f_C = 1$ for any class label C , then the set is totally pure, because all of its elements belong to the same class. We can therefore define the purity of a set as the frequency of the most common of its constituents. (In other words, if a set consists to 60% of items of class A, 30% of class B, and 10% of class C, then we would say that its purity is 60%.) This is not the only way to define purity, other definitions are also possible, as long as they reach a maximum when all elements of a set belong to the same class, and reach a minimum when the elements of the set are distributed uniformly across classes.

Another important quantity when dealing with decision trees is the *gain ratio* Δ from a parent node to its children: it measures the gain in purity from parent to children, weighted by the relative size of the subsets:

$$\Delta = I(\text{parent}) - \sum_{\text{children } j} \frac{N_j}{N} I(\text{child } j)$$

where I is the purity (or impurity) of a node, N_j is the number of elements assigned to child node j , and N is the total number of elements at the parent. We want to find a splitting that maximizes this gain ratio.

What I have described so far is the outline of the basic algorithm. As with all greedy algorithms, there is no guarantee that the solution that will be found is optimal, and therefore heuristics of various sorts play a large role, to ensure that the solution is as good as possible, and the various published (or proprietary!) algorithms for decision trees (you may find reference to CART, C4.5, or ID3), differ in details such as:

- What choice of purity/impurity measure is used?

- At what level of purity does the splitting procedure stop? (Continuing to split a training set until all leaf nodes are entirely pure usually results in over-fitting.)
- Is the tree binary, or can a node have more than two children?
- How should be non-categorical attributes be treated? (For attributes that take on a continuum of values, we also need to define the optimal splitting point.)
- Is the tree post-processed? (Some algorithms employ a “pruning” step, which attempts to eliminate leaf nodes with two few elements, in an attempt to reduce over-fitting.)

Decision trees are very popular, and combine several attractive features: with good algorithms, decision trees are relatively cheap to build, and are always very fast to evaluate. They are also rather robust in the presence of noise. It can even be instructive to examine the decision points of a decision tree, because they frequently reveal interesting information about the distribution of class labels (such as when 80 percent of the class information is contained in the single, top-most attribute). On the other hand, algorithms to build decision trees are almost entirely black-box and do not lend themselves to ad-hoc modifications or extensions.

There is an equivalence between decision trees and *rule-based classifiers*. Rule-based classifiers consist of a set of rules (that is, logical conditions on attribute values), which, taken in aggregate, determine the class label of a test instance. There are two ways to build a rule-based classifier: one the one hand, we can build a decision tree first, and then turn each complete path through the decision tree into a single rule. Alternatively, we can build rule-based classifiers directly from a training set, by finding a subset of instances, that can be described by a simple rule. These instances are subsequently removed from the training set, and the process is repeated. (This amounts to a bottom-up approach, while building a decision tree using a variant of Hunt’s algorithm is following a top-down approach.)

18.3.6 Other Classifiers

Besides the classifiers mentioned above, you will find others mentioned in the literature. I’ll name just two — mostly because of their historical importance.

Fisher’s Linear Discriminant Analysis was one of the first classifiers that were developed. It is similar to Principal Component Analysis (chapter 14). Whereas for Principal Component Analysis, we introduced a new coordinate system so that the spread along the new coordinates was maximized, in Linear Discriminant Analysis, we introduce new coordinates so that the locations of the two classes that we try to distinguish are maximally separated from each other. The position of the means, calculated separately for each class, are taken as the location of each class.

Artificial Neural Networks were conceived as extremely simplified models for biological brains. The idea was to have a network of nodes, each of which receives input from several other nodes, forms a weighted average of its input, and then sends it out to the next layer of nodes. During the learning stage, the weights used in the weighted average are adjusted to minimize training error. Neural networks were very popular for a while, but have somewhat fallen out of favor recently. On the one hand, the calculations required are more complicated than for other classifiers, while on the other hand the whole concept is very ad-hoc and lacks a solid theoretical grounding.

18.4 The Process

In addition to the primary algorithms for classification, various techniques are important to deal with practical problems. In this section, we look at some standard methods that are commonly used to enhance accuracy, in particular in the important case that the most “interesting” type of class occurs much less frequently than the standard class or classes.

18.4.1 Ensemble Methods: Bagging and Boosting

The term *ensemble methods* describes a set of techniques to improve accuracy by combining the results of individual or “base” classifiers. The rationale is the same as when performing some experiment or measurement multiple times and averaging the results: as long as the experimental runs are independent, we can expect the errors to cancel, and the average to have higher accuracy than each individual trial. The same logic applies to classification techniques: as long as the individual base classifiers are independent, a combination of their results will see a cancellation of errors, and the end result will have greater accuracy than the individual contributions.

To generate a set of independent classifiers, we have to introduce some randomness into the process by which they are built. We can try and manipulate basically any aspect of the overall system: we can play with the selection of training instances (as in bagging and boosting), with the selection of features (often in conjunction with random forests), or with parameters that are specific to the kind of classifier used.

Bagging is an application of the bootstrap idea (see chapter 12) to classification. We generate additional training sets by sampling with replacement from the original training set. Each of these training sets is then used to train a separate classifier instance. During production, we let each of these instances provide a separate assessment for each item we want to classify. The final class label is then assigned based on a majority vote or similar technique.

Boosting is another technique to generate additional training sets using a bootstrap approach. However, in contrast to plain bagging, boosting is an iterative process, which assigns higher weights to samples that have been misclassified in previous rounds. As the iteration progresses, higher emphasis is

placed on training instances that have proven hard to classify correctly. The final result consists of the aggregate result of all base classifiers generated during the iteration. A particularly popular variant of this technique is known as “AdaBoost”.

Random Forests are a method specifically for decision trees. In this technique, randomness is not introduced by sampling from training set, but by making a random choice of features to use in building the decision tree. Rather than examining all features at every node to find the feature that gives the greatest gain ratio, only a subset of features is evaluated for each tree.

18.4.2 Estimating Prediction Error

Coming soon:
Crossvalidation

18.4.3 Class Imbalance

A particular important special case concerns situations where one of the classes occurs much less frequently in any data set than any of the other classes — and as luck would have it, that's usually the class we are interested in! Think credit card fraud detection, for instance: only one out of every hundred credit card transactions may be fraudulent, but those are exactly the ones we are interested in. Screening lab results for patients with elevated heart attack risk, or manufactured items for defects, falls into the same camp: the “interesting” cases are rare, possibly even extremely rare, but those are precisely the cases that we would like to be able to identify.

For these situations, there is a bit of additional terminology, and some special techniques to overcome the technical difficulties. Because there is one particular class that is of greater interest, we refer to an instance belonging to this class as a *positive event* and the class itself as the *positive class*. With this terminology, the entries in the familiar confusion matrix are often referred to as follows:

True Positive	False Negative
False Positive	True Negative

I always find this terminology very confusing — partially, because what is called “positive” is usually something *bad*: a fraudulent transaction, a defective item, a bad heart. Use the table below as a cheat sheet:

True Positive: Predicted bad, truly bad — “Hit”	False Negative: Predicted good, truly bad — “Miss”
False Positive: Predicted bad, but actually good — “False Alarm”	True Negative: Predicted good, but actually good — “True Alarm”

The two different kinds of errors may have very different costs associated with them. From the point of view of a merchant accepting credit cards as payment, a false negative (that is, a fraudulent transaction incorrectly classified as “not fraudulent”) results in the total loss of the item purchased, whereas a false positive (a valid transaction, incorrectly classified as “not valid”) results only in the loss of the profit margin on that item.

The usual metrics, by which we evaluate a classifier (such as accuracy and error rate), may not be very helpful in situations with pronounced class imbalances: keep in mind that the trivial classifier, which labels *every* credit card transaction as “valid” is 99 percent accurate — and entirely useless! Two metrics that provide better insight into the ability of a classifier to detect instances belonging to the positive class are *recall* and *precision*:

$$\text{precision} = \frac{TP}{TP + FP} \quad \text{fraction of correct classifications among all instances labeled positive}$$

$$\text{recall} = \frac{TP}{TP + FN} \quad \text{fraction of correct classifications among all instances labeled negative}$$

You can see that we will need to strike a balance: on the one hand, we can build a classifier that is very aggressive, labeling many transactions as “bad”, but it will have a high false positive rate, and therefore low precision. On the other hand, we can build a classifier that is very selective, marking only those instances that are blatantly fraudulent as “bad”, but it will have a high rate of false negatives, and therefore low recall. These two competing goals (to have few false positives and few false negatives) can be summarized in a graph known as a *receiver operating characteristic* or ROC curve. (The concept originated in signal processing, where it was used to describe the ability of a receiver to distinguish a true signal from a spurious one in the presence of noise, hence the name.)

Figure 18.5 shows an example of a ROC curve. Along the horizontal axis we plot the false positive rate (good events that were labeled as bad) and along the vertical axis we plot the true positive rate (bad events labeled as bad). The bottom left-hand corner corresponds to an extremely conservative classifier, which labels every instance as good; the top right-hand corner corresponds to an extremely aggressive classifier, which labels everything as bad. We can now imagine that we tune the parameters and thresholds of our classifier to shift the balance between “misses” and “false alarms” and in doing so, we map out the characteristic curve for our classifier. The curve for a random classifier (which assigns a positive class label with fixed probability p , irrespective of attribute values) will be close to the diagonal, since it is equally likely to classify a good instance as good as it is to classify a bad one as good, hence its false positive rate equals its true positive rate. By contrast, the ideal classifier will have a true positive rate equal to 1 throughout. We want to tune our classifier to approximate the ideal classifier as much as possible.

Class imbalances pose some technical issues during the training phase: if positive instances are very rare, we want to make sure to retain as much of their information as possible in the training set. One way to achieve this is through oversampling (that is: resampling) from the positive class instances, and undersampling from the negative class instances, when generating a training set.

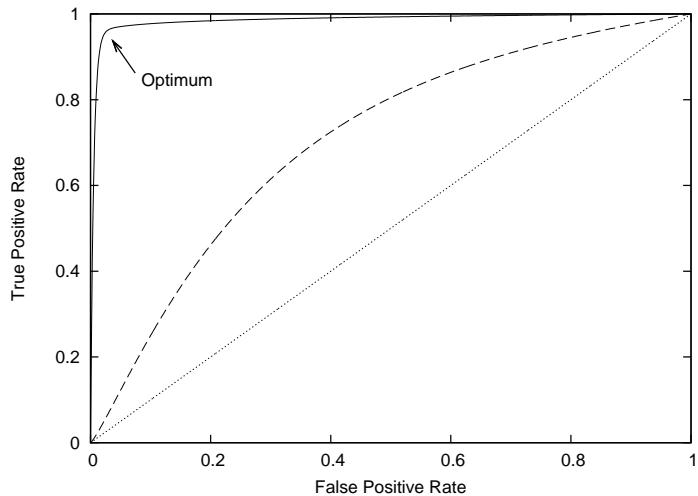


Figure 18.5: TBD

18.5 The Secret Sauce

All this detail about different algorithms and processes can easily leave the impression that that's all there is to classification. That would be unfortunate, because it leaves out what can be the most important, but also the most difficult part of the puzzle: finding the right attributes!

The choice of attributes matters for successful classification; arguably more so than the choice of classification algorithm. Here is an interesting case story. Paul Graham has written two essays on using Bayesian classifiers for spam filtering.² In the second one, he describes how using the information contained in the email *headers* is critical to obtaining good classification results, whereas using only information in the *body* is not enough. The punch line here is clear: in practice, it matters a lot which features you choose to include.

Unfortunately, there is not much information available on how to choose, prepare, and encode features for classification. Apparently, every problem domain is sufficiently different that it is difficult to come up with generally applicable recommendations. In the end, it comes down to experience, familiarity with the problem domain, and lots of time consuming trial-and-error. (The contrast to the development on the theoretical side, where there is a growing collection of often very complicated (or complicated *looking*) algorithms and calculational schemes, is striking — and a bit misleading!)

The difficulty to develop some recommendations that work in general and for a broad range of application domains may also be the reason for one par-

²"A Plan for Spam" <http://www.paulgraham.com/spam.html> and "Better Bayesian Filtering" <http://www.paulgraham.com/better.html>

ticular observation regarding classification: I am referring to the apparent absence of spectacular, well-publicized successes of classification. Spam filtering seems to be about the only application that clearly works and affects many people directly. Credit card fraud detection and credit scoring are two other widely used (if less directly visible) applications. But beyond those two, I only see a whole host of smaller, specialized applications. Which suggests again, that every successful classifier implementation depends strongly on the details of the particular problem — probably more so than on the choice of algorithm.

18.6 The Nature of Statistical Learning

Having seen some of the most commonly used algorithms for classification, as well as some related practical techniques, it is easy to feel a little overwhelmed — there seem to be so many different approaches (all of them non-trivial in their own way) that it can be hard to see the commonalities among them: the “big picture” easily gets lost. So, let’s step back for a moment and reflect on the specific challenges posed by classification problems and on the overall strategy by which the various algorithms overcome these challenges.

The crucial problem is that from the outset we don’t have good insight into which features are the most relevant in predicting the class — in fact, we may have no idea at all about the processes (if any!) that link observable features to the resulting class. Because we don’t know ahead of time which features are likely to be most important, we need to retain them all, and even expand the feature set in an attempt to include any possible clue we can get. In this way, the problem quickly becomes very multi-dimensional. That’s the first challenge.

But now we run into a problem: multi-dimensional data sets are invariably *sparse* data sets. Think of a histogram, with (let’s say) five bins per dimension. In one dimension, we have five bins total. If we want on average at least five items per bin, we can make do with twenty-five items total. Now consider the same data set in two dimensions. If we still require five bins per dimension, we have a total of twenty-five bins, so that each bin contains on average only a single element. But it is in three dimensions that the situation becomes truly dramatic: now there are 125 bins, so that we can be sure that the majority of bins will contain *no* element at all! It gets even worse in higher dimensions. (Mathematically speaking the problem is that the number of bins grows exponentially with the number of dimensions: N^d , where d is the number of dimensions and N is the number of bins per dimension. No matter what you do, the number of cells is going to grow faster than you can obtain data.) That’s the second challenge.

It is this combinatorial explosion that drives the need for larger and larger data sets: as we just saw, the number of possible attribute value combinations grows exponentially, so if we want to have a reasonable chance of finding at least one instance of each possible combination in our training data, we need to have very, very large data sets indeed. But despite our efforts, we will fre-

quently end up with a sparse data set as discussed above. Nevertheless, we will often deal with inconveniently large data sets. That's the third challenge.

