Lecture Book
B.Sc Data Science I - Tutorial

D. T. McGuiness, Ph.D

Version: SS.2025

Topics on Artificial Intelligence & Machine Learning

Contents

I Probability & Statistics 3

1 Theory of Probability 5

1.1 Introduction . . . 5
1.2 Experiments & Outcomes . . . 9
1.3 Probability . . . 10
1.4 Permutations & Combinations . . . 15
1.4.1 Permutations . . . 15
1.4.2 Combinations . . . 16
1.4.3 Factorial Function . . . 17
1.4.4 Binomial Coefficients . . . 18
1.5 Random Variables and Probability Distributions . . . 19
1.5.1 Discrete Random Variables and Distributions . . . 20
1.5.2 Continuous Random Variables and Distributions . . . 21
1.6 Mean and Variance of a Distribution . . . 23
1.7 Binomial, Poisson, and Hyper-geometric Distributions . . . 26
1.7.1 Sampling with Replacement . . . 29
1.7.2 Sampling without Replacement: Hyper-geometric Distribution . . . 29
1.8 Normal Distribution . . . 30
1.8.1 Distribution Function . . . 31
1.8.2 Numeric Values . . . 31
1.8.3 Normal Approximation of the Binomial Distribution . . . 32
1.9 Distribution of Several Random Variables . . . 33
1.9.1 Discrete Two-Dimensional Distribution . . . 33
1.9.2 Continuous Two-Dimensional Distribution . . . 34
1.9.3 Marginal Distributions of a Discrete Distribution . . . 34
1.9.4 Marginal Distributions of a Continuous Distribution . . . 35
1.9.5 Independence of Random Variables . . . 36
1.9.6 Functions of Random Variables . . . 37
1.9.7 Addition of Means . . . 37
1.9.8 Addition of Variances . . . 38

2 Statistical Methods 41

2.1 Introduction . . . 41
2.2 Point Estimation of Parameters . . . 44
2.2.1 Maximum Likelihood Method . . . 44

2.3 Confidence Intervals . . . 47
2.4 Testing of Hypotheses and Making Decisions . . . 54
2.4.1 Errors in Tests . . . 56
2.5 Goodness of Fit . . . 60
2.6 Regression and Correlation . . . 63
2.6.1 Regression Analysis . . . 63
2.6.2 Confidence Intervals . . . 66
2.6.3 Correlation Analysis . . . 68
2.6.4 Test for the Correlation Coefficient . . . 70

II Scientific Python Essentials 73

3 Numpy 75
3.1 Introduction . . . 75
3.2 Why use Numpy . . . 77
3.3 An Introductionary Example - Calculating Grades . . . 78
3.4 Getting Into Shape: Array Shapes and Axes . . . 80
3.4.1 Understanding Shapes . . . 80
3.4.2 Understanding Axes . . . 81
3.4.3 Broadcasting . . . 82
3.5 Data Science Operations: Filter, Order, Aggregate . . . 85
3.5.1 Indexing . . . 85
3.6 Masking and Filtering . . . 87
3.7 Transposing, Sorting, and Concatenating . . . 93
3.8 Aggregating . . . 96
3.9 Practical Exercise Implementing Maclauring Series . . . 97
3.10 Optimizing Storage: Data Types . . . 99
3.11 Numerical Types . . . 100

4 Scipy 103
4.1 Introduction . . . 103
4.2 Special Functions . . . 105
4.2.1 The Bessel Function . . . 105
4.3 Integration . . . 107
4.3.1 Numerical Integration . . . 107
4.4 Ordinary Differential Equations . . . 110
4.4.1 Double Pendulum . . . 111
4.4.2 Damped Harmonic Oscillator . . . 113
4.5 Fourier Transform . . . 115
4.6 Linear Algebra . . . 118
4.6.1 Linear Equation Systems . . . 118
4.6.2 Eigenvalues and Eigenvectors . . . 119
4.7 Interpolation . . . 121

4.8 Statistics . . . 123

5 SymPy **125**
5.1 Symbolic Variables . . . 126
5.1.1 Complex Numbers . . . 127
5.1.2 Rational Numbers . . . 127
5.2 Numerical Evaluation . . . 129
5.3 Algebraic manipulations . . . 131
5.3.1 Expand and Factor . . . 131
5.3.2 Simplify . . . 131
5.3.3 Apart and Together . . . 132
5.4 Calculus . . . 133
5.4.1 Differentiation . . . 133
5.4.2 Integration . . . 133
5.4.3 Sums and Products . . . 134
5.4.4 Limits . . . 135
5.4.5 Series . . . 135
5.5 Linear Algebra . . . 137
5.6 Solving equations . . . 138

6 Matplotlib **139**
6.1 Introduction . . . 139
6.1.1 Figure size, aspect ratio and DPI . . . 144
6.1.2 Legends, labels and titles . . . 145
6.1.3 Formatting text: LaTeX, fontsize, font family . . . 147
6.1.4 Setting colors, linewidths, linetypes . . . 149
6.1.5 Control over axis appearance . . . 150
6.1.6 Placement of ticks and custom tick labels . . . 151
6.1.7 Axis number and axis label spacing . . . 152
6.1.8 Axis grid . . . 153
6.1.9 Axis Spines . . . 153
6.1.10 Twin Axes . . . 154
6.1.11 Axes where x and y is zero . . . 154
6.1.12 Other 2D plot styles . . . 155
6.1.13 Text annotation . . . 155
6.1.14 Figures with multiple subplots and insets . . . 156
6.2 Colormap and contour figures . . . 158
6.2.1 3D Figures . . . 160

7 pandas **165**
7.1 Introduction . . . 165
7.2 Series . . . 167
7.3 DataFrames . . . 171
7.3.1 Selecting Data by Position . . . 172

7.3.2 Select Data by Conditions . . . 173
7.4 Pandas for Panel Data . . . 177
7.4.1 Slicing and Reshaping Data . . . 177
7.5 Application: Long-Run Growth . . . 180
7.5.1 GDP per Capita . . . 183
7.5.2 GDP Growth . . . 190
7.5.3 Regional Analysis . . . 193

III Appendix **195**

A Tables **197**
A.1 Introduction . . . 197
A.2 Special Functions . . . 198
A.2.1 Bessel Function . . . 198
A.2.2 Error Function, Sine and Cosine Integrals . . . 199
A.3 Student-t Distribution . . . 200
A.4 Chi-Square Distribution . . . 202

Bibliography **205**

List of Figures

1.1 The histogram of the data given in **Exercise 1**. 7
1.2 A visual comparison of the Stirling formula and the actual values of the factorial function. 17
1.3 A visual representation of the Eq. (1.42). 22
1.4 The Poisson distribution with different mean (μ) values. 28
1.5 The poster child of probability and statistics, the normal distribution. 30
1.6 A visual representation between the relationship of PDF and CDF. 31
1.7 Many samples from a bivariate normal distribution. The marginal distributions are shown on the z-axis. The marginal distribution of X is also approximated by creating a histogram of the X coordinates without consideration of the Y coordinates. . . 34

2.1 The student-t distribution with different degrees of freedom m. 50
2.2 Chi-square distribution with different degrees of freedom. 52
2.3 The t-distribution used in Example 2.8. 55
2.4 Illustration of Type I and II errors in testing a hypothesis $\theta = \theta_0$ against an alternative $\theta = \theta_0$. 57
2.5 Samples with various values of the correlation coefficient r. 69

3.1 The logo of numpy. 75

4.1 The Bessel functions of the first kind (J_{ff}) with different orders. 105
4.2 A model of the double compound pendulum with physical description of its parameters. 111
4.3 The results of the double pendulum experiment. **(a)** The angles of θ_1 and θ_2 with respect to time **(b)** The spatial position of the pendulum as it swings around its pivot point. 113
4.4 In classical mechanics, a harmonic oscillator is a system that, when displaced from its equilibrium position, experiences a restoring force F proportional to the displacement x with k being a positive constant. 114
4.5 Fourier transform can be taught of the deconstruction of a signal to their sine and cosine components with a bias term which is generally known as DC term. The transform takes a signal in its time domain and transform it to their frequency components. 115
4.6 The FFT analysis of the solution to the under-damped oscillator from the previous section. 116
4.7 As the signal was symmetric, we only need to see the positive side of the spectrum. 117
4.8 An example of interpolation of a function using linear, and cubic interpolation. . . 122