Basically all classification algorithms deal with these challenges through some form of *interpolation* between points in the sparse data set. In other words, they attempt to smoothly fill the gaps left in the high-dimensional feature space, supported only by a (necessarily sparse) set of points (namely the training instances).

They do this in different ways: nearest-neighbor methods and naive Bayesian classifiers explicitly "smear out" the training instances to fill the gaps locally, whereas regression and support-vector classifiers construct global structures as a way to form a smooth decision boundary from the sparse set of supporting points. Decision trees are similar to nearest-neighbor methods in this regard, but provide a particularly fast and efficient lookup of the most relevant neighbors. But all algorithms try to fill the gaps between the existing data points in some smooth, consistent way.

This brings up the question what can be predicted in this fashion? Obviously, class labels need to depend on attribute values, and they should depend on them in some form of smooth, predictable fashion. If the relationship between attribute values and class labels is too crazy, no classifier will be very useful.

Furthermore, the distribution of attribute values for different classes must *differ*, otherwise no classifier will be able to distinguish classes by examining the attribute values.

Unfortunately, there is — to my knowledge — no independent way to determine whether the information contained in a data set is sufficient to allow the data to be classified. To find out, we must build an actual classifier. If it works, then obviously there *is* enough information in the data set for classification. But if it does *not* work, then we have learned nothing, because it is always possible that a different or better classifier actually *might* work. But without an independent test, we can spend an infinite amount of time building and refining classifiers on data sets that contain no useful information. (We have encountered this kind of difficulty before, in chapter 13, when we talked about clustering, but it strikes me as even more of a problem here, because classification is by nature predictive (or at least should be), whereas this kind of uncertainty seems more acceptable in an exploratory technique such as clustering.)

To make this more clear: imagine we take a large, rich data set: many records with many features. We then arbitrarily assign class labels A and B to the records in the data set. Now, by construction, it is clear that there is no way to predict the labels from the "data" — they are, after all, purely random! However, there is no unambiguous test that will clearly say so. We can calculate the correlation coefficients between each feature (or combination of features!) and the class label, we can look at the distribution of feature values and see whether they differ from class to class, and so eventually convince ourselves that we won't be able to build a good classifier given this data set. But there is no clear test or diagnostic, which would give us (for instance) an upper

bound on the quality of any classifier that could be built, using this data set. If we are not careful, we may spend a lot of time, vainly trying and failing to build a classifier that will be able to extract useful information from this data set. This kind of problem is a trap to be aware of!

18.7 Toolbox

In particular when it comes to classification, it is really easy to end up with a “black box” solution: a tool or library, which provides an implementation of a classification algorithm — but we would not be able to write an implementation ourselves, if we had to. This kind of situation always makes me a bit uncomfortable, in particular if the algorithms require the tuning of parameters to work properly: to properly adjust such parameters, I need to understand the algorithm well enough that I could at least provide a rough-cut version myself (much as I am happy to rely on the library designer for the high-performance version).

In this spirit, instead of discussing an existing classification library, I want to show you how to write straightforward (you might say: toy version) implementations for two simple classifiers: a nearest-neighbor lazy learner, and a naive Bayesian classifier. (I’ll give some pointers to other libraries towards the end of this section.)

We will test our implementations on *the* classic data set in all of classification: Fisher’s Iris data set.³ The data set contains measurements of four different parts of an Iris flower (sepal length and width, petal length and width). There are 150 records in the data set, distributed equally among three species of Iris (Iris Setosa, Versicolor, and Virginica). The task is to predict the species, given a set of measurements.

First of all, let’s take a quick look at the distributions of the four quantities, to see whether it seems feasible to distinguish the three classes this way. Figure 18.6 shows histograms (actually: kernel density estimates) for all four quantities, separately for the three classes. One of the features (sepal width) does not seem very promising, but the distributions of the other three features seem sufficiently separated, that it should be possible to obtain good classification results.

As preparation, I split the original data set into two parts: a training set (in the file `iris.trn`) and a test set (in file `iris.tst`). I randomly selected five records from each class for the test set, the remaining records were used for training. The test set is shown in full in listing 18.1: the columns are (in order) sepal length, sepal width, petal length, petal width, followed by the class label. (All measurements are in centimeters, with millimeter precision.)

The nearest-neighbor classifier is shown in listing 18.2. It is exceedingly simple — in particular once you realize that about two-thirds of the listing

³First published in 1936. The data set is available from many sources, including the UCI Machine Learning repository: <http://archive.ics.uci.edu/ml/datasets/Iris>.

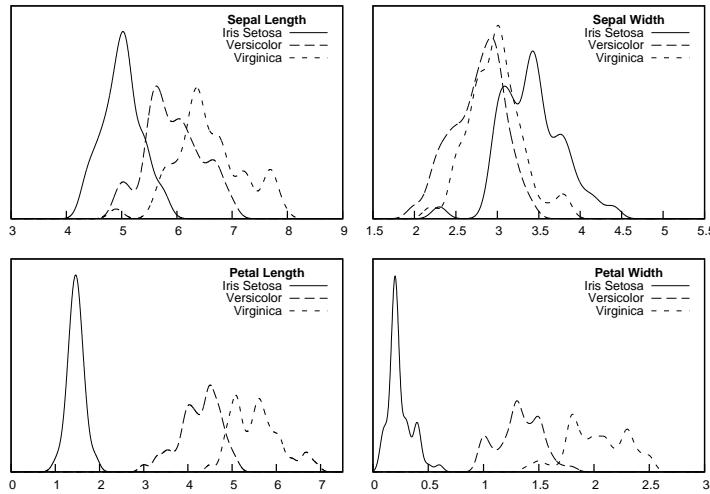


Figure 18.6: TBD

deal with file input and output. The actual “classification” is a matter of three lines in the middle.

The algorithm loads both the training and the test data set into two-dimensional NumPy arrays. Because all elements in a NumPy array must be of the same type, we keep the class labels in separate vectors.

Now follows the actual classification step: for each element of the test set, we calculate the Euclidean distance to each element in the training set. We make use of NumPy “broadcasting” (see the Workshop in chapter 2), to calculate the distance of the test instance `test[i]` from *all* training instances in one fell swoop. (The argument `axis=1` is necessary to tell NumPy that the sum in the Euclidean distance should be taken over the *inner* (horizontal) dimension of the two-dimensional array.) Next, we use the `argmin()` function to obtain the index of that training record which has the smallest distance to the current test record: this is our predicted class label. (Notice that we base our result only on a single record, namely the closest training instance.)

Simple as it is, the classifier works very well (on this data set!) — on the test set shown: all records in the test set are classified correctly!

The naive Bayesian classifier implementation is shown in figure 18.3. A naive Bayesian classifier needs an estimate of the probability distribution $P(\text{class } C \mid \text{feature } x)$, which we find from a histogram of attribute values, separately for each class. In this case, we need a total of twelve histograms (3 classes \times 4 features). I maintain this data in a triply nested data structure: `histo[label][feature][value]`. The first index is the class label, the second index specified the feature, and the third contains the values of the feature that occur in the histogram. The number of times that each value has been observed is stored in the histogram. I use

```

5.0,3.6,1.4,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
5.2,3.5,1.5,0.2,Iris-setosa
5.1,3.8,1.6,0.2,Iris-setosa
5.3,3.7,1.5,0.2,Iris-setosa
5.7,2.8,4.5,1.3,Iris-versicolor
5.2,2.7,3.9,1.4,Iris-versicolor
6.1,2.9,4.7,1.4,Iris-versicolor
6.1,2.8,4.7,1.2,Iris-versicolor
6.0,3.4,4.5,1.6,Iris-versicolor
6.3,2.9,5.6,1.8,Iris-virginica
6.2,2.8,4.8,1.8,Iris-virginica
7.9,3.8,6.4,2.0,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
6.5,3.0,5.2,2.0,Iris-virginica

```

Example 18.1: TBD

this kind of nested histogram data structure all the time.

(I'd like to point out two implementation details: the second index is an integer, which I use instead of the feature names. This simplifies some of the loops in the program. The second detail is more important: I know that the feature values are given in centimeters, with exactly one digit after the decimal point. In other words, the values are already discretized, so that I don't need to "bin" them even further — in effect, each bin in the histogram is one millimeter wide. Because I never need to operate on the feature values, I don't even convert them to numbers: I read them as strings from file and use them as strings as keys in the histogram. Of course, if we wanted to use a different bin width, then we would have to convert them into numerical values, so that we can operate on them.)

In the evaluation part, the program reads data points from the test set and evaluates the probability that the record belongs to a certain class for all three class labels. We then pick the class label that has the highest probability. (Notice that we don't need an explicit factor for the prior probability, since we know that each class is equally likely.)

On the test set from listing 18.1, the Bayesian classifier does a little worse than the nearest neighbor classifier: it correctly classifies 12 out of 15 instances, for a total accuracy of 80 percent.

If you look at the results of the classifier more closely, you will immediately notice a couple of problems that are common with Bayesian classifiers: first of all, the posterior probabilities are *small*. This should come as no surprise: each Bayes factor is smaller than one, so their product becomes very small very quickly. To avoid underflows, it usually makes sense to add the logarithms of the probabilities, rather than multiplying the probabilities directly. In particular if you have a greater number of features, this is a necessity. The other

```

from numpy import *

train = loadtxt( "iris.trn", delimiter=',', usecols=(0,1,2,3) )
trainlabel = loadtxt( "iris.trn", delimiter=',', usecols=(4,), dtype=str )

test = loadtxt( "iris.tst", delimiter=',', usecols=(0,1,2,3) )
testlabel = loadtxt( "iris.tst", delimiter=',', usecols=(4,), dtype=str )

hit, miss = 0, 0
for i in range( test.shape[0] ):
    dist = sqrt( sum( (test[i] - train)**2, axis=1 ) )
    k = argmin( dist )

    if trainlabel[k] == testlabel[i]:
        flag = '+'
        hit += 1
    else:
        flag = '-'
        miss += 1

    print flag, "\tPredicted: ", trainlabel[k], "\tTrue: ", testlabel[i]

print
print hit, "out of", hit+miss, "correct - Accuracy: ", hit/(hit+miss+0.0)

```

Example 18.2: TBD

problem is that many of the posterior probabilities come out as exactly zero: this occurs whenever for at least one of the feature values in the test record no entry in the histogram can be found: the histogram evaluates to zero, which means the entire product of probabilities is also identical to zero. There are different ways to deal with this problem — in our case, you might want to experiment with replacing the histogram of discrete feature values with a kernel density estimate (similar to those in figure 18.6, which are everywhere non-zero. Keep in mind that you will need to determine a suitable bandwidth for each histogram!

Let me be clear: the implementations of both classifiers are extremely simple-minded. My intention here is to demonstrate the basic ideas behind these algorithms in as few lines of code as possible, but also to show that there is nothing mystical about writing a simple classifier. Because the implementations are so simple, it is easy to continue experimenting with them: can we do better if we use a larger number of neighbors in the listing 18.2? How about a different distance function? In listing 18.3, we want to experiment with different bin widths in the histogram, or better yet, get rid of the histogram of discrete bins and replace it with a kernel density estimate. In either case, we need to start

thinking about run-time efficiency: for a data set consisting of only 150 elements this does not matter much, but evaluating a kernel density estimate of a few thousand points can get quite expensive!

If you want to use an established tool or library, there are several choices in the open-source world. Three projects have put together entire data analysis and mining “toolboxes”, complete with graphical user interface, plotting capabilities, and various plugins: RapidMiner⁴, WEKA⁵ (both Java) and Orange⁶ (Python). WEKA has been around for a long time and is very well established, RapidMiner is part of a more comprehensive tool suite (and includes WEKA as a plugin), Orange is an alternative using Python.

All three of these projects use a “pipeline” metaphor: you select different processing steps (discretizers, smoothers, Principal Component Analysis, regression, classifiers) from a toolbox and string them together to build up the whole analysis workflow entirely within the tool. Give it a shot — the idea has a lot of appeal, but I must confess that I have *never* succeeded in doing anything useful with any of them!

There are some additional libraries that are worth checking out, and that have Python interfaces: libSVM⁷ and Shogun⁸ provide implementations of Support Vector Machines, while the Modular toolkit for Data Processing (MDP)⁹ is more general. (The latter also adheres to the “pipeline” metaphor.)

Finally, basically all classification algorithms are also available as R packages. I’ll mention just three: the `class` package for nearest-neighbor classifiers and the `rpart` package for decision trees (both part of the R standard distribution) and the `e1071` package (which can be found on CRAN) for Support Vector Machines and naive Bayesian classifiers.

18.8 Further Reading

- *Introduction to Data Mining*. Pang-Ning Tan, Michael Steinbach, Vipin Kumar. Addison-Wesley. 2005.

This is my favourite book on data mining. Contains two accessible chapters on classification.

- *The Elements of Statistical Learning*. Trevor Hastie, Robert Tibshirani, Jerome Friedman. 2nd ed., Springer. 2009.

This book exemplifies some of the problems with current machine learning theory: a whole book of highly non-trivial mathematics — and what feels like not a single real-world example, or a discussion of “what to use when”.

⁴<http://rapid-i.com/>

⁵<http://www.cs.waikato.ac.nz/ml/weka/>

⁶<http://www.ailab.si/orange/>

⁷<http://www.csie.ntu.edu.tw/~cjlin/libsvm/>

⁸<http://www.shogun-toolbox.org/>

⁹<http://mdp-toolkit.sourceforge.net/>

```

total = {} # Training instances per class label
histo = {} # Histogram

# Read the training set and build up a histogram
train = open( "iris.trn" )
for line in train:
    # seplen, sepwid, petlen, petwid, label
    f = line.rstrip().split( ',' )
    label = f.pop()

    if not total.has_key( label ):
        total[ label ] = 0
        histo[ label ] = [ {}, {}, {}, {} ]

    # Count training instances for the current label
    total[ label ] += 1

    # Iterate over features
    for i in range( 4 ):
        histo[ label ][ i ][ f[ i ] ] = 1 + histo[ label ][ i ].get( f[ i ], 0.0 )

train.close()

# Read the test set and evaluate the probabilities
hit, miss = 0, 0
test = open( "iris.tst" )
for line in test:
    f = line.rstrip().split( ',' )
    true = f.pop()

    p = {} # Probability for class label, given the test features
    for label in total.keys():
        p[ label ] = 1
        for i in range( 4 ):
            p[ label ] *= histo[ label ][ i ].get( f[ i ], 0.0 ) / total[ label ]

    # Find the label with the largest probability
    mx, predicted = 0, -1
    for k in p.keys():
        if p[ k ] >= mx:
            mx, predicted = p[ k ], k

    if true == predicted:
        flag = '+'
        hit += 1
    else:
        flag = '-'
        miss += 1

    print flag, "\t", true, "\t", predicted, "\t",
    for label in p.keys():
        print label, ":", p[ label ], "\t",
    print

```

print

Chapter 19

Epilogue: Facts Are Not Reality

Coming soon

Appendix A

Programming Environments for Science and Data Analysis

A.1 Overview

Tools are important: although they don't limit what we problems *can* solve, they do determine what we consider an "easy problem". In regards to data analysis, the hard problem that we should be grappling with is always the data and what it is trying to tell us — the mechanics of it should be sufficiently convenient that we don't even think about them. We want to keep the technical hurdle low, so that we have no reason to hesitate to start working on any problem that seems promising or interesting. A sufficiently well-stocked toolbox is therefore essential.

But it is not only the size of the toolbox that matters, but also our mastery of the various elements within it. Only tools that we know well enough that using them feels effortless truly leverage our abilities.

Balancing these opposing trends (breadth of tool selection and depth of mastery) is a constant challenge. When in doubt, I recommend you opt for depth — superficiality does not pay.

On the other hand, it does not matter much *which* tools we use (as long as their technical limitations are not self-defeating). After all, the important aspect is not the tool itself, but the abilities that it bestows on us. This point is sometimes a little bit forgotten, in particular when talking of particularly complex or cutting-edge tools.

Properties I look for in a tool or programming environment include:

- Low overhead or ceremony: must be easy to get started on a new investigation.
- Facilitate iterative, interactive use.
- No arbitrary limitations (within reasonable limits).

- Scriptable: not strictly required, but often very nice to have.
- Stable, correct, mature; free of random defects or other annoying detractions.

Most of these items are probably not contentious. The one point I would like to emphasize concerns arbitrary limitations, imposed by the tool itself. These can seriously influence the nature of your results. Quick case story: on one project, I was able to discover very relevant information because I was able to generate literally hundreds of graphs, using a scriptable graphing program. Putting all of them on a web page next to each other revealed obvious similarities and differences between different data sets — a fact that had never been noticed before, not least because everybody else was using tools (mostly Excel) that would only allow graphs to be created one at a time. (Excel is notorious for unnecessarily limiting what can be done, and so is SQL: putting even minimal programming abilities on top of SQL greatly expands the range of problems that can be tackled.)

In summary, which programming environment you use does not matter, as long as you are comfortable and proficient with it, it gets the work done, and the limitations of the tool do not prevent you from doing things that you *should* be doing.

In this appendix, I am going to briefly survey and discuss a number of particularly popular programming environments that are relevant for the kind of data analysis discussed in the rest of the book. The emphasis is on open source software, although I will also mention some of the most important commercial players.

A.1.1 Scientific Software is Different

It is important to realize that scientific software (for a sufficiently wide definition of “scientific”) faces some unusual challenges. First of all, scientific software is *hard*. Writing high-quality scientific programs is difficult, requiring rather rare and specialized skills. (We’ll come back to this below.) Additionally, the market for scientific software is *small*, making it correspondingly harder for any one program or vendor to pick up critical mass.

Both of these issues affect all players equally, but a third problem poses a particular challenge for open source offerings: many users of scientific software are transients. Graduate students graduate, moving on from their projects and often leaving the research environment entirely. In consequence, *abandonware* is particularly frequent among open source scientific software projects. (And not just there — the long-term viability of commercial offerings is also far from assured.)

Before investing significant time and effort into mastering any one tool, it is therefore necessary to determine whether:

- the tool is of sufficiently high *quality*, and

- the tool is has strong enough *momentum* and *support*

to be worth the investment.

A.2 A Catalog of Scientific Software

There are currently three main contenders for interactive, numeric programming available: Matlab (and its open-source clone Octave), R (and its commercial predecessor S/S-Plus), and the NumPy/SciPy set of libraries for Python. Fundamentally, all three packages are vector/matrix packages: they treat vectors and matrices as atomic data types, and allow mathematical functions to operate on them directly (addition, multiplication, application of a function to all elements in a vector or matrix). In addition to this basic functionality, all three offer all kinds of additional mathematical operations, such as special functions, support for function minimization, or numerical integration and non-linear equation solving. It is important to keep in mind that all three are packages for *numerical* computations, operating on floating-point numbers. None of these three handles *symbolic* computations, such as the expansion of a function into its Taylor series. For this you need a symbolic math packages, such as Mathematica or Maple (both commercial) or Maxima/MACSYMA, SAGE, or Axiom (all three open-source).

In addition to these three, there are of course other players, but they tend to be smaller or to cater to more specialized niches. Also, while all three of the numerical packages offer similar functionality, each one of them has its own share of drawbacks, so that none of them comes out as clear winner.

A.2.1 Matlab

Matlab has been around since the mid 1980s, and it has a very large user base, mostly in the engineering professions, but also in pure mathematics and in the machine learning community. Rather than do all the heavy lifting itself, it was conceived as a user-friendly frontend to existing high-performance numerical linear algebra libraries (LINPACK and EISPACK, which have been replaced by LAPACK). Matlab was one of the first widely used languages to treat complex data structures (such as vectors and matrices) as atomic data types, allowing to operate with them as if they were scalar variables and without the need for explicit looping. (In this day and age, when object-oriented programming and operator overloading are commonly used and totally mainstream, it is hard to imagine how revolutionary this concept seemed when it was first developed.¹⁾) In 2008, The MathWorks (the company which develops Matlab) acquired the rights to the symbolic math package MuPAD and incorporated it into subsequent Matlab releases.

¹⁾I remember how blown away I personally was when I first read about such features in the programming language APL in the mid 80s!

Matlab was mostly designed to be used interactively, and its programming model has serious deficiencies for larger programming projects. (There are problems with abstraction and encapsulation, as well as memory management issues.) It is a commercial product, and although its academic licenses are very reasonably priced, a full commercial installation can cost as much as a (small) new car.

Matlab places particular emphasis on the quality of its numerical and floating point algorithms and implementations.

There is an open source clone to Matlab called Octave. Octave strives to be fully compatible; however, there are numerous reports about difficulties porting programs back and forth.

A.2.2 R

R is the open-source clone of the S/S-Plus statistical package, originally developed at Bell Labs. R has a very large user base, mostly in the academic statistics community, and a particularly healthy tradition of user-contributed packages. The *Comprehensive R Archive Network* (CRAN) is a huge central repository of user-contributed modules.

On the other hand, R was not conceived as a general purpose programming language, but is strongly geared towards statistical applications. Its programming model is quite different from current mainstream languages, making it surprisingly difficult for somebody with a strong programming background to switch to R. Finally, its primarily academic outlook fits awkwardly with a commercial enterprise environment.

The strongest point of R is the large number of built-in (or user-contributed) functions for primarily *statistical* calculations. In contrast to Matlab, R is not intended as a general numerical workbench (although it can, with some limitations, be used for that purpose). It is also (maybe contrary to expectations) not intended as a general-purpose data manipulation language, although it can serve as scripting language for text and file manipulations, and similar tasks.

One serious problem when working with R is its dated programming model. It relies strongly on implicit behavior and “reasonable defaults”, leading to particularly opaque programs. Neither the language nor the libraries provide strong support for organizing information into larger structures, making it uncommonly difficult to locate pertinent information. Although it is easy to pick up isolated “tricks”, it is notoriously difficult to develop a comprehensive understanding of the whole environment.

Like Matlab, R is here to stay. It has proven its worth (for 30 years!), it is mature, and it has a strong, high-caliber, and vocal user base. Unlike Matlab, it is free and open-source, making it easy to get started.

A.2.3 Python

Python has become a scripting language of choice for scientists and scientific applications, in particular in Machine Learning and the biological and social

sciences. (Hard-core, large-scale numerical applications in Physics and related fields continue to be done in C/C++ or — *horresco referens* — in Fortran.)

The barrier to programming in Python is low, making it easy to start new projects. That is somewhat of a mixed blessing: there is an abundance of exciting Python projects out there; on the other hand they seem to be particularly prone to the “abandonware” problem mentioned earlier. Also, scientists are not programmers, and it often shows (in particular in regards to long-term vision and the cultivation of a strong and committed community).

In addition to a very large number of smaller and more specialized projects, there have been five major attempts to provide a *comprehensive* Python library for scientific applications, and it can be confusing to understand how they relate to each other.² They are:

Numeric This is the original Python module for the manipulation of numeric arrays, initiated in 1995 at MIT. Superceded by NumPy.

numarray An alternative implementation from the Space Telescope Science Institute (2001). Considered obsolete, replaced by NumPy.

NumPy The NumPy project was begun in 2005 to provide a unified framework for numerical matrix calculations. NumPy builds on (and superceded) Numeric, and includes the additional functionality developed by numarray.

SciPy Started in 2001, the SciPy project evolved out of an effort to combine several previously separate libraries for scientific computing. Builds on and includes NumPy.

ScientificPython An earlier (started in 1997) general-purpose library for scientific applications. In contrast to SciPy, this library tries to stay with “pure Python” implementations, for better portability.

Today, the NumPy/SciPy project has established itself as the clear winner among general-purpose libraries for scientific applications in Python, and we will take a closer look at it below.

A strong point in favor of Python is the convenient support it has for relatively fancy and animated graphics. The matplotlib library is the most commonly used Python library to generate standard plots, and it has a particularly close relationship with NumPy/SciPy. Besides matplotlib, there are Chaco and Mayavi (for 2- and 3-dimensional graphics, respectively) and libraries such as PyGame and Pyglet (for animated and interactive graphics) — and of course many more.

Uncertainties associated with the future and adoption of Python3 affect all Python projects, but are particularly critical for many of the scientific and graphics libraries just mentioned: to achieve higher performance, these libraries

²For more information on the history and interrelations of these libraries, check out the first chapter in Travis B. Oliphant’s “Guide to NumPy”, which can be found on the Web.

usually rely heavily on C bindings, which do not port easily to Python3. Coupled with the “abandonware” issue discussed earlier, this poses a particular challenge for all scientific libraries based on Python at this time.

NumPy/SciPy

The NumPy/SciPy project has evolved to become the dominant player in scientific programming for Python. NumPy provides efficient vector- and matrix-operations; SciPy consists of a set of higher-level functions built on top of NumPy. Together with the matplotlib graphing library and the IPython interactive shell, they provide functionality resembling Matlab. NumPy/SciPy is open-source (BSD-style license) with a large user community and is supported and distributed by a commercial company (Enthought).

NumPy is intended to contain low-level routines for vector- and matrix-handling, and SciPy is meant to contain all higher level functionality. However, for reasons of backwards compatibility, some additional functions are included in NumPy, and for convenience, all NumPy functions are aliased into the SciPy namespace. In consequence, the distinction between NumPy and SciPy is not very clear in practice.

On the face of it, NumPy/SciPy should come out ahead among scientific computing environments: combining the wide range of functionality provided by SciPy with a modern, general-purpose programming environment, and coupled with a strong user base and sponsored by a commercial company.

On closer inspection, NumPy/SciPy do not quite meet these expectations. The project does not seem to be managed very professionally, with a persistent tendency to emphasize quantity over quality. The list of features claimed in the documentation is impressive, but the choice of algorithms, as well as their implementations, often appear somewhat amateurish. The design feels unnecessarily complicated, with a very large, but poorly organized API that is often awkward to use.

What worries me most is that although the project has been around for almost 10 years and has a large and very active user base, it has apparently not been able to achieve and maintain a consistent level of reliability and maturity throughout: while for most routines the “happy case” works just fine, you may find that edge and error cases are not handled properly. (The lack of comprehensive unit test coverage is openly acknowledged.) Despite occasional efforts in this regards, the documentation remains patchy. Features remain not or only partially implemented (while at the same time the API continues to grow).

NumPy/SciPy is interesting, because among scientific and numeric projects, it has probably the lowest barrier to entry, and is flexible and versatile. That makes it a convenient environment to get started and for casual use. However, because of the overall quality issues, I would not want to rely on it for “serious” production work at this point.

A.2.4 What about Java?

Java is largely a no-show for numerical computation, so much so that a Java Numerics Working Group ceased operations years ago (around the year 2002) for lack of interest. There are problems with performing floating point calculations in Java — Java Virtual Machine encapsulates the underlying platform and therefore makes it impossible for programs to take advantage of the floating point hardware.

On the other hand, a lot of production quality machine learning programming is done in Java, where Java's (compared to C) convenient string handling, and the affinity to all the *enterprise* programming in Java come into play. (Java is largely non-existent in the academic world.) One will have to see whether these applications will over time lead to the development of high-quality numerical libraries as well.

If you want a comfortable programming environment for large (possibly distributed) systems, that's relatively fast, Java is a reasonable choice. However, Java programming has become very heavy weight (with tools to manage your frameworks, and so on), which does not encourage ad-hoc, exploratory programming. Groovy carries less programming overhead, but is slow. A last issue concerns Java's traditionally not very strong capabilities for interactive graphics and UIs, in particular on Linux.

Java is very strong in regards to “Big Data” — in particular Hadoop, the most popular open-source map/reduce implementation is written in Java. Java is also popular for text processing and searching.

A relatively new project is Incanter, which uses Clojure (a Lisp dialect running on top of the Java virtual machine) to develop an “R-like statistical computing and graphics environment”. While Incanter is an interesting project, I don't feel that it has stood the test of time yet, and one will have to see how it will position itself compared to R.

A.2.5 Other Players

Of course, the preceding list of programs and packages is far from complete. Among the other players, I just want to briefly mention three:

SAS SAS is a classical statistics packages, with strongly established uses in credit scoring and medical trials. SAS was originally developed for OS/360 mainframes, and it shows. Its command language has a distinct 1960s feel, and the whole development cycle is strongly batch oriented (not interactive, not exploratory). SAS works best when a well-defined procedures need to be repeated often, and on large data sets. One of the unique features of SAS is that it works well with data sets that are too large to fit into memory and therefore need to be processed on disk.

SAS, like the mainframes it used to run on, requires specially trained operators — it is not for the casual user. (It is not exactly fun, either. Some-

body described using it as comparable to “scraping down the wallpaper with your fingernails.”)

SciLab SciLab is an open source project similar to Matlab, created by the French research institute INRIA.

GSL The GSL (the Gnu Scientific Library) is a C library for classical numerical analysis: special functions, linear algebra, non-linear equations, differential equations, the lot. The GSL was designed and implemented by a relatively small team of developers, who clearly know what they are doing — beyond the standard introductory textbooks treatment (as is evident from some specific design choices that address ugly but important real-world needs).