4.9 he plots of three statistical properties for a discrete distribution. From top to bottom **(a)** Probability density function **(b)** Cumulative density function **(c)** Histogram. . . 123
4.10 The plots of three statistical properties for a continuous distribution. From top to bottom **(a)** Probability density function **(b)** Cumulative density function **(c)** Histogram. 124

6.1 . . . 141
6.2 . . . 142
6.3 . . . 143
6.4 . . . 143
6.5 . . . 144
6.6 . . . 145
6.7 . . . 147
6.8 . . . 148
6.9 . . . 149
6.10 . . . 159
6.11 . . . 159
6.12 . . . 160
6.13 . . . 161
6.14 . . . 162
6.15 . . . 162
6.16 . . . 163

7.1 A spreadsheet is a computer application for computation, organization, analysis and storage of data in tabular form and plays a vital role in many organisation be it engineering or economics. . . . 166
7.2 The evolution of the GDP of many nations with respect to year. . . . 180
7.3 GDP per Capita of United Kingdom. . . . 184
7.4 GDP per Capita of United Kingdom with continuation. . . . 185
7.5 GDP per capita of UK, US, China with important economic event imposed. . . . 187
7.6 GDP per capita of China with important economic event imposed. . . . 189
7.7 GDP per capita of US and UK with important economic event imposed. . . . 190
7.8 GDP in the early industrialization era. . . . 191
7.9 . . . 192
7.10 GDP in the modern era. . . . 193
7.11 Regional GDP per capita. . . . 194

List of Tables

2.1 Frequency table of the sample given in the question. 62
2.2 Dataset 66

3.1 The data types supported by Python and Numpy. 100

4.1 Scipy offers a wide variety of packages the user can work with. Above are some of them but the reader is recommended to look at other packages described within the documentation. 104

A.1 Particular values of the Bessel function 198
A.2 Particular values of Error function, sine and cosine integrals. 199
A.3 Values of z for given values of the distribution function F(z) with $m = 1 - 10$. . . 200
A.4 Values of z for given values of the distribution function F(z) with $m = 11 - 20$. . . 200
A.5 Values of z for given values of the distribution function F(z) with $m = 21 - 30$. . . 201
A.6 Values of z for given values of the distribution function F(z) with $m = 1 - 10$. . . 202
A.7 Values of z for given values of the distribution function F(z) with $m = 11 - 20$. . . 202
A.8 Values of z for given values of the distribution function F(z) with $m = 21 - 30$. . . 202

List of Examples

1.1 Recording Data . . . 6
1.2 Leaf Plots . . . 6
1.3 Histogram . . . 6
1.4 Empirical Rule Outliers and z-Score . . . 8
1.5 Sample Spaces of Random Experiments & Events . . . 9
1.6 Fair Die . . . 10
1.7 Coin Tossing . . . 11
1.8 Mutually Exclusive Events . . . 12
1.9 Union of Arbitrary Events . . . 12
1.10 Multiplication Rule . . . 13
1.11 Sampling w/o Replacement . . . 14
1.12 An Encrypted Message . . . 16
1.13 Sampling Light-bulbs . . . 17
1.14 Waiting Time Problem . . . 21
1.15 Continuous Distribution . . . 22
1.16 Mean and Variance . . . 23
1.17 Binomial Distribution . . . 27
1.18 Poisson Distribution . . . 28
1.19 The Parking Problem . . . 28
1.20 Marginal Distributions of a Discrete Two-Dimensional Random Variable . . . 35
1.21 Independence and Dependence . . . 36

2.1 Generating Random Numbers . . . 42
2.2 Maximum Likelihood of Gaussian Distribution . . . 45
2.3 Estimating Poisson Parameters . . . 46
2.4 Confidence Interval for mean with known variance in Normal Distribution . . . 48
2.5 Sample Size Needed for a Confidence Interval of Prescribed Length . . . 48
2.6 Confidence Interval for Mean of Normal Distribution with Unknown Variance . . . 50
2.7 Confidence Interval for the Variance of the Normal Distribution . . . 51
2.8 Test of a Hypothesis . . . 54
2.9 Test for the Mean of the Normal Distribution with Known Variance . . . 58
2.10 Comparison of the Means of Two Normal Distributions . . . 59
2.11 Test of Normality . . . 61
2.12 Regression Line . . . 66

2.13 Confidence Interval for the Regression Coefficient 67
2.14 Uncorrelated but Dependent Random Variables 70
2.15 Test for the Correlation Coefficient 70

3.1 A Simple Filter Exercise 88
3.2 Creating a Numpy Array 90
3.3 An Identity Matrix 90
3.4 A Random Value 91
3.5 A Random Array 91
3.6 Sum of an Array 95
3.7 Find the Missing Values 100
3.8 Array Value Parity 101

5.1 Integration 138

7.1 Series - An Introduction 167
7.2 Sorting a Series 168
7.3 DataFrame - An Introduction 172

List of Theorems

1.1 First Definition of Probability . . . 10
1.2 General Definition of Probability . . . 11
1.3 Complementation Rule . . . 11
1.4 Addition Rule for Mutually Exclusive Events . . . 12
1.5 Addition Rule for Arbitrary Events . . . 12
1.6 Multiplication Rule . . . 13
1.7 Permutations . . . 15
1.8 Permutations . . . 16
1.9 Combinations . . . 17
1.10 Random Variable . . . 19
1.11 Mean of a Symmetric Distribution . . . 24
1.12 Transformation of Mean and Variance . . . 24
1.13 Relationship between PDF and CDF . . . 31
1.14 Normal Probabilities for Intervals . . . 31
1.15 Limit Theorem of De Moivre and Laplace . . . 32
1.16 Addition of Means . . . 38
1.17 Multiplication of Means . . . 38
1.18 Addition of Variances . . . 39

2.1 Sum of Independent Normal Random Variables . . . 48
2.2 Student's t-Distribution . . . 50
2.3 Chi-Square Distribution . . . 51
2.4 Central Limit Theorem . . . 52
2.5 Least Square Principle . . . 64
2.6 Assumption A1 . . . 64
2.7 Assumption A2 . . . 66
2.8 Assumption A3 . . . 67
2.9 Sample Correlation Coefficient . . . 68
2.10 Correlation Coefficient . . . 69
2.11 Independence and Relation to Normal Distribution . . . 69

Part I

Probability & Statistics

Chapter 1

Theory of Probability

Table of Contents

1.1 Introduction 5
1.2 Experiments & Outcomes 9
1.3 Probability 10
1.4 Permutations & Combinations 15
1.5 Random Variables and Probability Distributions 19
1.6 Mean and Variance of a Distribution 23
1.7 Binomial, Poisson, and Hyper-geometric Distributions 26
1.8 Normal Distribution 30
1.9 Distribution of Several Random Variables 33

1.1 Introduction

When the data we are working are influenced by "chance", by factors whose effect we cannot predict exactly[1], we have to rely on **probability theory**. The application of this theory nowadays appears in numerous fields such as from studying a game of cards to the global financial market and allow us to model processes of chance called random experiments.

[1] This could be weather data, stock prices, life spans or ties, etc.

In such an experiment we observe a random variable X, that is, a function whose values in a trial[2] occur "by chance" according to a probability distribution which gives the individual probabilities, which possible values of X may occur in the long run.

[2] a performance of an experiment.

i.e., each of the six faces of a die should occur with the same probability, 1/6.

Or we may simultaneously observe more than one random variable, for instance, height and weight of persons or hardness and tensile strength of steel. But enough about spoiling all the fun and let's begin with looking at data.

Representing Data

Data can be represented numerically or graphically in different ways

> i.e., a news website may contain tables of stock prices and currency exchange rates, curves or bar charts illustrating economical or political developments, or pie charts showing how inflation is calculated.

And there are numerous other representations of data for special purposes. In this section, we will discuss the use of standard representations of data in statistics[3].

[3] There are various software dedicated to analyse and visualise statistical data. Some of these include: R, a statistical programming language, Python, MATLAB, ...

Exercise 1.1: Recording Data

Sample values, such as observations and measurements, should be recorded in the order in which they occur. Sorting, that is, ordering the sample values by size, is done as a first step of investigating properties of the sample and graphing it.
As an example let's look at super alloys.
Super alloys is a collective name for alloys used in jet engines and rocket motors, requiring high temperature (typically 1000° C), high strength, and excellent resistance to oxidation.
Thirty (30) specimens of Hastelloy C (nickel-based steel, investment cast) had the tensile strength (in 1000 lb>sq in.), recorded in the order obtained and rounded to integer values.

$$\begin{array}{cccccccccccccc} 89 & 77 & 88 & 91 & 88 & 93 & 99 & 79 & 87 & 84 & 86 & 82 & 88 & 89 & 78 \\ 90 & 91 & 81 & 90 & 83 & 83 & 92 & 87 & 89 & 86 & 89 & 81 & 87 & 84 & 89 \end{array} \tag{1.1}$$

Of course depending on the need the data needs to be sorted which is shown below:

$$\begin{array}{cccccccccccccc} 77 & 78 & 79 & 81 & 81 & 82 & 83 & 83 & 84 & 84 & 86 & 86 & 87 & 87 & 87 \\ 88 & 88 & 88 & 89 & 89 & 89 & 89 & 89 & 90 & 90 & 91 & 91 & 92 & 93 & 99 \end{array}$$

Graphic Representation of Data

Let's now use the data we have seen in Example 1 and see the methods we can use for graphic representations.

Exercise 1.2: Leaf Plots

One of the simplest yet most useful representations of data [1]. For Eq. (1.1) it is shown in **Table ??**.
The numbers in Eq. (1.1) range from 78 to 99; which you can also see this in the sorted list. To visualise this data feature, we divide these numbers into five (5) groups:

75-79, 80-84, 85-89, 90-94, 95-99.

The integers in the tens position of the groups are 7, 8, 8, 9, 9. These form the stem which can be seen in **Table ??**. The first leaf is 789, representing 77, 78, 79. The second leaf is 1123344, representing 81, 81, 82, 83, 83, 84, 84. And so on. The number of times a value occurs is called its **absolute frequency**.
Therefore in this example, 78 has absolute frequency 1, the value 89 has absolute frequency 5, etc. ■

Exercise 1.3: Histogram

For large sets of data, histograms are better in displaying the distribution of data than stem-and-leaf plots. The principle is explained in **Fig.** 1.1.

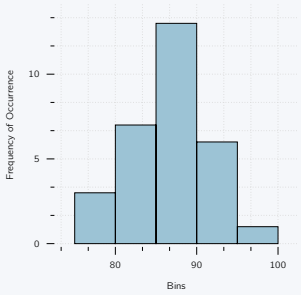


Figure 1.1: The histogram of the data given in **Exercise 1**.

The bases of the rectangles in seen in **Fig.** 1.1 are the x-intervals[4] where there rage is:

$$74.5-79.5, \quad 79.5-84.5, \quad 84.5-89.5,$$
$$89.5-94.5, \quad 94.5-99.5,$$

whose midpoints, known as class marks, are

$$x = 77, 82, 87, 92, 97,$$

respectively. The height of a rectangle with class mark x is the relative class frequency $f_{\text{rel}}(x)$, defined as the number of data values in that class interval, divided by n (= 30 in our case). Hence the areas of the rectangles are proportional to these relative frequencies,

$$0.10,\ 0.23,\ 0.43,\ 0.17,\ 0.07,$$

so that histograms give a good impression of the distribution of data.

[4] known as class intervals.

Mean, Standard Deviation, and Variance

Medians and quartiles are easily obtained by ordering and counting[5].

[5] This can be done without the need of calculators.

> However this method does not give full information on data as you can change data values to some extent without changing the median.

The average size of the data values can be measured in a more refined way by the mean:

$$\overline{x} = \frac{1}{n}\sum_{j=1}^{n} x_j = \frac{1}{n}\left(x_1 + x_2 + \cdots + x_n\right). \tag{1.2}$$

This is the arithmetic mean of the data values, obtained by taking their sum and dividing by the data size (n). Therefore the arithmetic mean for Eq. (1.1) is:

$$\overline{x} = \frac{1}{30}(89 + 77 + \cdots + 89) = \frac{260}{3} \approx 86.7 \quad \blacksquare$$

As we can see every data value contributes, and changing one of them will change the mean. Similarly, the spread[6] of the data values can be measured in a more refined way by the **standard deviation** s or by its square, the variance[7]

$$s^2 = \frac{1}{n-1}\sum_{j=1}^{n}\left(x_j - \overline{x}\right)^2 = \frac{1}{n-1}\left[\left(x_1 - \overline{x}\right)^2 + \cdots + \left(x_n - \overline{x}\right)^2\right] \tag{1.3}$$

Therefore, to obtain the variance of the data, take the difference (i.e., $x_j - \overline{x}$) of each data value from the mean, square it, take the sum of these n squares, and divide it by $n-1$.

[6] also known as variability.

[7] The symbol for variance is interesting as each domain have their own definition, as s^2, σ^2 and Var() are all acceptable symbols.

To get the standard deviation s, take the square root of s^2.

[8] which we calculated previously

Returning back to our super alloy example, using $\overline{x} = 260/3$[8], we get for the data given in Eq. (1.1) the variance:

$$s^2 = \frac{1}{29}\left[\left(89 - \frac{260}{3}\right)^2 + \left(77 - \frac{260}{3}\right)^2 + \cdots + \left(89 - \frac{260}{3}\right)^2\right] = \frac{2006}{87} \approx 23.06 \quad \blacksquare$$

Therefore, the standard deviation is calculated to be:

$$s = \sqrt{2006/87} \approx 4.802$$

The standard deviation has the same dimension as the data values, which is an advantage, whereas, the variance is preferable to the standard deviation in developing statistical methods.

Empirical Rule

For any round-shaped symmetric distribution of data the intervals:

$$\overline{x} \pm s, \quad \overline{x} \pm 2s, \quad \overline{x} \pm 3s, \qquad \text{contain about} \qquad 68\%, \quad 95\%, \quad 99.7\%.$$

respectively, of the data points. This information is quite useful in doing quick calculation of statistical properties such as the quality of production which will be the focus in Chapter 2.

Exercise 1.4: Empirical Rule Outliers and z-Score

For the data set given in Example 1.1, with $\overline{x} = 86.7$ and $s = 4.8$, the three (3) intervals in the Rule are:

$$81.9 \le x \le 91.5, \quad 77.1 \le x \le 96.3, \quad 72.3 \le x \le 101.1$$

and contain 73% (22 values remain, 5 are too small, and 5 too large), 93% (28 values, 1 too small, and 1 too large), and 100%, respectively.
If we reduce the sample by omitting the outlier value of 99, mean and standard deviation reduce to $\overline{x}_{\text{red}} = 86.2$, and $s_{\text{red}} = 4.3$, approximately, and the percentage values become 67% (5 and 5 values outside), 93% (1 and 1 outside), and 100%.
Finally, the relative position of a value x in a set of mean $\overline{x}$ and standard deviation s can be measured by the **z-score**:

$$z(s) = \frac{x - \overline{x}}{s}$$

This is the distance of x from the mean $\overline{x}$ measured in multiples of s. For instance:

$$z(s) = \frac{(83 - 86.7)}{4.8} = -0.77$$

This is negative because 83 lies below the mean. By the empirical rule, the extreme z-values are about -3 and 3. $\blacksquare$

1.2 Experiments & Outcomes

Now we have the basis covered, it is time to look at probability theory[9]. This theory has the purpose of providing mathematical models of situations affected or even governed by change effects, for instance, in weather forecasting, life insurance, quality of technical products (computers, batteries, steel sheets, etc.), traffic problems, and, of course, games of chance with cards or dice, and the accuracy of these models can be tested by suitable observations or experiments.