The API is wonderfully clear and consistent, the implementations are high quality, even the documentation is complete and finished. I find the GSL thoroughly enjoyable to use. (If you learned numerical analysis from “Numerical Recipes”³: this is the software that should have shipped with the book, but didn’t.)

The only problem is: the GSL is in C, and you need to be comfortable with C programming, including memory management and function pointers, if you want to use it. Bindings to scripting languages exist, but they are not part of the core project, and may not be as complete or mature as the GSL itself.

A.2.6 Recommendations

So, which to pick? No clear winner emerges, and every single program or environment has significant (not just superficial) drawbacks. But I can make some qualified recommendations:

- Matlab is the 800lb Gorilla of scientific software. As a commercially developed product, it also has a certain amount of “polish” that many open source alternatives lack. If you don’t have a preferred programming environment yet, *and* you can afford it or can get your employer to pay for it, then Matlab is probably the most comprehensive, most mature, and best supported all-purpose tool. Octave is a cheap way to get started and “try before you buy”.
- If you work with statisticians or have otherwise a need for formal statistical methods (tests, models), then R is a serious contender. It can also stand in as a scripting language for data manipulation if you don’t already have a favorite one yet. Since it is open source software, its financial cost to you is zero, but be prepared for a significant investment of time and effort before you start feeling comfortable and proficient.

³*Numerical Recipes 3rd Edition: The Art of Scientific Computing*. William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery. Cambridge University Press. 2007.

- If you are short on money and long on time and enthusiasm, you might want to give NumPy/SciPy a try. Be prepared for the need to work around defects or missing functionality as required.
- NumPy/SciPy, together with some of its associated graphics packages, is also a contender if you have a need for fancier, possibly interactive, graphics.
- If you have a need for serious numerical analysis *and* know C well, then the GSL is a mature, high-quality library.

I am well aware that this list of options does not cover all possibilities that may occur in practice!

A.3 Writing your own

Given the unsatisfying tool situation, it may seem tempting to write your own. There is nothing wrong with that — it can be very effective to write a piece of software specifically for *your* particular problem and application domain. It is much harder to write general purpose scientific software.

Just how much harder is usually under-appreciated. When P. J. Plauger worked on his reference implementation of the standard C library⁴, he found that he “spent about as much time writing and debugging the functions declared in `<math.h>` as [he] did all the rest of this library combined.” (!) He then went on to state his design goals for his implementation of those functions.

It is here that your ears should prick up: *design goals*? Why should a reference implementation need design goals, beyond faithfully and correctly representing the standard?

The reason is that scientific and numerical routines can fail in more ways than most people expect. For such routines, correctness is not so much a binary property, as a floating point value itself. Numerical routines have more complicated contracts than `strlen(char *)`.

My prime example for this kind of problem is the sine function. What could possibly go wrong with it? It is analytic everywhere, strictly bounded by $[-1, 1]$, perfectly smooth, no weird behavior anywhere. Yet, it is impossible to evaluate the sine accurately for sufficiently large values of x . The reason is that the sine sweeps out its entire range of values when x changes by as little as 2π . Today’s floating point values carry 16 digits of precision. Once x has become so large that all of these digits are required to represent the value of x to the left of the decimal point, we are no longer able to determine the location of x within the interval of length 2π — hence the “value” returned by $\sin(x)$ is basically random. (In practice, the quality of the results starts to degrade long before we reach this extreme regime.) There are two points to take away here: first, note how “correctness” is a relative quality, which can degrade smoothly,

⁴*The Standard C Library*. P. J. Plauger. Prentice-Hall. 1992.

depending on circumstances (that is: inputs). Additionally, you should register the sense of surprise, how a function, which in theory is perfectly harmless, can turn nasty in the harsh reality of a computer program!

Similar examples can be found all over and are not limited to function evaluations. In particular for iterative algorithms (and basically all numerical algorithms are iterative), one needs to monitor that all intermediate values are uncorrupted, even in cases where the final result is perfectly reasonable. (This warning applies to many matrix operations, for instance.)

The punchline is here that although it is often not hard to produce an implementation that works well for a limited set of input values and in a narrow application domain, it is very difficult to write routines that work equally well for all possible arguments. And it takes a lot of experience to anticipate all possible applications and provide built-in diagnostics for the likely failure modes. If at all possible, leave this work to specialists!

A.4 Further Reading

A.4.1 Matlab

- *Numerical Computing with MATLAB*. Cleve B. Moler. Revised Reprint, SIAM. 2008.

The literature on Matlab is vast. I mention this title because of its author: Cleve Moler, they guy who started it all.

A.4.2 R

- *A Beginner's Guide to R*. Alain F. Zuur, Elena N. Ieno, Erik H. W. G. Meesters. Springer Verlag. 2009.

Probably the most elementary introduction into the mechanics of R. A useful book to get started, but it won't carry you very far. Obviously very hastily produced.

- *R in a Nutshell*. Joseph Adler. O'Reilly. 2009.

This is the first book on R that is organized by the *task* that you want to perform, making it an invaluable resource in those situations where you know exactly what you want to do, but can't find the appropriate commands to tell R how to do it. The first two-thirds of the book are on data manipulation, programming, and graphics, with statistics making up the remainder.

- *Using R for Introductory Statistics*. John Verzani. Chapman & Hall/CRC. 2004.

Probably my favorite introductory text on how to perform basic statistical analysis using R.

A.4.3 NumPy/SciPy

There is no comprehensive book on NumPy/SciPy currently available that takes a user's perspective. (The "Guide to NumPy" by Travis Oliphant which can be found on the NumPy website is too concerned with implementation issues.) Some useful bits, together with an introduction to Python and some other libraries can be found in either of the following two books:

- *Python Scripting for Computational Science*. Hans Petter Langtangen. 3rd ed., Springer. 2009.
- *Beginning Python Visualization: Crafting Visual Transformation Scripts*. Shai Vaingast. Apress. 2009.

Appendix B

Results from Calculus

In this appendix, I collect some of the results from Calculus that I either need explicitly in the main part of the book, or that are conceptually sufficiently important when doing data analysis and mathematical modeling that you should at least be aware that they exist.

Obviously, this appendix cannot replace a class (or two) in beginning and intermediate calculus, and this is also not intended. Instead, I want to remind you of things that I expect you know already, and, more importantly, I want to present these results in a slightly different context than usual. Calculus is generally taught with an eye towards the theoretical development — it has to be, because the intent is to teach the whole body of knowledge which is the Calculus, therefore the theoretical development is most important. However, for applications, you need a different sort of tricks (based on the same fundamental techniques, of course), and it has been my experience that it takes *years* of experience to make out the tricks from the theory. In this appendix, I assume that you have seen the theory at least once, so I am just reminding you of it, but I want to emphasize those elementary techniques that are most useful in applications of the kind explained in the rest of this book.

This appendix is also intended as somewhat of a teaser: I have included some results that are particularly interesting, noteworthy, or fascinating, as an invitation for further study.

The last part of this appendix (quasi an appendix to the appendix) collects some material on basic operations, notation, and on reading equations — start there if you are getting confused.

A note for the mathematically fussy: this appendix quite intentionally eschews much mathematical sophistication. I know that many of my statements can either be made more general or more precise. But the way they are worded here is sufficient for what I want to achieve, and I wanted to avoid the obscurity that is the by-product of presenting mathematical statements in their most general form.

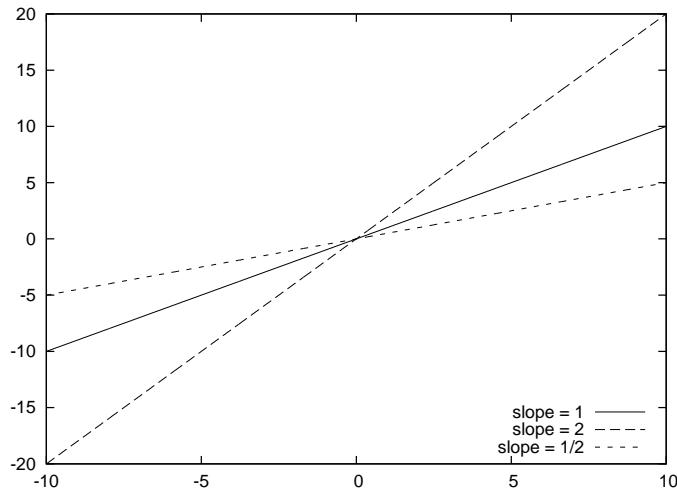


Figure B.1: TBD: linear.png

B.1 Common Functions

Functions are mappings, which map a real number into another real number: $f : \mathbb{R} \mapsto \mathbb{R}$. This mapping is always unique: every input value x is mapped to exactly one result value $f(x)$. (The converse is not true: many input values may be mapped to the same result, for example the mapping $f(x) = 0$, which maps *all* values to zero is a valid function.)

More complicated functions are often built up as combinations of simpler functions. The most important simple functions are powers, polynomials and rational functions, and trigonometric and exponential functions.

B.1.1 Powers

The simplest non-trivial function is the *linear* function:

$$f(x) = ax$$

The constant factor a is the *slope*: if x increases by one, then $f(x)$ increases by a . Figure B.1 shows linear functions with different slopes.

The next-simplest set of functions are simple powers:

$$f(x) = x^k$$

The power k can be greater or smaller than 1. The exponent can be positive or negative, and it can be an integer or a fraction. Figure B.2 shows graphs of some function with positive integer powers, and B.3 of functions with fractional powers.

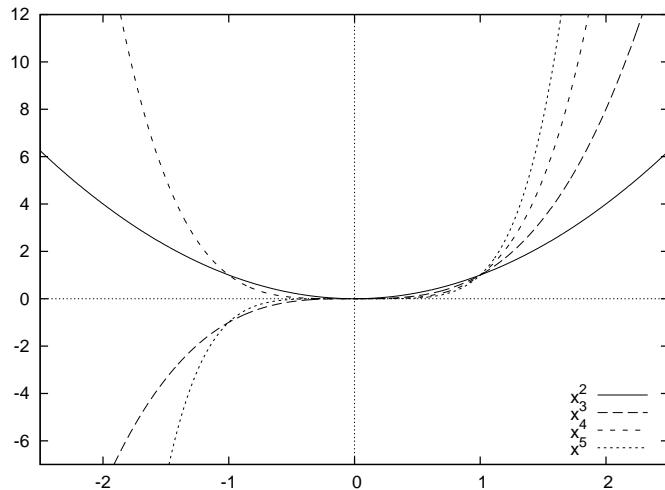


Figure B.2: TBD: powers.png

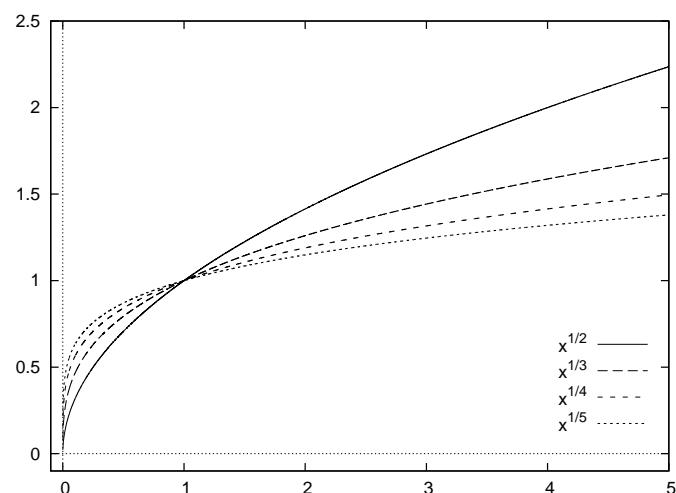


Figure B.3: TBD: fracpowers.png

Simple powers have some important properties:

- All simple powers go through the two points $(0, 0)$ and $(1, 1)$.
- The linear function $f(x) = x$ is a simple power with $k = 1$.
- The square-root function $f(x) = \sqrt{x}$ is a simple power with $k = 1/2$.
- Integer powers ($k = 1, 2, 3, \dots$) can be evaluated for negative x ; for fractional powers, we have to be more careful.

Powers obey the following laws:

$$\begin{aligned}x^n x^m &= x^{n+m} \\x^n x^{-m} &= \frac{x^n}{x^m} \\x^0 &= 1 \\x^1 &= x\end{aligned}$$

If the exponent is negative, it turns the expression into a *fraction*:

$$x^{-n} = \frac{1}{x^n}$$

Whenever we are dealing with fractions, we must remember that the denominator must not become zero. As the denominator of a fraction approaches zero, the value of the overall expression goes to infinity. We say: it *diverges* and the function has a *singularity* at the position where the denominator vanishes. Figure B.4 shows graphs of functions with negative powers. Note the divergence for $x = 0$.

B.1.2 Polynomials and Rational Functions

Polynomials are sums of integer powers together with constant coefficients:

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_2 x^2 + a_1 x + a_0$$

Polynomials are nice, because they are very easy to handle mathematically (after all, they are just sums of simple integer powers). Yet, more complicated functions can be approximated very well using polynomials. Polynomials therefore play a large role as approximations of more complicated functions.

All polynomials exhibit some “wiggles” and eventually diverge as x goes to plus or minus infinity (see figure B.5). The highest power occurring in a polynomial is known as the *degree* of the polynomial.