[9] Sometimes known as probability calculus.

Let's start by defining some standard terms:

experiment A process of measurement or observation, in a laboratory, in a factory, ...

randomness Situation where absolute prediction is not possible.

trial A single performance of an experiment

outcome The result of a trial[10]

[10] also known as sample point.

sample space Defined as S, is the set of all possible outcomes of an experiment.

Exercise 1.5: Sample Spaces of Random Experiments & Events

- Inspecting a lightbulb | S = {Defective, Non-defective} .
- Rolling a die | S = {1, 2, 3, 4, 5, 6}

  events are
  - A = 1, 3, 5 ("Odd number")
  - B = 2, 4, 6 ("Even number"), etc.
- Counting daily traffic accidents in Vienna | S = {the integers in some interval}.

1.3 Probability

The probability of an event A in an experiment is to measure **how frequently** A is roughly to occur if we make many trials. If we flip a coin, then heads H and tails T will appear about equally[11] often.

[11] on the condition, the measurements are done for a long time.

> we say that H and T are **"equally likely."**

Similarly, for a regularly shaped die of homogeneous material[12] each of the six (6) outcomes $1, \cdots, 6$ will be equally likely. These are examples of experiments in which the sample space S consists of finitely many outcomes (points) that for reasons of some symmetry can be regarded as equally likely.

[12] called a fair dice

Let's formulate this in a theory.

Theory 1.1: First Definition of Probability

If the sample space S of an experiment consists of **finitely** many outcomes (points) being equally likely, the probability $P(A)$ of an event A is defined to be:

$$P(A) = \frac{\text{Number of points in A}}{\text{Number of points in S}}.$$

From this definition it follows immediately, in particular, the probability of all events occurring in the sample space S is:

$$P(S) = 1.$$

Exercise 1.6: Fair Die

In rolling a fair die once:

1. What is the probability $P(A)$ of A of obtaining a 5 or a 6?
2. The probability of B: "Even number"?

Solution

The six outcomes are equally likely, so that each has probability 1/6. Therefore:

$$P(A) = \frac{2}{6} = \frac{1}{3} \quad \text{and} \quad P(B) = \frac{3}{6} = \frac{1}{2} \quad \blacksquare$$

The above theory takes care of many games as well as some practical applications, but not of all experiments, as in many problems we do not have finitely many equally likely outcomes. To arrive at a more general definition of probability, we regard probability as the counterpart of **relative frequency**:

$$f_{\text{rel}}(A) = \frac{f(A)}{n} = \frac{\text{Number of times A occurs}}{\text{Number of trials}} \tag{1.4}$$

Now if A did not occur, then $f(A) = 0$. If A always occurred, then $f(A) = n$. These are of course extreme cases. Division by n gives:

$$0 \leq f_{\text{rel}}(A) \leq 1 \tag{1.5}$$

In particular, for $A = S$ we have $f(S) = n$ as S always occurs[13]. Division by n gives:

[13] meaning that some event always occurs

$$f_{\text{rel}}(S) = 1 \tag{1.6}$$

Finally, if A and B are mutually exclusive, they cannot occur together. Therefore the absolute frequency of their union A = B must equal the sum of the absolute frequencies of A and B. Division

by n gives the same relation for the relative frequencies:

$$f_{\text{rel}}(A \cup B) = f_{\text{rel}}(A) + f_{\text{rel}}(B) \tag{1.7}$$

We can now extend the definition of probability to experiments in which equally likely outcomes are not available.

Theory 1.2: General Definition of Probability

Given a sample space S, with each event A of S (A being a subset of S) there is associated a number $P(A)$, called the **probability** of A, such the following **axioms of probability** are satisfied.

- For every A in S,
$$0 \leqq P(A) \leqq 1. \tag{1.8}$$
- The entire sample space S has the probability
$$P(S) = 1. \tag{1.9}$$
- For mutually exclusive events A and B:
$$P(A \cup B) = P(A) + P(B) \qquad (A \cap B = \emptyset). \tag{1.10}$$
- If S is infinite[14], the previous statement has to be replaced by Eq. (1.4), where for mutually exclusive events $A_1, A_2, \cdots$,
$$P\left(A_1 \cup A_2 \cup \cdots\right) = P\left(A_1\right) + P\left(A_2\right) + \cdots. \tag{1.11}$$

[14] i.e., has infinitely many points.

In the infinite case the subsets of S on which $P(A)$ is defined are restricted to form a so-called σ-algebra.

Basic Theorems of Probability

We will see that the axioms of probability will enable us to build up probability theory and its application to statistics. We begin with three (3) basic theorems. The first one is useful if we can get the probability of the complement A^c more easily than $P(A)$ itself.

Theory 1.3: Complementation Rule

For an event A and its complement A^c in a sample space S,

$$P\left(A^c\right) = 1 - P(A) \tag{1.12}$$

Exercise 1.7: Coin Tossing

Five (5) coils are tossed simultaneously.
Find the probability of the event A:

At least one head turns up. Assume that the coins are fair.

Solution

As each coin can turn up either heads or tails, the sample space consists of $2^5 = 32$ outcomes. Given the coins are fair, we may assign the same probability (1/32) to each outcome. Then the event A^c (No heads turn up) consists of only 1 outcome. Hence $P\left(A^c\right) = 1/32$, and the answer is:

$$P(A) = 1 - P\left(A^c\right) = \frac{31}{32} \quad \blacksquare$$

Theory 1.4: Addition Rule for Mutually Exclusive Events

For **mutually exclusive events** $A_1, \cdots, A_m$ in a sample space S,

$$P\left(A_1 \cup A_2 \cup \cdots A_m\right) = P\left(A_1\right) + P\left(A_2\right) + \cdots + P\left(A_m\right). \tag{1.13}$$

Exercise 1.8: Mutually Exclusive Events

If the probability that on any workday a garage will get 10-20, 21-30, 31-40, over 40 cars to service is 0.20, 0.35, 0.25, 0.12, respectively, what is the probability that on a given workday the garage gets at least 21 cars to service?

Solution

As these are mutually exclusive events, the answer is:

$$0.35 + 0.25 + 0.12 = 0.72 \quad \blacksquare$$

However, most situations, events will **NOT** be mutually exclusive. Then we have the following theorem to formalise the previous statement.

Theory 1.5: Addition Rule for Arbitrary Events

For events A and B in a sample space, their union is defined as:

$$P(A \cup B) = P(A) + P(A) - P(A \cap B). \tag{1.14}$$

For mutually exclusive events A and B we have $A \cap B = \emptyset$ by definition:

$$P\left(\emptyset\right) = 0 \tag{1.15}$$

Exercise 1.9: Union of Arbitrary Events

In tossing a fair die, what is the probability of getting an odd number or a number less than 4?

Solution

Let A be the event "Odd number" and B the event "Number less than 4." As these event are linked we can write:

$$P(A \cup B) = \frac{3}{6} + \frac{3}{6} - \frac{2}{6} = \frac{2}{3}$$

as $A \cup B = \textit{Odd number less than 4} = \{1, 3\}$ $\blacksquare$

Conditional Probability and Independent Events

It is often required to find the probability of an event B given the condition of an event A occurs. This probability is called the **conditional probability** *of B given A* and is denoted by $P\left(B|A\right)$.

In this case A serves as a new, reduced, sample space, and that probability is the fraction of $P(A)$ which corresponds to $A \cap B$. Therefore,

$$P\left(A|B\right) = \frac{P\left(A \cap B\right)}{P\left(A\right)} \qquad \text{where} \qquad P\left(A\right) \neq 0 \tag{1.16}$$

Similarly, the conditional probability of A given B is:

$$P\left(B|A\right) = \frac{P\left(A \cap B\right)}{P\left(B\right)} \qquad \text{where} \qquad P\left(B\right) \neq 0 \tag{1.17}$$

Theory 1.6: Multiplication Rule

Given A and B are events defined in a sample space S and $P\left(A\right) \neq 0, P\left(B\right) \neq 0$, then

$$P\left(A \cap B\right) = P\left(A\right) P\left(B|A\right) = P\left(B\right) P\left(A|B\right). \tag{1.18}$$

Exercise 1.10: Multiplication Rule

In producing screws, let:

- A mean "screw too slim",
- B mean "screw too short."

Let $P\left(A\right) = 0.1$ and let the conditional probability that a slim screw is also too short be $P\left(B|A\right) = 0.2$. What is the probability that a screw that we pick randomly from the lot produced will be both too slim and too short?

Solution

$$P\left(A \cap B\right) = P\left(A\right) P\left(B|A\right) = 0.1 \times 0.2 = 0.02 = 2\% \quad \blacksquare$$

Independent Events

If events A and B are such that

$$P\left(A \cap B\right) = P\left(A\right) P\left(B\right), \tag{1.19}$$

they are called **independent events**. Assuming $P(A) \neq 0, P(B) \neq 0$, we see from Eq. (1.16) - Eq. (1.18):

$$P\left(A|B\right) = P\left(A\right), \qquad P\left(B|A\right) = P\left(B\right).$$

This means that the probability of A does not depend on the occurrence or nonoccurrence of B, and conversely. This justifies the term independent.

Independence of m Events

Similarly, m events $A_1, \cdots, A_m$ are called independent if:

$$P\left(A_1 \cap \cdots \cap A_m\right) = P\left(A_1\right) \cdots P\left(A_m\right) \tag{1.20}$$

as well as for every k different events $A_{j_1}, A_{j_2}, \cdots, A_{j_k}$.

$$P\left(A_{j_1} \cap A_{j_2} \cap \cdots \cap A_{j_k}\right) = P\left(A_{j_1}\right) P\left(A_{j_2}\right) \cdots P\left(A_{j_k}\right) \tag{1.21}$$

where $k = 2, 3, \cdots, m-1$. Accordingly, three events A, B, C are independent if and only if

$$P(A \cap B) = P(A)\, P(B)\,, \tag{1.22}$$
$$P(B \cap C) = P(B)\, P(C)\,, \tag{1.23}$$
$$P(C \cap A) = P(C)\, P(A)\,, \tag{1.24}$$
$$P(A \cap B \cap C) = P(A)\, P(B)\, P(C)\,. \tag{1.25}$$

Sampling

Our next example has to do with randomly drawing objects, *one at a time*, from a given set of objects. This is called **sampling from a population**, and there are two ways of sampling, as follows.

- **In sampling with replacement**, the object that was drawn at random is placed back to the given set and the set is mixed thoroughly. Then we draw the next object at random.
- **In sampling without replacement** the object that was drawn is put aside.

Exercise 1.11: Sampling w/o Replacement

A box contains 10 screws, three (3) of which are defective. Two screws are drawn at random. Find the probability that neither of the two screws is defective.

Solution

We consider the events

A First drawn screw non-defective,

B Second drawn screw non-defective.

We can see:

$$P(A) = \frac{1}{10}$$

as 7 of the 10 screws are non-defective and we sample at random, so that each screw has the same probability ($\frac{1}{10}$) of being picked.
If we sample with replacement, the situation before the second drawing is the same as at the beginning, and $P(B) = \frac{7}{10}$. The events are independent, and the answer is

$$P(A \cap B) = P(A)\, P(B) = 0 \cdot 7 \cdot 0.7 = 0.49\%.$$

If we sample without replacement, then $P(A) = \frac{7}{10}$, as before. If A has occurred, then there are 9 screws left in the box, 3 of which are defective.
Thus $P\left(B|A\right) = \frac{6}{9} = \frac{2}{3}$, therefore:

$$P(A \cap B) = \frac{7}{10} \cdot \frac{2}{3} = 47\% \quad \blacksquare$$

1.4 Permutations & Combinations

Permutations and combinations help in finding probabilities $P(A) = a/k$ by systematically counting the number a of points of which an event A consists.

where, k is the number of points of the sample space S.

The practical difficulty is that a may often be surprisingly large, so that actual counting becomes hopeless. For example, if in assembling some instrument you need 10 different screws in a certain order and you want to draw them randomly from a box[15] the probability of obtaining them in the required order is only 1/3,628,800 because there are exactly:

$$10! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7 \cdot 8 \cdot 9 \cdot 10 = 3,628,800$$

orders in which they can be drawn. Similarly, in many other situations the numbers of orders, arrangements, etc. are often incredibly large.

[15] Of course, this goes without saying, there is nothing but screws in this imaginary box.

1.4.1 Permutations

A **permutation** of given things[16] is an arrangement of these things in a row in some order.

[16] such as *elements* or *objects*.

> i.e., for three (3) letters a, b, c there are $3! = 1 \cdot 2 \cdot 3 = 6$ permutations: abc, acb, bca, cab, cba

Let's write this behaviour down as a theory:

Theory 1.7: Permutations

Different things
The number of permutations of n different things taken all at a time is

$$n! = 1 \cdot 2 \cdot 3, \cdots, n. \tag{1.26}$$

Classes of Equal Things
If n given things can be divided into c classes of alike things differing from class to class, then the number of permutations of these things taken all at a time is

$$\frac{n!}{n_1! n_2! \cdots n_c!} \quad \text{where} \quad n_1 + n_2 + \cdots + n_c = n, \tag{1.27}$$

where n_j is the number of things in the j^{th} class.

Permutation of n things taken k at a time

A permutation containing only k of the n given things. Two such permutations consisting of the same k elements, in a different order, are different, by definition.

> i.e., there are 6 different permutations of the three letters a, b, c, taken two letters at a time, ab, ac, bc, ba, ca, cb.

Permutation of n things taken k at a time with repetitions

An arrangement obtained by putting any given thing in the first position, any given thing, including a repetition of the one just used, in the second, and continuing until k positions are filled.

i.e., there are $3^2 = 9$ different such permutations of a, b, c taken 2 letters at a time, namely, the preceding 6 permutations and aa, bb, cc.

Theory 1.8: Permutations

The number of different permutations of n different things taken k at a time **without repetitions** is

$$n(n-1)(n-2)\cdots(n-k+1) = \frac{n!}{(n-k)!}, \tag{1.28}$$

and **with repetitions** is,

$$n^k. \tag{1.29}$$

Exercise 1.12: An Encrypted Message

In an encrypted message the letters are arranged in groups of five (5) letters, called words. Knowing the letter can be repeated, we see that the number of different such words is

$$26^5 = 11,881,376 \quad \blacksquare$$

For the case of different such words containing each letter no more than once is

$$\frac{26!}{(26-5)!} = 26 \cdot 25 \cdot 24 \cdot 23 \cdot 22 = 7,893,600 \quad \blacksquare$$

1.4.2 Combinations

In a permutation, the order of the selected things is essential. In contrast, a **combination** of a given things means any selection of one or more things without regard to order. There are two (2) kinds of combinations, as follows:

1. The number of **combinations of n different things, taken k at a time, without repetitions** is the number of sets that can be made up from the n given things, each set containing k different things and no two (2) sets containing exactly the same k things.

2. The number of **combinations of n different things, taken k at a time, with repetitions** is the number of sets that can be made up of k things chosen from the given n things, each being used as often as desired.

   i.e, there are three (3) combinations of the three (3) letters a, b, c, taken two (2) letters at a time, without repetitions, namely, ab, ac, bc, and six such combinations with repetitions, namely, ab, ac, bc, ca, bb, cc.