Rational functions are fractions, having polynomials in both the numerator and the denominator:

$$r(x) = \frac{p(x)}{q(x)} = \frac{a_n x^n + a_{n-1} x^{n-1} + \dots + a_2 x^2 + a_1 x + a_0}{b_m x^m + b_{m-1} x^{m-1} + \dots + b_2 x^2 + b_1 x + b_0}$$

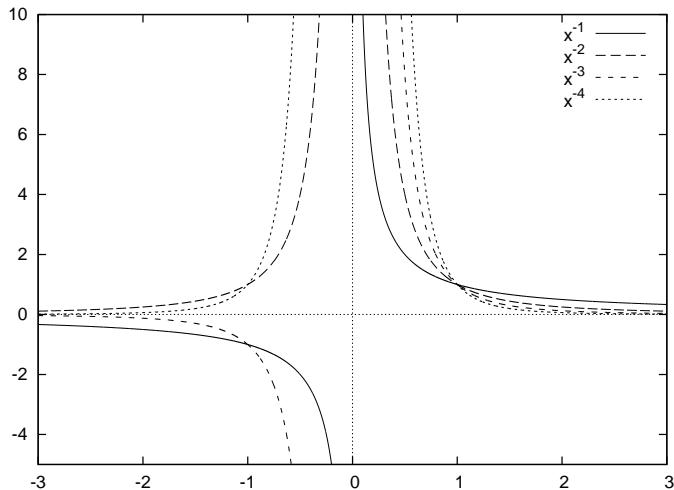


Figure B.4: TBD negpowers.png

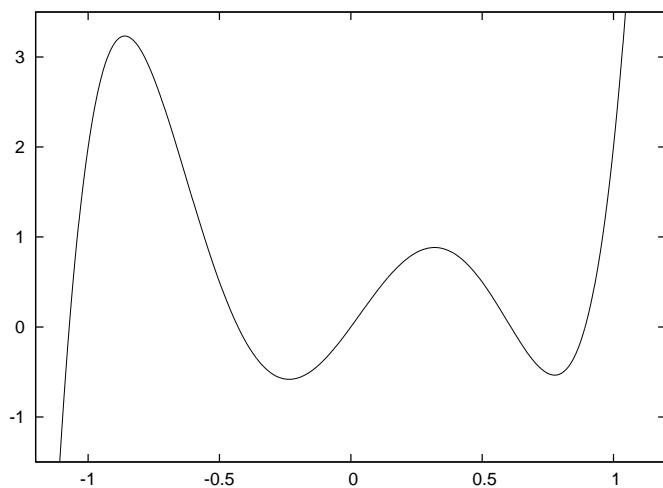


Figure B.5: TBD : Polynomial.png

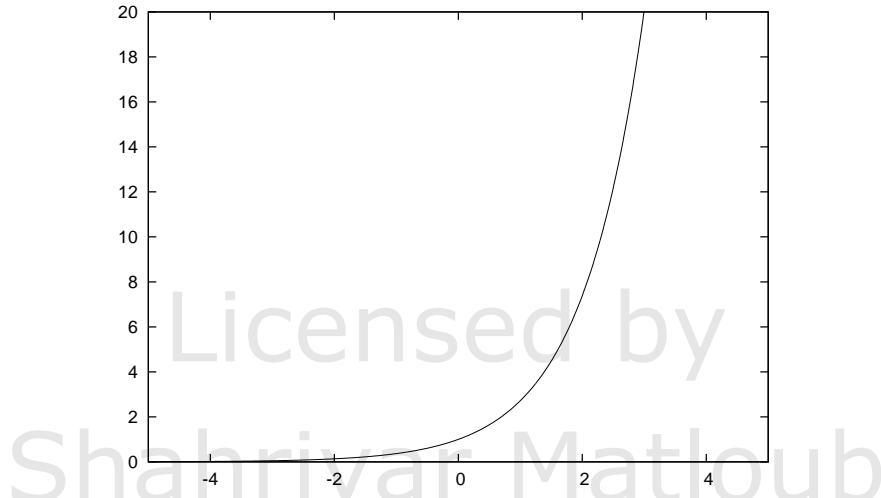


Figure B.6: TBD exponential.png

Although they may appear similarly harmless, rational functions are entirely more complicated beasts than polynomials. Whenever the denominator becomes zero, they blow up. The behavior as x goes to infinity depends on the relative size of the largest powers in numerator and denominator, respectively. Rational functions are *not* simple functions.

B.1.3 Exponential Function and Logarithm

Some functions cannot be expressed as polynomials (or fraction of polynomials) of finite degree. Such functions are known as *transcendental functions*. For our purposes, the most important ones are the exponential function $f(x) = e^x$, and its inverse, the logarithm.

A graph of the exponential function is shown in figure B.6. For positive argument, the exponential function grows *very* quickly, and for negative argument it decays equally quickly. The exponential function plays a huge role in growth and decay processes.

Some properties of the exponential function follow from the rules for powers:

$$e^x e^y = e^{x+y}$$

$$e^{-x} = \frac{1}{e^x}$$

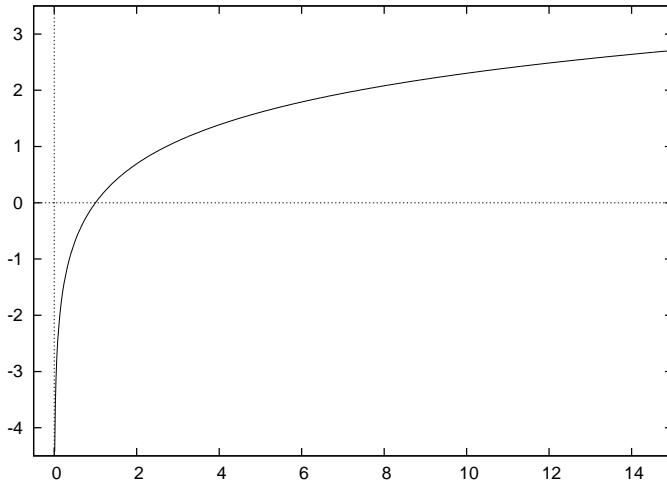


Figure B.7: TBD

The logarithm is the inverse of the exponential function, in other words:

$$\begin{aligned} y = e^x &\iff \log y = x \\ e^{\log(x)} = x &\quad \text{and} \quad \log(e^x) = x \end{aligned}$$

A plot of the logarithm is shown in figure B.7. The logarithm is only defined for strictly positive values of x , and it tends to negative infinity as x approaches zero. Going in the opposite direction, as x becomes large, the logarithm grows beyond all bounds, but it grows almost unbelievably slowly. For $x = 2$, we have $\log 2 = 0.69\dots$, for $x = 10$ we find $\log 10 = 2.30\dots$, but for $x = 1000$ and $x = 10^6$ we only have $\log 1000 = 6.91\dots$ and $\log 10^6 = 13.81\dots$, respectively. Yet the logarithm does not have an upper bound: it keeps on growing, but the rate of growth keeps getting smaller and smaller.

The logarithm has a number of basic properties:

$$\begin{aligned} \log(1) &= 0 \\ \log(xy) &= \log x + \log y \\ \log(x^k) &= k \log x \end{aligned}$$

As you can see, logarithms turn products into sums and powers into products. In other words, logarithms “simplify” expressions. This property was (and is!) used in numerical calculations: rather than multiplying two numbers (complicated!), you add their logarithms (easy — if you have a logarithm table or a slide rule!), and then exponentiate the result. This kind of calculational scheme is still relevant today, but not for the kinds of simple products

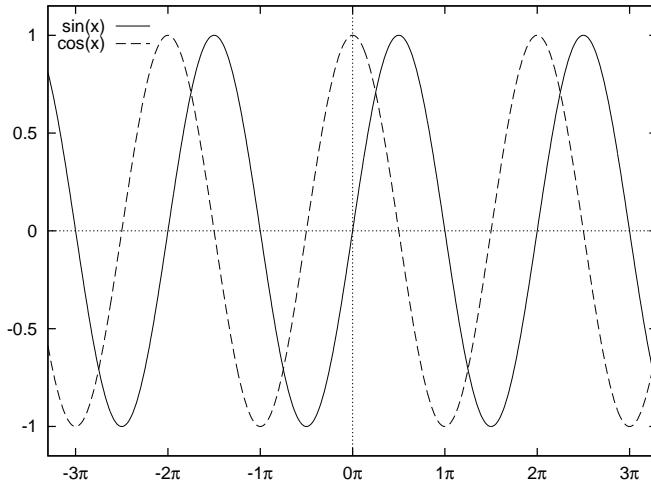


Figure B.8: TBD : trig.png

that previous generations would have done using slide rules. Instead, logarithmic multiplication can be required when dealing with products that would generate intermediate over- or under-flows, although the final result may be of entirely reasonable size. In particular in certain kinds of combinatorial and probabilistic problems you encounter the need to find the maximum of expressions such as $p^n(1-p)^k$, where $p < 1$ is a probability and n and k may be large numbers. Brute-force evaluation will underflow even for quite modest values of the exponents, but taking logarithms first will result in a numerically entirely reasonable expression.

B.1.4 Trigonometric Functions

The trigonometric functions describe oscillations of all kinds and therefore they play a central role in sciences and engineering. Like the exponential function, they are transcendental functions, meaning they cannot be written down as a polynomial of finite degree.

Figure B.8 shows graphs of the two most important trigonometric functions: $\sin(x)$ and $\cos(x)$. The cosine is equal to the sine, but shifted by $\pi/2$ ("90 degrees") to the left. We can see that both functions are *periodic*: they repeat themselves *exactly* after a period of length 2π ; in other words $\sin(x + 2\pi) = \sin(x)$ and $\cos(x + 2\pi) = \cos(x)$.

The length of the period is 2π , which you may recall is the *circumference of a circle with radius equal to one*. That should make total sense: $\sin(x)$ and $\cos(x)$ repeat themselves after advancing by 2π , and so does the circle: if you go around the circle once, you are back to where you started. This similarity between the

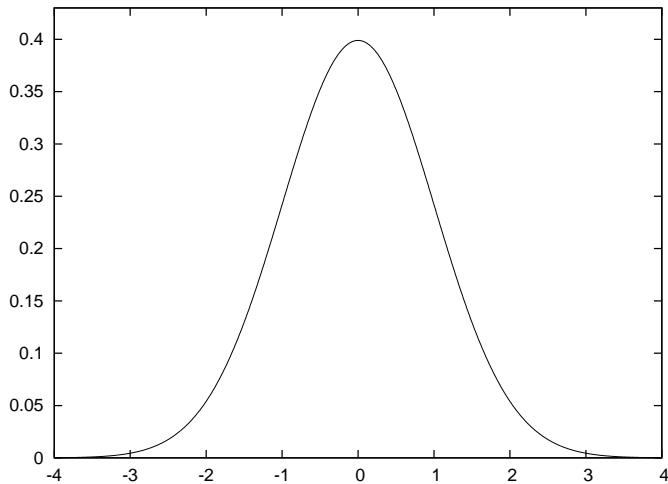


Figure B.9: TBD

trigonometric functions and the geometry of the circle is no accident, but this is not the place to explore it.

Besides their periodicity, the trigonometric functions obey a number of rules and properties ("trig identities"), of which only one is important enough to mention here:

$$\sin^2 x + \cos^2 x = 1 \quad \text{for all } x$$

Lastly, I should mention the tangent function, which is occasionally useful:

$$\tan x = \frac{\sin(x)}{\cos(x)}$$

B.1.5 Gaussian Function and Normal Distribution

The Gaussian function arises frequently and in many different contexts. It is given by the formula:

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

and its plot is shown in figure B.9. (This is the form in which the Gaussian should be memorized, with the factor 1/2 in the exponent and the factor 1/ $\sqrt{2\pi}$ up front: they ensure that the integral of the Gaussian over all x will be equal to one.)

Two applications of the Gaussian stand out. First of all, a very strong result from probability theory (the so-called *Central Limit Theorem*) says that — under rather weak assumptions — if we sum many random quantities, then this sum

will be distributed according to a Gaussian distribution. In particular if we take several samples from a population and calculate the mean for each sample, then the sample means will be distributed according to a Gaussian. Because of this, the Gaussian arises *all the time* in probability theory and statistics.

It is because of this connection that the Gaussian is often identified as “the” bell curve — quite incorrectly so, since there are many bell-shaped curves, often with drastically different properties. In fact, there are important cases where the Central Limit Theorem fails and the Gaussian is *not* a good way to describe the behavior of a random system (see the discussion of power-law distributions in chapter 9).

The other context in which the Gaussian arises frequently is as a *kernel*, that is as a strongly peaked, localized, yet very smooth function. Although it is greater than zero everywhere, it falls off to zero so quickly that almost the entire area underneath it is concentrated on the interval $-3 \leq x \leq 3$. It is this last property that makes the Gaussian so convenient to use as a kernel: although defined and non-zero everywhere, it can be multiplied against almost any function and retain only those values in the neighborhood of zero in a smooth fashion.

In statistical applications we are often interested in the area under certain parts of the curve, because that will provide the answer to questions such as: “What is the probability that the point lies between -1 and 1 ?” The anti-derivative of the Gaussian cannot be expressed in terms of elementary functions; instead it is defined through the integral directly:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-\frac{1}{2}t^2} dt$$

This function is known as the *Normal Distribution Function*. See figure B.10. As previously mentioned, the factor $1/\sqrt{2\pi}$ is a normalization constant to ensure that the area under the entire curve is 1.

Given the function $\Phi(x)$, a question such as the one given earlier can be answered easily: the area over the interval $[-1, 1]$ is simply $\Phi(1) - \Phi(-1)$.

B.1.6 Other Functions

There are a number of other functions which appear in applications often enough to be familiar with them, but which are a bit more exotic than the families of functions we have considered so far.

The *absolute value* function is defined as:

$$|a| = \begin{cases} a & \text{if } a \geq 0 \\ -a & \text{otherwise} \end{cases}$$

In other words, it is the positive (“absolute”) value of its argument. From a mathematical perspective, the absolute value is hard to work with, because of the need to treat the two possible cases separately, and because of the kink at

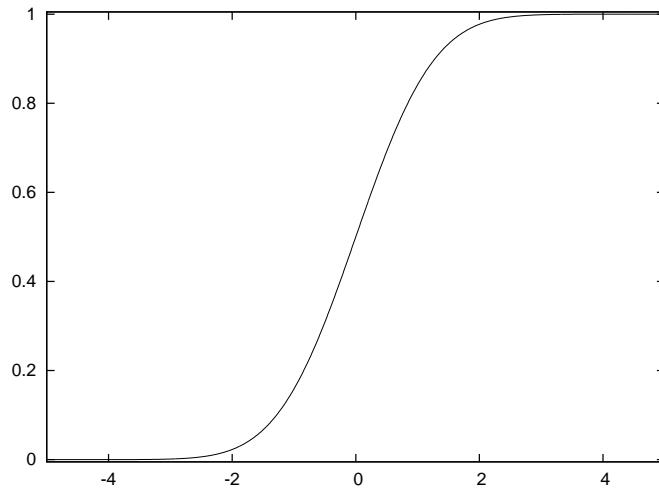


Figure B.10: TBD

$x = 0$, which poses difficulties when doing analytical work. If one wants to guarantee positive result from a function, one therefore often uses the square x^2 instead. The square relieves us of the need to worry about special cases explicitly, and it is smooth throughout. On the other hand, the square is relatively smaller than the absolute value for small values of x , but relatively larger for large values of x . Weight functions based on the square (which are used in “least-squares” methods, for instance) therefore tend to overemphasize outliers (see figure B.11).

Both the *hyperbolic tangent* $\tanh(x)$ (pronounced: tan-sh) and the *logistic function* are S-shaped or sigmoidal functions. (The latter function is the solution to the *logistic differential equation*, hence the name. The logistic differential equation is used to model constrained growth processes, such as bacteria competing for food or infection rates for contagious diseases.)

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$P(x) = \frac{1}{1 + e^{-x}}$$

Both functions are smooth approximations to a step function, differing mostly in the range of values they assume: the $\tanh(x)$ takes on values in the interval $[-1, 1]$, whereas the logistic function takes on only positive values, between 0 and 1 (see figure B.12). It is not hard to show that the two functions can be transformed into each other, in fact, the following relationship holds: $P(x) = (\tanh(x/2) + 1)/2$.

Either of these two functions is occasionally referred to as *the sigmoid func-*

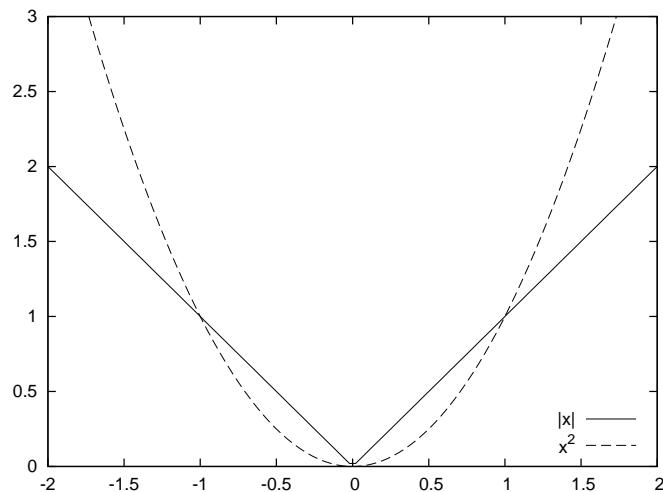


Figure B.11: TBD : absvalue

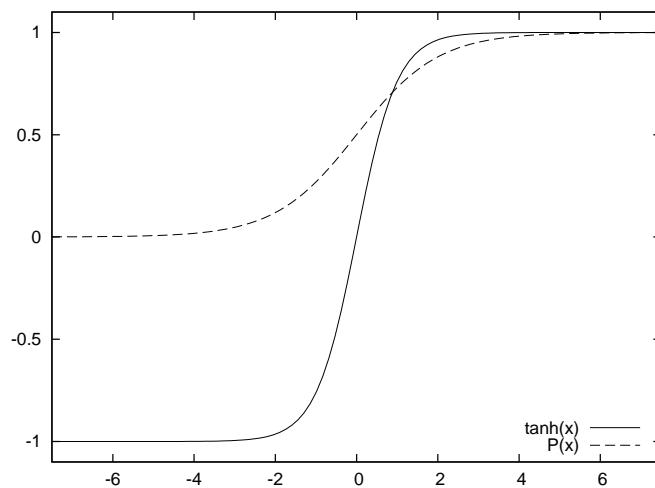


Figure B.12: TBD step.png

tion. That is incorrect: there are infinitely many functions that smoothly interpolate a step function. But among them, the two functions discussed here have the advantage that — although everywhere smooth — they basically consist of three straight lines: very flat as x goes to plus or minus infinity, and almost linear in the transition regime. The position and steepness of the transition can be changed through a standard variable transformation, for example $\tanh((x - m)/a)$ will have a transition at m with local slope $1/a$.

The last function to consider here is the *factorial*: $n!$. The factorial is only defined for positive integers, as follows:

$$\begin{aligned} 0! &= 1 \\ n! &= 1 \cdot 2 \cdot \dots \cdot (n-1) \cdot n \end{aligned}$$

The factorial plays an important role in combinatorial problems, since it is the number of ways that n distinguishable objects can be arranged. (To see this, imagine that you have to fill n boxes with n items. To fill the first box, you have n choices. To fill the second box, you have $n-1$ choices. And so on. The total number of arrangements or *permutations* is therefore $n \cdot (n-1) \cdots 1 = n!$.)

The factorial grows *very* quickly. It grows faster even than exponential; in fact it grows slightly faster than x^x . Because it grows so quickly, it is often convenient to work with its logarithm, and there is an important approximation for the logarithm of the factorial, known as *Stirling's Approximation*:

$$\log n! \approx n \log(n) - n \quad \text{for large } n$$

For the curious: it is possible to define a function that smoothly interpolates the factorial for all positive numbers (not just integers). It is known as the *Gamma Function* and is another example of a function defined through an integral:

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

The variable t in this expression is just a “dummy” variable of integration — it does not appear in the final result. You can see that the first term in the integral grows like a power, whereas the second falls exponentially, with the effect that the value of the integral is finite. Note that the limits of integration are fixed. The independent variable x enters the expression only as a parameter. Finally, it is easy to show that the Gamma function obeys the rule $n \Gamma(n) = \Gamma(n+1)$, which is the defining property of the factorial function.

We will not need the Gamma function in this book, but I find it interesting as an example how *integrals* can be used to define and construct new functions.

B.1.7 The Inverse of a Function

A function maps its argument to a result: given a value for x , we can find the corresponding value of $f(x)$. Occasionally, we want to turn this relation around and ask: given a value of $f(x)$, what is the corresponding value of x ?

That's what the *inverse function* does: if $f(x)$ is some function, then its inverse $f^{-1}(x)$ is defined as that function, which applied to $f(x)$ returns the original argument:

$$f^{-1}(f(x)) = x$$

Sometimes we can invert a function explicitly. For example, if $f(x) = x^2$, then the inverse function is the square root, because $\sqrt{x^2} = x$ (which is the definition of the inverse function). Similarly, for logarithm and exponential function: $\log(e^x) = x$.

In other cases, it may not be possible to find an explicit form for the inverse function. For example, we'll occasionally need the inverse of the Gaussian Distribution function $\Phi(x)$. However, no simple form for this function exists, so we write it symbolically as $\Phi^{-1}(x)$, which refers to the function for which $\Phi^{-1}(\Phi(x)) = x$ is true.

B.2 Calculus

Calculus proper deals with the consideration of limit processes: how does a sequence of values behave if we make infinitely many steps? The slope of a function and the area underneath a function are defined through such limit processes (the derivative and the integral, respectively).

Calculus allows us to make statements about properties of functions, and to develop approximations.

B.2.1 Derivatives

We have already met the slope as the rate of change of a linear function above. The same concept can be extended to other functions as well, only that for non-linear functions the slope itself will vary from place to place. We are therefore talking about the *local slope*.

Let's study the slope as the *rate of change* of a function some more, because this concept is very important whenever we try to interpolate or approximate some data with a smooth function. Figure B.13 shows the construction one uses to calculate the slope for a linear function. As x goes from a to b , the function goes from $f(a)$ to $f(b)$. The rate of change is the ratio of the change in $f(x)$ and the change in x :

$$\text{slope} = \frac{f(b) - f(a)}{b - a}$$

Make sure that you really understand this formula!

Now, let's apply this concept to a function which is non-linear. Because the slope of the curve varies from point to point, we cannot find the slope directly using the previous formula, but we can use the formula to *approximate* the local slope.

Figure B.14 demonstrates the concept. We fix two points on a curve and put a straight line through them. This line has a slope, which is $\frac{f(b) - f(a)}{b - a}$. Now,

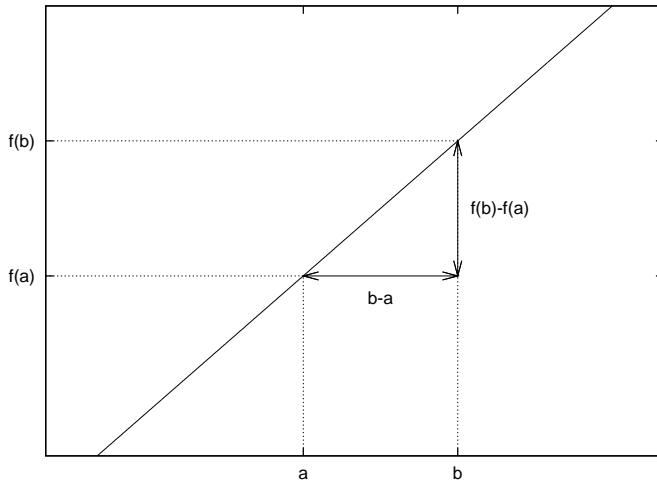


Figure B.13: TBD — linslope

this is only an approximation to the slope at point a . But we can improve the approximation by moving the second point b closer to a . If we let b go all the way to a , we end up with the slope *at* the point a exactly. This is called the *derivative*. (It is a central result of calculus that although numerator and denominator in $\frac{f(b)-f(a)}{b-a}$ separately go to zero in this process, the fraction itself goes to a well-defined value.)

The construction we performed above was done graphically and for a single point only, but in fact it can be carried out analytically in a fully general way. The process is sufficiently instructive that we shall study a simple example, namely finding a general rule for the derivative of the function $f(x) = x^2$, in detail. It will be useful to rewrite the interval $[a, b]$ as $[x, x + \epsilon]$. We can now go ahead and form the familiar ratio:

$$\begin{aligned}
 \frac{f(b) - f(a)}{b - a} &= \frac{f(x + \epsilon) - f(x)}{(x + \epsilon) - x} \\
 &= \frac{(x + \epsilon)^2 - x^2}{x + \epsilon - x} \\
 &= \frac{x^2 + 2x\epsilon + \epsilon^2 - x^2}{\epsilon} \\
 &= \frac{2x\epsilon + \epsilon^2}{\epsilon} \\
 &= 2x + \epsilon \\
 &\rightarrow 2x \quad \text{as } \epsilon \text{ goes to zero}
 \end{aligned}$$

In the second step, the terms not depending on ϵ cancel each other, in the third

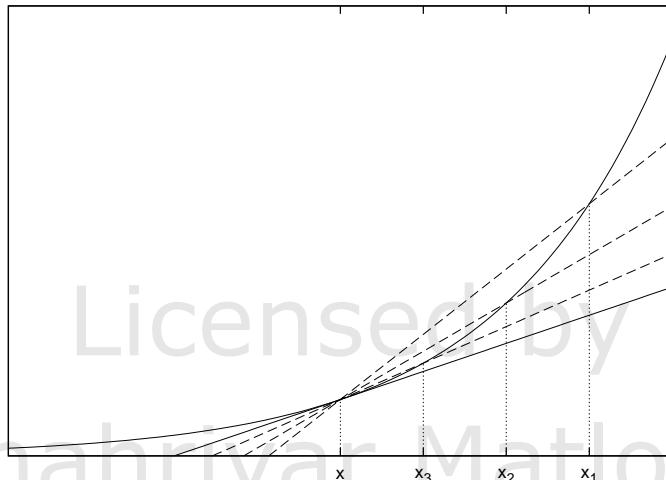


Figure B.14: TBD derivative.png

Function	Derivative	Integral
x^n	nx^{n-1}	$\frac{1}{n+1}x^{n+1}$
e^x	e^x	e^x
$\log x$	$1/x$	$x \log x - x$
$\sin x$	$\cos x$	$-\cos x$
$\cos x$	$-\sin x$	$\sin x$

Table B.1: TBD

step, we cancel an ϵ between the numerator and the denominator, leaving an expression which is perfectly harmless as ϵ goes to zero! The (harmless) result is the sought-for derivative of the function. Notice that the result is true for *any* x , which means that we have obtained an expression for the derivative of x^2 which is true for all x : the derivative of x^2 is $2x$. Always. Similar rules can be set up for other functions (you may try your hand at finding the rule for x^3 or even x^k for general k). Table B.1 lists a few of the most important ones.

There are two ways to indicate the derivative. A short form uses the prime, like this: $f'(x)$ is the derivative of $f(x)$. Another form uses the *differential operator* $\frac{d}{dx}$, which acts on the expression to its right. Using the latter, we can write:

$$\frac{d}{dx} x^2 = 2x$$

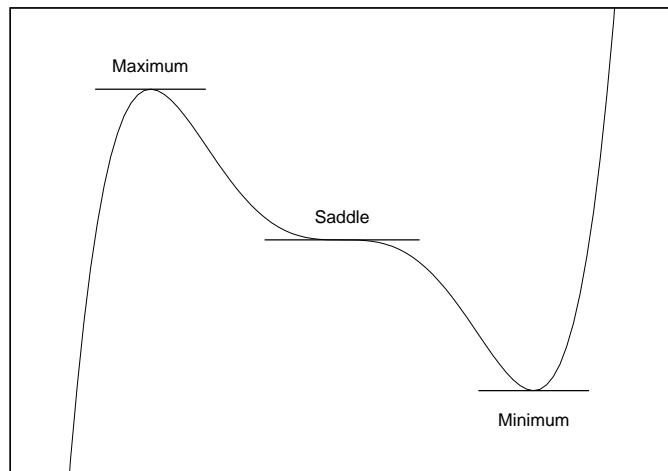


Figure B.15: TBD minmaxsaddle.png

B.2.2 Finding Minima and Maxima

When a smooth function reaches a local minimum or maximum, its slope at that point is zero. This is easy to see: as you approach a peak, you go uphill (positive slope), once over the top, you go downhill (negative slope) — hence you must have passed a point where you were going neither uphill nor downhill, in other words, where the slope was zero. (From a mathematically rigorous point of view, this is not quite as obvious as it may seem; you may want to check for “Rolle’s Theorem” in a calculus text.)

The opposite is also true: if the slope (and that means: the derivative) is zero somewhere, the function has either a minimum or a maximum at that position. (There is also a third possibility, namely that the function has a so-called saddle point there. In practice, this occurs less frequently.) Figure B.15 demonstrates all these cases.

We can therefore use derivatives to locate minima or maxima of a function: find the derivative of the function, then find the locations where the derivative is zero (its so-called *roots*). The roots are the locations of the extrema of the original function.

Extrema are important, because they are the solution to *optimization* problems. Whenever we want to find the “best” solution in some context, we are looking for an extremum: the lowest price, the longest duration, the greatest utilization, the highest efficiency. Hence, if we have a mathematical expression for the price, duration, utilization, or efficiency, we can take its derivative with respect to its parameters, set the derivative to zero, and solve for those values of the parameters that extremize our objective function.

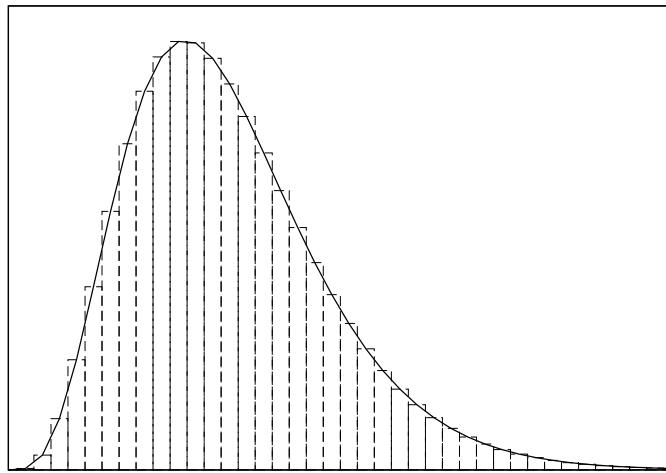


Figure B.16: TBD integral.png

B.2.3 Integrals

Derivatives find the local rate of change of a curve as the limit of a sequence of better and better approximations; integrals calculate the area underneath a curve by a similar method.