Theory 1.9: Combinations

The number of different combinations of n different things taken, k at a time, **without repetitions**, is:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)\cdots(n-k+1)}{1\cdot 2\cdots k}, \tag{1.30}$$

and the number of those combinations **with repetitions** is:

$$\binom{n+k-1}{k}. \tag{1.31}$$

Exercise 1.13: Sampling Light-bulbs

The number of samples of five (5) light-bulbs that can be selected from a lot of 500 bulbs is

$$\binom{500}{5} = \frac{500!}{5!495!} = \frac{500\cdot 499\cdot 498\cdot 497\cdot 476}{1\cdot 2\cdot 3\cdot 4\cdot 5} = 255,244,687,600 \quad \blacksquare$$

1.4.3 Factorial Function

In Eq. (1.26)-Eq. (1.31) the **factorial function** is relatively straightforward. By definition[17],

$$0! = 1.$$

Values may be computed recursively from given values by

$$(n+1)! = (n+1)n!.$$

For large n the function is very large and hard to keep track of. A convenient approximation for large n is the **Stirling formula**, defined as:

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \qquad \text{where} \qquad e = 2.718\cdots \tag{1.32}$$

where $\sim$ is read asymptotically equal[18] and means that the ratio of the two sides of Eq. (1.32) approaches 1 as n approaches infinity.

[17] This is done by convention. An intuitive way to look at it is $n!$ counts the number of ways to arrange distinct objects in a line, and there is only one way to arrange nothing.

[18] it means the percentage difference between the vertical distances between points on the two graphs approaches 0.

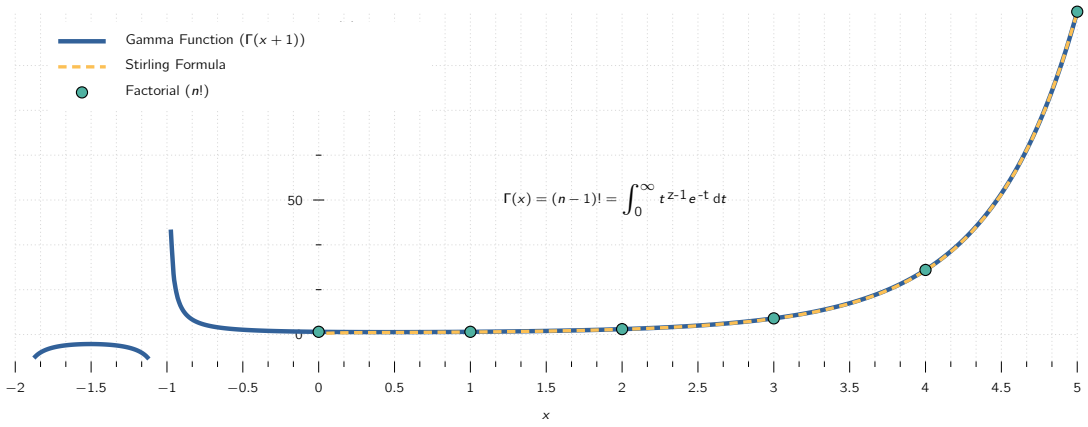


Figure 1.2: A visual comparison of the Stirling formula and the actual values of the factorial function.

1.4.4 Binomial Coefficients

The **binomial coefficients** are defined by the following formula:

$$\binom{a}{k} = \frac{(a)\,(a-1)\,(a-2)\cdots(a-k+1)}{k!} \qquad \text{where} \qquad (k \geq 0, \text{integer}) \qquad (1.33)$$

The numerator has k factors. Furthermore, we define

$$\binom{a}{0} = 1, \qquad \text{in particular,} \qquad \binom{0}{0} = 1.$$

For integer $a = n$ we obtain from Eq. (1.33):

$$\binom{n}{k} = \binom{n}{n-k} \qquad (n \geqq 0 \quad \text{and} \quad 0 \leqq k \leqq n).$$

Binomial coefficients may be computed recursively, because

$$\binom{a}{k} + \binom{a}{k+1} = \binom{a+1}{k+1} \qquad (k \geqq 0, \text{ integer}).$$

Formula Eq. (1.33) also gives:

$$\binom{-m}{k} = (-1)^k \binom{m+k-1}{k} \qquad \text{where} \qquad k \geqq 0, \text{ integer} \quad \text{and} \quad m > 0.$$

There are two (2) important relations worth mentioning:

$$\sum_{s=0}^{n-1} \binom{k+s}{k} = \binom{n+k}{k+1} \qquad (k \geq 0 \quad \text{and} \quad n \geq 1)$$

and

$$\sum_{k=0}^{r} \binom{p}{k}\binom{q}{r-k} = \binom{p+q}{r} \qquad (r \geq 0, \text{ integer}).$$

1.5 Random Variables and Probability Distributions

In the beginning of this chapter we considered frequency distributions of data[19]. These distributions show the absolute or relative frequency of the data values.

[19] Remember we did a histogram and a stem-and-leaf plot.

Similarly, a **probability distribution** or, a **distribution**, shows the probabilities of events in an experiment. The quantity we observe in an experiment will be denoted by X and called a random variable[20] as the value it will assume in the next trial depends on the **stochastic process**

> i.e., if you roll a die, you get one of the numbers from 1 to 6, but you don't know which one will show up next. An example would be, X = Number a die turns up, which is a random variable.

[20] or **stochastic variable** if you want to be pedantic.

If we count[21], we have a **discrete random variable and distribution**. If we measure (electric voltage, rainfall, hardness of steel), we have a **continuous random variable and distribution**. For both cases (discrete, discontinuous), the distribution of X is determined by the **distribution function**:

[21] cars on a road, defective parts in a production, tosses until a die shows the first six (6) .

$$F(x) = P(X \leq x) \tag{1.34}$$

This is the probability that in a trial, X will assume any value not exceeding x.

> The terminology is unfortunately **NOT** uniform across the field as $F(x)$ is sometimes also called the **cumulative distribution function**.

For Eq. (1.34) to make sense in both the discrete and the continuous case we formulate conditions as follows.

Theory 1.10: Random Variable

A **random variable** X is a function defined on the sample space S of an experiment. Its values are real numbers. For every number a the probability:

$$P(X = a),$$

with which X assumes a is defined. Similarly, for any interval I, the probability

$$P(X \in I),$$

with which X assumes any value in I is defined[22].

[22] Although this definition is very general, in practice only a very small number of distributions will occur over and over again in applications.

From Eq. (1.34) we can define the fundamental formula for the probability corresponding to an interval $a < x \leqq b$:

$$P(a < X \leqq b) = F(b) - F(a). \tag{1.35}$$

This follows because $X \leqq a$ (X assumes any value **NOT** exceeding a) and $a < X \leqq b$ (X assumes any value in the interval $a < x \leqq b$) are mutually exclusive events, so based on Eq. (1.34):

$$\begin{aligned} F(b) = P(X \leqq b) &= P(X \leqq a) + P(a < X \leqq b) \\ &= F(a) + P(a < X \leqq b) \end{aligned}$$

and subtraction of $F(a)$ on both sides gives Eq. (1.35).

1.5.1 Discrete Random Variables and Distributions

By definition, a random variable X and its distribution are **discrete** if X assumes only finitely many or at most countably many values $x_1, x_2, x_3, \cdots$, called the **possible values** of X, with positive probabilities,

$$p_1 = P\left(X = x_1\right), p_2 = P\left(X = x_2\right), \quad p_3 = P\left(X = x_3\right), \cdots$$

whereas the probability $P\left(X \in I\right)$ is zero for any interval I containing no possible value. Clearly, the discrete distribution of X is also determined by the **probability function** $f\left(x\right)$ of X, defined by

$$f\left(x\right) = \begin{cases} p_j & \text{if} \quad x = x_j \\ 0 & \text{otherwise} \end{cases} \qquad \text{where} \qquad j = 1, 2, \cdots, \tag{1.36}$$

From this we get the values of the **distribution function** $F\left(x\right)$ by taking sums,

$$F\left(x\right) = \sum_{x_j \leq x} f\left(x_j\right) = \sum_{x_j \leq x} p_j \tag{1.37}$$

where for any given x we sum all the probabilities p_j for which x_j is smaller than or equal to that of x. This is a **step function** with upward jumps of size p_j at the possible values x_j of X and constant in between. The two (2) useful formulas for discrete distributions are readily obtained as follows. For the probability corresponding to intervals we have from Eq. (1.35) and Eq. (1.37):

$$P\left(a < X \leqq b\right) = F\left(b\right) - F\left(a\right) = \sum_{a < x_j \leqq b} p_j \tag{1.38}$$

[23] Be careful about $<$ and $\leqq$ as the former means it is NOT included and the latter means it is.