Figure B.16 demonstrates the process. We approximate the area underneath a curve with rectangular boxes. As we make the boxes narrower, the approximation gets better. In the limit of infinitely many, infinitely narrow boxes, we get the exact area under the curve.

Integrals are conceptually very simple, but analytically much harder than derivatives. It is always possible to find a closed form expression for the derivative of a function. This is not so for integrals in general, but for some simple functions an expression for the integral can be found. Some examples are included in table B.1.

Integrals are often denoted using upper-case letters, and there is a special symbol to indicate the “summing” of the area underneath a curve:

$$F(y) = \int f(x) dx$$

We can include the limits of the domain over which we want to integrate like this:

$$A = \int_a^b f(x) dx$$

Notice that this is a *number*, namely the area underneath the curve between $x = a$ and $x = b$, whereas the indefinite integral (without the limits) is a *function*, which can be evaluated at any point.

B.2.4 Limits, Sequences, Series

The central concept in all of calculus is the notion of a *limit*. The basic idea is as follows: we construct some process, which goes on forever and approximates some value ever more closely as the process goes on, but without reaching the limit in any finite number of steps, no matter how many. The important insight is to realize that, although the limit is never reached, we can nevertheless make statements about the limiting value. The derivative (as the limit of the difference ratio) and the integral (as the limit of the sum of approximating “boxes”) are examples that we have already encountered.

As simpler example, consider the numbers $1/1, 1/2, 1/3, 1/4, \dots$ or $1/n$ in general, with n going to infinity. Clearly, the numbers approach 0 ever more closely, yet for any finite n , the value of $1/n$ is always larger than zero. We call such an infinite, ordered set of numbers a *sequence* and zero is the limit of this particular sequence.

A *series* is a sum:

$$\begin{aligned}s_n &= \sum_{i=0}^n a_n \\ &= a_0 + a_1 + a_2 + a_3 + \cdots + a_n\end{aligned}$$

As long as the number of terms in the series is finite, there is no problem. But once we let the number of terms go to infinity, we need to ask whether the sum still converges to a finite value. We have already seen a case where it does: in our definition of the integral, we defined the integral as the value of the infinite sum of infinitely small boxes.

It may be surprising that an *infinite* sum can still sum up to a *finite* value. Yet, this can happen, provided the terms in the sum get smaller rapidly enough. Here is one example: if you sum up $1, 0.1, 0.01, 0.001, 0.0001, \dots$ and so on, you can see that the sum goes towards $1.1111\dots$ — and will never be larger than 1.2 (for example). Here is a more dramatic example: I have a piece of chocolate. I break it into two equal parts and give you one. Now I repeat the process with what I have left. And so on. Obviously, we can go on like this forever, since I always retain half of what I had before. Yet, you will never accumulate more chocolate than what I started out with!

An infinite series only converges to a finite value if the magnitude of the terms drops sufficiently quickly — if the terms do not get smaller fast enough, the series diverges (that means, its value is infinite). An important series which does *not* converge is the *harmonic series*:

$$\sum_{k=1}^{\infty} \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots = \infty$$

One can work out rigorous tests to determine whether a given series converges or not — for example by comparing the terms to those from a series that is known to converge: if the terms in the new series get smaller faster than in the converging series, the new series will also converge.

Finding the value of an infinite sum is often tricky, but there is one example which is rather straightforward. The solution involves a trick which is well-worth knowing. Consider the infinite *geometric series*:

$$s = \sum_{i=0}^{\infty} 1 + q + q^2 + q^3 + \dots \quad |q| < 1$$

Now, let's multiply this by q and add 1 to it:

$$\begin{aligned} qs + 1 &= q(1 + q + q^2 + q^3 + \dots) + 1 \\ &= q + q^2 + q^3 + q^4 + \dots + 1 \\ &= s \end{aligned}$$

To understand the last step, realize that the right-hand side equals our earlier definition of s . We can solve the resulting equation for s and obtain:

$$s = \frac{1}{1 - q}$$

This is a good trick that is worth remembering: if you can express an infinite series in terms of itself, you might end up with an equation that you can solve explicitly for the unknown value of the infinite series.

B.2.5 Power Series and Taylor Expansion

A particularly important kind of series contains consecutive powers of the variable x , multiplied by the constant coefficients a_i . Such series are called *power series*. The variable x can take on any value (it is a so-called “dummy variable”), and the sum of the series is therefore a *function of x* :

$$s(x) = \sum_{i=0}^n a_i x^i$$

If n is finite, there is only a finite number of terms in the series: in fact, the series is simply a polynomial (and, conversely, every polynomial is a finite power-series). But the number of terms can also be infinite (in which case we have to ask for what values of x does the series still converge). Infinite power-series are of great theoretical interest, since they are a (conceptually straightforward) generalization of polynomials, and hence represent the “simplest” non-elementary functions.

But power-series are also of the utmost *practical* importance. The reason is a remarkable theorem, known as *Taylor’s Theorem*. Taylor’s theorem states that any reasonably smooth function can be *expanded into a power series*. This process (and the resulting series) is known as the *Taylor expansion* of the function.

Taylor’s theorem gives an explicit construction for the coefficients in the series expansion:

$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \dots$$

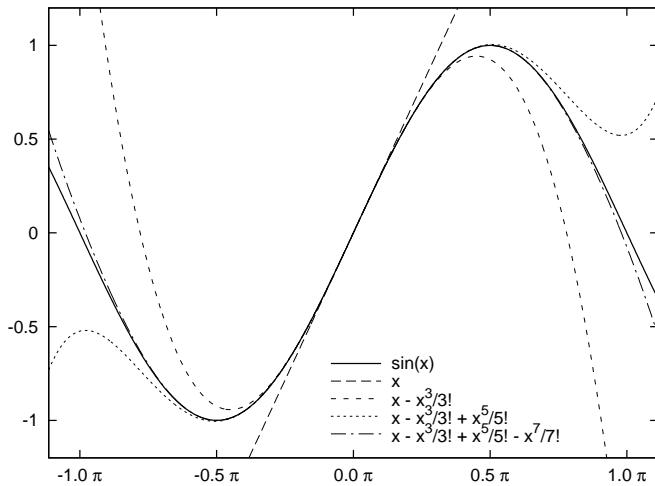


Figure B.17: The sine function $\sin(x)$ and its Taylor expansions around zero, truncated after retaining different numbers of terms. Note how the Taylor expansion remains a good approximation over greater ranges the more terms are retained.

In other words, the coefficient of the n -th term is the n -th derivative (evaluated at zero), divided by $n!$. The Taylor series converges for *all* x — the factorial in the denominator grows so quickly that convergence is guaranteed no matter how large x is.

The Taylor series is an exact representation of the function on the left hand side if we retain all (infinitely many) terms. But we can *truncate* the series after just a few terms and will obtain a good *local approximation* of the function in question. The more terms we keep, the larger will be the range over which the approximation is good. Figure B.17 shows for the sine function how the Taylor expansion improves as more and more terms are kept. Table B.2 shows the Taylor expansions for some functions we have encountered so far.

It is this last step that makes Taylor's theorem so useful from a practical point of view: it tells us that *we can approximate any smooth function locally by a polynomial*. And polynomials are always easy to work with — often much easier than the complicated functions that we started with.

One important practical point: the approximation provided by a truncated Taylor series is good only *locally*, near the point around which we expand. Why is that? Because in that case, x is small (that means: $x \ll 1$), and hence higher powers become negligible fast. The way Taylor series are usually represented, it is assumed that the expansion takes place around zero. If this is not the case, we need to remove or factor out some large quantity so that we are left with a "small parameter" in which to expand. As an example, imagine we want

Function	Taylor Expansion	Comment
e^x	$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$	all x
$\sin x$	$x - \frac{x^3}{3!} + \frac{x^5}{5!} + \dots$	all x
$\cos x$	$1 - \frac{x^2}{2!} + \frac{x^4}{4!} + \dots$	all x
$\log(1 + x)$	$x - \frac{x^2}{2} + \frac{x^3}{3} + \dots$	$-1 < x \leq 1$
$\sqrt{1 + x}$	$1 + \frac{x}{2} + \frac{x^2}{8} + \frac{x^3}{16} + \dots$	$ x \leq 1$
$1/(1 + x)$	$1 - x + x^2 - x^3 + \dots$	$ x < 1$

Table B.2: TBD - truncation valid if x is small

obtain an approximation to e^x for values of x near 10. If we were to expand in the usual fashion around zero, we would have to sum *many* terms before the approximation becomes good (the terms grow until $10^n < n!$, which means we need to keep over twenty terms). Instead, we proceed as follows: we write $\exp(x) = \exp(10 + \delta) = \exp(10) \exp(\delta) = \exp(10) (1 + \delta + \frac{\delta^2}{2} + \dots)$. In other words, we set it up in such a way that δ is small, so that we can expand $\exp(\delta)$ around zero as before.

Another important point to keep in mind is that the function must be smooth at the point around which we expand: it must not have a kink or other singularity there. This is the reason that the logarithm is usually expanded around 1 (not zero): recall that the logarithm diverges as x goes to zero.

B.3 Useful Tricks

B.3.1 The Binomial Theorem

Probably everyone has encountered the binomial formulas at some point:

$$(a + b)^2 = a^2 + 2ab + b^2$$

$$(a - b)^2 = a^2 - 2ab + b^2$$

The binomial theorem provides an extension of this result to higher powers. The theorem states that for an arbitrary integer power n , the expansion of the left-hand side can be written as:

$$(a + b)^n = \sum_{k=0}^n \binom{n}{k} a^{n-k} b^k$$

$$= \binom{n}{0} a^n b^0 + \binom{n}{1} a^{n-1} b^1 + \binom{n}{2} a^{n-2} b^2 + \dots + \binom{n}{n} a^0 b^n$$

This complicated looking expression involves the so-called *binomial coefficients*:

$$\binom{n}{k} = \frac{n!}{k! (n-k)!} \quad 0 \leq k \leq n$$

which are combinatorial factors that count the number of different ways one can choose k items from a set of n items, and in fact there is a close relationship between the binomial theorem and the binomial probability distribution.

However, as is the case for many exact results, the greatest practical use of the binomial theorem does not come from the exact solution, but from an approximate expression. Assume that $b < a$, so that $b/a < 1$. Now we can write:

$$\begin{aligned}(a+b)^n &= a^n \left(1 + \frac{b}{a}\right)^n \\ &\approx a^n \left(1 + n\frac{b}{a} + \frac{n(n-1)}{2} \left(\frac{b}{a}\right)^2 + \dots\right)\end{aligned}$$

where we neglected terms involving higher powers of b/a , which are small compared to the retained terms, since $b/a < 1$ by construction (so that higher powers of b/a , which involve multiplying a small number repeatedly by itself, become negligible very quickly).

In this form, the binomial theorem is very often useful as a way to generate approximate expansions. In particular the first-order approximation:

$$(1+x)^n \approx 1+nx$$

should be memorized.

B.3.2 The Linear Transformation

There is a quick trick, which is almost trivial, but useful enough to be committed to memory. Any variable can be transformed to a similar variable, but taking on only values from the interval $[0,1]$, by using the following linear transformation, where x_{\min} and x_{\max} are the minimum and maximum values that x can take on:

$$y = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

This transformation is frequently useful, for instance if we have two quantities and would like to compare how they develop over time. If the two quantities tend to have very different magnitudes, we need to reduce both of them to a common range of values. The transformation given above does exactly that.

If we want the transformed quantity to *fall* whenever the original quantity goes up, we can do this by writing:

$$\bar{y} = 1 - y = 1 - \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

We don't have to shift by x_{\min} and rescale by the original range $x_{\max} - x_{\min}$. Instead we can subtract any "typical" value and divide by any "typical" measure of the range. In statistical applications, for example, it is frequently

useful to subtract the mean μ and to divide by the standard deviation σ . The resulting quantity is also referred to as the *z-score*:

$$z = \frac{x - \mu}{\sigma}$$

Or you may want to subtract the median and divide by the inter-quartile range. The exact choice of parameters is not crucial and will depend on the specific application context. The important take-away here is that we can normalize any variable by:

- subtracting a typical value (shifting), and
- dividing by the typical range (re-scaling).

B.3.3 Dividing by Zero

Please remember that *you cannot divide by zero!* I am sure you know this — but it is surprisingly easy to forget (and then the computer reminds us with a terminal “Divide by Zero” error).

It is instructive to understand what happens if you try to divide by zero. Take some fixed number (for example 1) and divide it by a sequence of numbers that approach 0:

$$\begin{aligned} \frac{1}{10} &= 0.1 \\ \frac{1}{5} &= 0.2 \\ \frac{1}{1} &= 1.0 \\ \frac{1}{1/5} &= 5 \\ \frac{1}{1/10} &= 10 \\ \frac{1}{0} &= ??? \end{aligned}$$

In other words, as you divide a constant by numbers that *approach* zero, the result gets bigger and bigger. Finally, as you let the divisor go to zero, the result grows beyond all bounds: it diverges. Figure B.18 shows this graphically.

What you should take away from this exercise and figure B.18 is that you cannot replace $1/0$ by something else — for instance, it is *not* a smart move to replace $1/0$ by 0, “because both don’t really mean anything, anyway”. If $1/0$ you need to find a numeric value for $1/0$, then it should be something like “infinity”, but in practical applications, this is not a useful value to operate with.

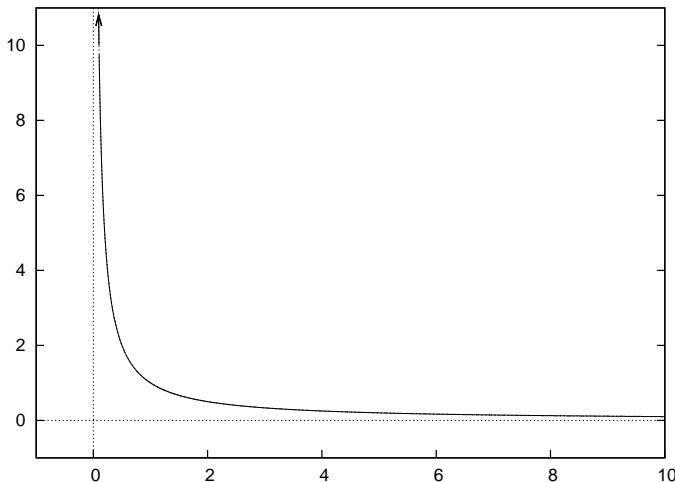


Figure B.18: TBD — divide by zero

Therefore, whenever you encounter a fraction $\frac{a}{b}$ of any kind, you **MUST** check whether the denominator can become zero and exclude these points from consideration.

Failing to do so is one of the most common sources of error, and what is worse, these errors are difficult to recover from — not just in implementations, but also conceptually. A typical example involves “relative errors”, where we divide the difference between the observed and the expected value by the expected value:

$$\text{relative error} = \frac{\text{observed} - \text{expected}}{\text{expected}}$$

What happens when on one day the expected value drops to zero? You are toast. There is no way to assign a meaningful value to the error in this case. (If the observed value is also zero, you can treat this as a special case and *define* the relative error to be zero in this situation, but if the observed value is not zero, then this definition is obviously inappropriate.)

These kinds of problems have an unpleasant ability to sneak up on you — for example, a quantity such as the relative error or the defect rate (a ratio: defects found divided by units produced) is a quantity commonly found in reports or dashboards. You don’t want your entire report to crash because no units were produced for some product on this day and therefore the denominator in one of your formulas became zero!

There are a couple of work-arounds, neither of them perfect. In the case of the defect rate, where you can be sure that the numerator becomes zero when the denominator does (because no defects can be found if no items were produced), you can add a small, positive number to the denominator to prevent it from ever becoming exactly zero. As long as this number is small compared to

the number of items typically produced in a day, it will not affect the reported defect rate significantly, and it relieves you from having to check for the $\frac{0}{0}$ special case explicitly. In the case of calculating a relative error, you might want to replace the numerator with the average of the expected and the observed values. This has the advantage that now the denominator can only become zero if the numerator does, too, which brings us back to the “defect rate” example just discussed. The problem with this method is that if no events are observed, but some number was expected, the relative error is reported as -2 (negative 200% instead of negative 100%), because of the factor $1/2$ in the denominator which comes from calculating the average in the denominator.

So, let me say it again: whenever you are dealing with fractions, you *must* consider the case of denominators becoming zero, and either rule them out or handle them explicitly.

B.4 Notation and Basic Math

This section is not intended as a comprehensive overview over mathematical notation or as your first introduction to mathematical formulas. Rather, it should serve as a general reminder of some basic facts and to clarify some conventions that I use in this book. (All my conventions are pretty standard — I have been careful not to use any symbols or conventions that are not generally used and understood.)

B.4.1 On Reading Formulas

A mathematical formula combines different components, called *terms*, by use of operators. The most basic operators are *plus* and *minus* (+ and -), and *times* and *divides* (\cdot and $/$). While plus and minus are always given explicitly, the multiplication operator is usually silent — in other words, if you see two terms next to each other, with nothing between them, they should be multiplied. The division operator can be written in two forms: $1/n$ or $\frac{1}{n}$, which mean exactly the same thing. The former is more convenient in text such as this; the latter is more clear for long, “display” equations. An expression such as $1/n + 1$ is ambiguous and should not be used, but if you encounter it, you should assume that it means $\frac{1}{n} + 1$, not $1/(n + 1)$ (which would be equivalent to $\frac{1}{n+1}$).

Multiplication and division have higher precedence than addition and subtraction, therefore $ab + c$ means that *first* you multiply a and b , and *then* you add c to the result. If you want to change the priority, you need to use parentheses: $a(b + c)$ means that *first* you add b and c , and *then* you multiply the result by a . Parentheses can either be round (...) or square [...], but their meaning is the same in either case.

Functions take one (or several) arguments and return a result. A function always has a *name* followed by the *arguments*. Very often the arguments are enclosed in parentheses: $f(x)$. Strictly speaking, this notation is ambiguous, because there is no way to tell whether an expression such as $f(a + b)$ means:

add a and b and *multiply* by f or: add a and b and pass the result to the *function* $f(\dots)$ to evaluate. However, it general it should not be too difficult to figure out the meaning from the context.

(There is a slightly more advanced way to look at this: you can think of f as an operator, similar to a differential operator like $\frac{d}{dx}$ or an integral operator like $\int dt$, and this operator is now being applied to the expression on the right of it. If f is a function, that means applying the function to the argument; if the operator is a differential operator, that means taking the derivative; and if f is merely a *number*, then applying it simply means multiplying the term on its right by it.)

A function may take more than one argument, for example the function $f(x, y, z)$ takes three arguments. Sometimes you may want to emphasize that not all of these arguments are equivalent: some are actual variables, whereas others are “parameters”, which are kept constant while the variables change. Consider: $f(x) = ax + b$. In this function, x is the “variable” (the quantity usually plotted along the horizontal axis), whereas a and b would be considered “parameters”. If we want to express that the function f does depend on the parameters as well as on the actual variable, we can do this by including the parameters in the list of arguments, like so: $f(x, a, b)$. To visually separate the parameters from the actual variable (or variables), one sometimes uses a semicolon: $f(x; a, b)$. There are no hard and fast rules when to use a semicolon as opposed to a comma — it is a convenience, that is sometimes used and other times not.

One more word on functions: several functions are regarded as “well-known” in mathematics (such as sine and cosine, or the exponential function, or the logarithms). The names of such well-known functions are always written in upright letter; whereas the names of functions in general are written in italic letters. (Variables are always written in italic letters). For well known functions, the parentheses around the arguments can be omitted if the argument is sufficiently simple. (This is again an example of the “operator” point of view mentioned earlier.) Here are some examples — note that sine and logarithm use upright letters, and I omitted the parentheses around the argument for the logarithm, because it consists of only a single term: $\sin(x + 1) + \log x - f(x)$. This has a different meaning than: $\sin(x + 1) + \log(x - f(x))$ (note the parentheses!).

B.4.2 Elementary Algebra

For numbers, the following is generally true:

$$a(b + c) = ab + ac$$

This is often applied in situations like the following, where we *factor out* a :

$$a + b = a(1 + b/a)$$

If a is much greater than b , we have now converted the original expression $a + b$ into another expression of the form:

$$\text{something large} \cdot (1 + \text{something small})$$

which makes it easy to see which terms matter and which can be neglected in an approximation scheme. (The small term in the parentheses is “small” compared to the 1 in parentheses and can therefore be treated as a perturbation.)

The three binomial formulas should be committed to memory:

$$\begin{aligned}(a + b)^2 &= a^2 + 2ab + b^2 \\ (a - b)^2 &= a^2 - 2ab + b^2 \\ (a + b)(a - b) &= a^2 - b^2\end{aligned}$$

Because the easiest things are often the most readily forgotten, let me just work out the first of these identities explicitly:

$$\begin{aligned}(a + b)^2 &= (a + b)(a + b) \\ &= a(a + b) + b(a + b) \\ &= a^2 + ab + ba + b^2 \\ &= a^2 + 2ab + b^2\end{aligned}$$

where I have made use of the fact that $ab = ba$.

B.4.3 Working with Fractions

Let me remind you of the basic rules for working with fractions. The expression on top is called the *numerator*, the one at the bottom is the *denominator*:

$$\frac{\text{numerator}}{\text{denominator}}$$

If you have a fraction, and you can factor out a common factor in both numerator and denominator, then this common factor can be canceled:

$$\frac{2 + 4x}{2 + 2 \sin(y)} = \frac{2(1 + 2x)}{2(1 + \sin(y))} = \frac{1 + 2x}{1 + \sin y}$$

To add two fractions, you have to bring them onto the same denominator, in an operation that is the opposite of canceling a common factor:

$$\frac{1}{a} + \frac{1}{b} = \frac{a}{ab} + \frac{b}{ab} = \frac{a + b}{ab}$$

A numeric example:

$$\frac{1}{2} + \frac{1}{3} = \frac{3}{6} + \frac{2}{6} = \frac{5}{6}$$

B.4.4 Sets, Sequences, and Series

A set is a grouping of elements in no particular order, whereas the elements in a sequence occur in a fixed order, one after the other.

The individual elements of sets and sequences are usually shown with subscripts, which denote the index the element in the set or its position in the sequence.

Sets are usually indicated through curly braces. The following expressions are equivalent:

$$\begin{aligned} & \{x_1, x_2, x_3, \dots, x_n\} \\ & \{x_i \mid i = 1, \dots, n\} \end{aligned}$$

For brevity, it is customary to suppress the range of the index if it can be understood from context — in other words, if it is clear that there are n elements in the set, I might simply write $\{x_i\}$.

One often wants to sum a finite or infinite sequence of numbers:

$$x_1 + x_2 + x_3 + \dots + x_n$$

Instead of writing out the terms explicitly as in the expression above, it is often useful to make use of the sum notation:

$$\sum_{i=1}^n x_i = x_1 + x_2 + x_3 + \dots + x_n$$

The meaning of the summation symbol should be clear from the example above: the variable which is used as index is written underneath the summation sign (here: i), followed by the lower limit (here: 1). The upper limit is written above the summation sign (here: n). As a shorthand, any one of these specifications can be omitted. For instance if it is clear from the context that the lower limit is 1 and the upper limit is n , then I might write the above expression as $\sum_i x_i$ or even simply as $\sum x_i$. In the last expression, it is understood that the sum runs over the index of the summands.

Lastly, I often find it useful to describe the terms I intend to sum over in words, rather than giving specific limits, like so:

$$\sum_{\text{all data points}} x_i$$

Some standard transformations involving the summation notation are used often. For example, there is often a need to shift indices. The following three expressions are equal, as you can easily see by writing out the terms of the sum explicitly in each case:

$$\sum_{i=0}^n x_i = \sum_{i=1}^{n+1} x_{i-1} = x_0 + \sum_{i=1}^n x_i$$

Keep in mind that the summation notation is just a shorthand for the explicit form given at the start of this section. If you get confused, you can always write out the terms explicitly to understand what is going on.

Lastly, we may take the upper limit of the sum to be infinity, so that the sum runs over infinitely many terms. (Infinite series play a fundamental roles in the theoretical development of mathematics, but all series you are going to encounter in applications are of course finite — feel free skip the rest of this section if you like.)

B.4.5 Special Symbols

There are several special symbols to describe the relationship between two expressions. I listed some of the most useful ones in table B.4.5.

Operator	Meaning
$= \neq$	equal to, not equal to
$< >$	less than, greater than
$\leq \geq$	less or equal than, greater or equal than
$\ll \gg$	much less than, much greater than
\propto	proportional to
\approx	approximately equal to
\sim	scales as

The last three might require a word of explanation. I say two quantities are *approximately equal* when they are equal, up to a “small” error. Put differently, the difference between the two quantities must be small compared to the quantities themselves: x and $1.1x$ are “approximately equal”: $x \approx 1.1x$ (because the difference, which is $0.1x$ is small compared to x).

A quantity is *proportional* to another one if they are equal, up to a constant factor, which has been omitted from the expression. Very often, this factor will have units associated with it. As an example, when we say “time is money”, what we really mean is:

$$\text{money} \propto \text{time}$$

where the omitted constant of proportionality is the hourly rate (which is required to fix the units: hours on the left, dollars on the right, so the hourly rate must have units of “dollars per hour” to make the equation dimensionally consistent).

We say that a quantity *scales as* some other quantity if we want to express how one quantity depends on another one, but in a very general way. For example, recall that the area of a circle is πr^2 , where r is the length of the radius, but that the area of a square is a^2 , where a is the length of the side of the square. We can now say that “the area *scales as* the square of the length”. This is a more general statement than saying that the area is proportional to the length: the latter implies that they are equal up to a constant factor, whereas the scaling behavior allows for more complicated dependencies. (In the present example, the

constant of proportionality depends on the *shape* of the figure, but the scaling behavior area $\sim \text{length}^2$ is true for all figures.)

In particular when evaluating the complexity of algorithms, there is another notation to express a very similar notion: the so-called *Big-O* notation. For example, the expression $\mathcal{O}(n^2)$ states that the complexity of an algorithm grows (“scales”) with the square of the number of elements in the input.

Occasionally, the interval notation is convenient:

$$[a, b] \quad \text{all } x, \text{ such that } a \leq x \leq b$$

For the purpose of this book we don’t need to worry about the distinction between open and closed intervals (that is intervals which don’t or do contain their endpoints).

The only other special symbol that I use occasionally is the familiar square root sign:

$$y = \sqrt{x}$$

which simply states that: $y^2 = x$.

B.4.6 The Greek Alphabet

Greek letters are being used all the time in mathematics and other sciences and should be committed to memory. (See table B.4.6).

B.5 On Learning Math

Coming soon

B.6 Further Reading

B.6.1 Calculus

- **The Hitchhiker’s Guide to Calculus by Michael Spivak; Mathematical Association Of America (1995)** If all the material in this appendix is really new to you, then this little (120 page) booklet provides a surprisingly complete, approachable, yet mathematically respectable introduction. Highly recommended for the curious and the confused.
- **Precalculus: A Prelude to Calculus by Sheldon Axler; Wiley (2008)** Axler’s book covers the basics: numbers, basic algebra, inequalities, coordinate systems, functions, including exponential, logarithmic, and trigonometric functions, but stops short of derivatives and integrals. If you want to brush up on foundational material, this is an excellent text.

Lower Case	Upper Case	Name
α	A	Alpha
β	B	Beta
γ	Γ	Gamma
δ	Δ	Delta
ϵ	E	Epsilon
ζ	Z	Zeta
η	H	Eta
θ	Θ	Theta
ι	I	Iota
κ	K	Kappa
λ	Λ	Lambda
μ	M	Mu
ν	N	Nu
ξ	Ξ	Xi
\o	O	Omicron
π	Π	Pi
ρ	R	Rho
σ	Σ	Sigma
τ	T	Tau
v	Y	Upsilon
ϕ	Φ	Phi
χ	X	Chi
ψ	Ψ	Psi
ω	Ω	Omega

- **Calculus by Michael Spivak; Publish or Perish (4th ed, 2008)** This is a comprehensive book on calculus. It concentrates exclusively on the clear development of the mathematical theory, and thereby avoids the confusion that often results from an oversupply of (more or less) artificial examples. The presentation is written for the reader who is relatively new to formal mathematical reasoning and tries to motivate the peculiar arguments required by formal mathematical manipulations. Rightly popular.
- **Yet Another Introduction to Analysis by Victor Bryant; Cambridge University Press (1990)** This short book is intended as a quick introduction for those readers who already possess passing familiarity with the topic and are comfortable with abstract operations.

B.6.2 Linear Algebra

B.6.3 For Further Study