This is the sum of all probabilities p_j for which x_j satisfies $a < x_j \leqq b$[23]. From this and $P\left(S\right) = 1$ we obtain the following formula.

$$\sum_j p_j = 1 \qquad \text{(sum of all probabilities)}. \tag{1.39}$$

Exercise 1.14: Waiting Time Problem

In tossing a fair coin, let X be the Number of trials until the first head appears. Then, by independence of events we get (where H is heads, and T is tails):

$$\begin{aligned} P(X=1) = P(H) &= \frac{1}{2} \\ P(X=2) = P(TH) &= \frac{1}{2}\cdot\frac{1}{2} = \frac{1}{4} \\ P(X=3) = P(TTH) &= \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2} = \frac{1}{8} \end{aligned}$$

and in general, $P(X=n) = \left(\frac{1}{2}\right)^2$, $n = 1, 2, 3, \ldots$ which when all possible event are summed up will always give 1.

1.5.2 Continuous Random Variables and Distributions

Discrete random variables appear in experiments in which we count[24]. Continuous random variables appear in experiments in which we measure (lengths of screws, voltage in a power line, etc.). By definition, a random variable X and its distribution are of *continuous type* or, briefly, **continuous**, if its distribution function $F(x)$, defined in Eq. (1.34), can be given by an integral[25]:

$$F(x) = \int_{-\infty}^{x} f(v)\, dv \tag{1.40}$$

[24] defectives in a production, days of sunshine in Kufstein, customers in a line, etc.

[25] we write v as a toss-away variable because x is needed as the upper limit of the integral.

whose integrand $f(x)$, called the **density** of the distribution, is non-negative, and is continuous, perhaps except for finitely many x-values. Differentiation gives the relation of f to F as

$$f(x) = F'(x) \tag{1.41}$$

for every x at which $f(x)$ is continuous.

From Eq. (1.35) and Eq. (1.40) we obtain the very important formula for the probability corresponding to an interval[26]:

$$P(a < X \leq b) = F(b) - F(a) = \int_{a}^{b} f(v)\, dv \tag{1.42}$$

[26] This is an analog of Eq. (1.38)

Which can be seen visually in **Fig.** 1.3. From Eq. (1.40) and $P(S) = 1$ we also have the analogue of Eq. (1.39):

$$\int_{-\infty}^{\infty} f(v)\, dv = 1. \tag{1.43}$$

Continuous random variables are simpler than discrete ones with respect to intervals as, in the continuous case the four probabilities corresponding to $a < X \leq b$, $a < X < b$, $a \leq X \leq b$,

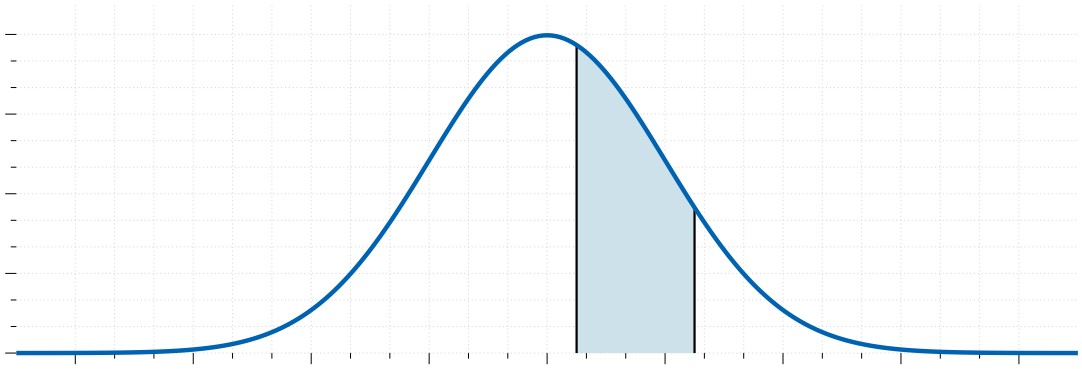


Figure 1.3: A visual representation of the Eq. (1.42).

and $a \leq X \leq b$ with any fixed a and b $(> a)$ are all the same.

The next example illustrates notations and typical applications of our present formulas.

Exercise 1.15: Continuous Distribution

Let X have the density function:

$$f(x) = 0.75(1 - x^2) \qquad \text{if} \qquad -1 \leq x \leq 1,$$

and zero otherwise. Find:

1. The distribution function.
2. Find the probabilities $P\left(-\frac{1}{2} \leq X \leq \frac{1}{2}\right)$ and $P\left(\frac{1}{2} \leq X \leq 2\right)$
3. Find x such that $P(X \leq x) = 0.95$.

Solution

From Eq. (1.40), we obtain $F(x) = 0$ if $x \leqq -1$,

$$F(x) = 0.75 \int_{-1}^{x} (1 - v^2)\, dv = 0.5 + 0.75x - 0.25x^3 \qquad \text{if} \qquad -1 < x \leqq 1,$$

and $F(x) = 1$ if $x > 1$. From this and Eq. (1.42) we get:

$$P(-\tfrac{1}{2} \leq X \leq \tfrac{1}{2}) = F(\tfrac{1}{2}) - F(-\tfrac{1}{2}) = 0.75 \int_{-1/2}^{1/2} (1 - v^2)\, dv = 68.75\%$$

because $P(-\frac{1}{2} \leq X \leq \frac{1}{2}) = P(-\frac{1}{2} < X \leq \frac{1}{2})$ for a continuous distribution we can write:

$$P(\frac{1}{4} \leq X \leq 2) = F(2) - F(\frac{1}{4}) = 0.75 \int_{1/4}^{1} (1 - v^2)\, dv = 31.64\%.$$

Note that the upper limit of integration is 1, not 2. Finally,

$$P(X \leq x) = F(x) = 0.5 + 0.75x - 0.25x^2 = 0.95.$$

Algebraic simplification gives $3x - x^3 = 1.8$. A solution is $x = 0.73$, approximately ■

1.6 Mean and Variance of a Distribution

The mean μ and variance σ^2 of a random variable X and of its distribution are the theoretical counterparts of the mean $\overline{x}$ and variance s^2 of a frequency distribution and serve a similar purpose.

The mean characterises the central location and the variance the spread (the variability) of the distribution. The **mean** μ is defined by:

$$\text{(a)} \quad \mu = \sum_j x_j f\left(x_j\right) \qquad \text{(Discrete distribution)} \tag{1.44a}$$

$$\text{(b)} \quad \mu = \int_{-\infty}^{\infty} x f(x)\,dx \qquad \text{(Continuous distribution)} \tag{1.44b}$$

and the **variance** σ^2 by:

$$\text{(a)} \quad \sigma^2 = \sum_j (x_j - \mu)^2 f\left(x_j\right) \qquad \text{(Discrete distribution)} \tag{1.45a}$$

$$\text{(b)} \quad \sigma^2 = \int_{-\infty}^{\infty} (x-\mu)^2 f(x)\,dx \qquad \text{(Continuous distribution)} \tag{1.45b}$$

σ (the positive square root of σ^2) is called the standard deviation[27] of X and its distribution. f is the probability function or the density, respectively, in (a) and (b).

[27] Sometimes it is known as Var (x)

The mean μ is also denoted by $E(X)$ and is called the **expectation of** X because it gives the average value of X to be expected in many trials.

> Quantities such as μ and σ^2 that measure certain properties of a distribution are called **parameters**. μ and σ^2 are the two (2) most important ones.

From Eq. (1.45a) and Eq. (1.45b), we see that[28]:

$$\sigma^2 > 0$$

[28] except for a discrete distribution with only one possible value.

We assume that μ and σ^2 exist[29], as is the case for practically all distributions that are useful in applications.

[29] and finite.

> **Exercise** 1.16: Mean and Variance
>
> The random variable X, *Number of heads in a single toss of a fair coin*, has the possible values $X = 0$ and $X = 1$ with probabilities $P(X=0) = \frac{1}{2}$ and $P(X=1) = \frac{1}{2}$. From Eq. (1.44a) we thus obtain the mean:
>
> $$\mu = 0 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} = \frac{1}{2},$$
>
> and Eq. (1.45a) gives the variance:
>
> $$\sigma^2 = (0 - \frac{1}{2})^2 \cdot \frac{1}{2} + (1 - \frac{1}{2})^2 \cdot \frac{1}{2} = \frac{1}{4} \quad \blacksquare$$

Symmetry

We can obtain the mean μ without calculation if a distribution is symmetric. Indeed, we can write:

Theory 1.11: Mean of a Symmetric Distribution

If a distribution is **symmetric** with respect to $x = c$, that is,

$$f(c - x) = f(c + x)$$

then $\mu = c$.

Transformation of Mean and Variance

Given a random variable X with mean μ and variance σ^2, we want to calculate the mean and variance of $X^* = a_1 + a_2 X$, where a_1 and a_2 are given constants.

This problem is important in statistics, where it often appears.

Theory 1.12: Transformation of Mean and Variance

If a random variable X has mean μ and variance σ^2, then the random variable:

$$X^* = a_1 + a_2 X \qquad \text{where} \qquad a_2 > 0$$

has the mean μ^* and variance σ^{*2}, where

$$\mu^* = a_1 + a_2 \mu \qquad \text{and} \qquad \sigma^{*2} = a_2^2 \sigma^2.$$

In particular, the **standardised random variable** Z corresponding to X, given by:

$$Z = \frac{X - \mu}{\sigma}$$

has the mean 0 and the variance 1.

Expectation & Moments

[30] the value of X to be expected on the average

If we recall, Eq. (1.44a) and Eq. (1.44b) define the mean of X[30], written $\mu = E(X)$. More generally, if $g(x)$ is non-constant and continuous for all x, then $g(X)$ is a random variable. Therefore its mathematical expectation or, briefly, its expectation $E\left(g(X)\right)$ is the value of $g(X)$ to be expected on the average, defined by:

$$E(g(X)) = \sum_j g\left(x_j\right) f\left(x_j\right) \qquad \text{or} \qquad E(g(X)) = \int_{-\infty}^{\infty} g(x) f(x)\, dx$$

In the formula on the Left Hand Side (LHS), f is the probability function of the discrete random variable X. In the formula on the Right Hand Side (RHS), f is the density of the continuous random variable X. Important special cases are the k$^{\text{th}}$ of X (where $k = 1, 2, \cdots$)

$$E\left(X^k\right) = \sum_j x_j^k f\left(x_j\right) \qquad \text{or} \qquad \int_{-\infty}^{\infty} x^k f(x)\ dx$$

and the k$^{\text{th}}$ of X ($k = 1, 2, \cdots$)

$$E\left([X - \mu]^k\right) = \sum_j \left(x_j - \mu\right)^k f\left(x_j\right) \qquad \text{or} \qquad \int_{-\infty}^{\infty} (x - \mu)^k f(x)\, dx.$$

This includes the first moment, the **mean** of X

$$\mu = E(X) \qquad \text{where} \qquad k = 1 \tag{1.46}$$

It also includes the second central moment, the **variance** of X

$$\sigma^2 = E\left([X - \mu]^2\right) \qquad \text{where} \qquad k = 2. \tag{1.47}$$

1.7 Binomial, Poisson, and Hyper-geometric Distributions

These are the three (3) most important discrete distributions, with numerous applications therefore are worth of a bit of a detailed look.

> Of course these are not the only distributions present. There are as many distributions as there are problems with some distributions used in wide variety of fields (Gaussian) whereas some are used only in a very narrow field (Nakagami).

Binomial Distribution

The **binomial distribution** occurs in problems involving of chance[31].

[31] rolling a dice, quality inspection (e.g., counting of the number of defectives), opinion plots (counting number of employees favouring certain schedule changes, etc.), medicine (e.g., recording the number of patterns who covered on a new medication)

What we are interested is in the number of times an event A occurs in n independent trials. In each trial, the event A has the same probability $P(A) = p$. Then in a trial, A will **NOT** occur with probability $q = 1 - p$. In n trials the random variable that interests us is:

$$X = \text{Number of times the event A occurs in n trials.} \tag{1.48}$$

X can assume the values 0, 1, $\cdots$, n, and we want to determine the corresponding probabilities. Now $X = x$ means that A occurs in x trials and in $n - x$ trials it does not occur. We can write this down as follows:

$$\underbrace{A \;\; A \;\; \cdots A}_{x \text{ times}} \qquad \text{and} \qquad \underbrace{B \;\; B \;\; \cdots B}_{n \text{ - } x\text{times}} \tag{1.49}$$

Here $B = A^c$ is the complement of A, meaning that A does not occur. We now use the assumption that the trials are independent[32]. Hence Eq. (1.49) has the probability:

[32] e.g., they do **NOT** influence each other

$$\underbrace{p \;\; p \;\; \cdots p}_{x \text{ times}} \cdot \underbrace{q \;\; q \;\; \cdots q}_{n \text{ - } x\text{times}} = p^x q^{n-x} \tag{1.50}$$

Now Eq. (1.49) is just one order of arranging $x A$'s and $n - x B$'s. We will now calculate the number of permutations of n things[33] consisting of two (2) classes;

[33] the n outcomes of the n trials

1. class 1 containing the $n_1 = x$ A's
2. class 2 containing the $n - n_1 = n - x$ B's

This number is:

$$\frac{n!}{x!(n - x)!} = \binom{n}{x}. \tag{1.51}$$

Accordingly, Eq. (1.50), multiplied by this binomial coefficient, gives the probability $P(X = x)$ of $X = x$, that is, of obtaining A precisely x times in n trials. Hence X has the probability function:

$$f(x) = \binom{n}{x} p^x q^{n-x} \qquad (x = 0, 1, \cdots, n) \tag{1.52}$$

and $f(x) = 0$ otherwise. The distribution of X with probability function (2) is called the **binomial distribution** or *Bernoulli distribution*. The occurrence of A is called success[34] and the non-occurrence of A is called *failure*.

[34] regardless of what it actually is; it may mean that you miss your plane or lose your watch

The mean of the binomial distribution is:

$$\mu = np$$

and the variance is:

$$\sigma^2 = npq.$$

For the *symmetric case* of equal chance of success and failure ($p = q = \frac{1}{2}$) this gives the mean $n/2$, the variance $n/4$, and the probability function

$$f(x) = \binom{n}{x} \left(\frac{1}{2}\right) \qquad (x = 0, 1, \cdots, n).$$

Exercise 1.17: Binomial Distribution

Calculate the probability of obtaining at least two (2) "six" in rolling a fair die 4 times.

Solution

$p = P(A) = P(\text{six}) = \frac{1}{6}$, $q = \frac{5}{6}$, $n = 4$. The event "At least two (2) "six" occurs if we obtain 2 or 3 or 4 "six" Hence the answer is:

$$P = f(2) + f(3) + f(4) = \binom{4}{2}\left(\frac{1}{6}\right)^2\left(\frac{5}{6}\right)^2 + \binom{4}{3}\left(\frac{1}{6}\right)^3\left(\frac{5}{6}\right) + \binom{4}{4}\left(\frac{1}{6}\right)^4$$

$$= \frac{1}{6^4}(6 \cdot 25 + 4 \cdot 5 + 1) = \frac{171}{1296} = 13.2\%.$$

Poisson Distribution

The discrete distribution with infinitely many possible values and probability function:

$$f(x) = \frac{\mu^x}{x!} e^{-\mu} \qquad \text{where} \qquad x = 0, 1, \cdots \tag{1.53}$$

is called the **Poisson distribution**, named after *S. D. Poisson*. **Fig.** 1.4 shows Eq. (1.53) for some values of μ[35].

[35] While μ is used here, some textbook use λ

It can be proved that this distribution is obtained as a limiting case of the binomial distribution, if we let $p \to 0$ and $n \to \infty$ so that the mean $\mu = np$ approaches a finite value. The Poisson distribution has the mean μ and the variance:

$$\sigma^2 = \mu. \tag{1.54}$$

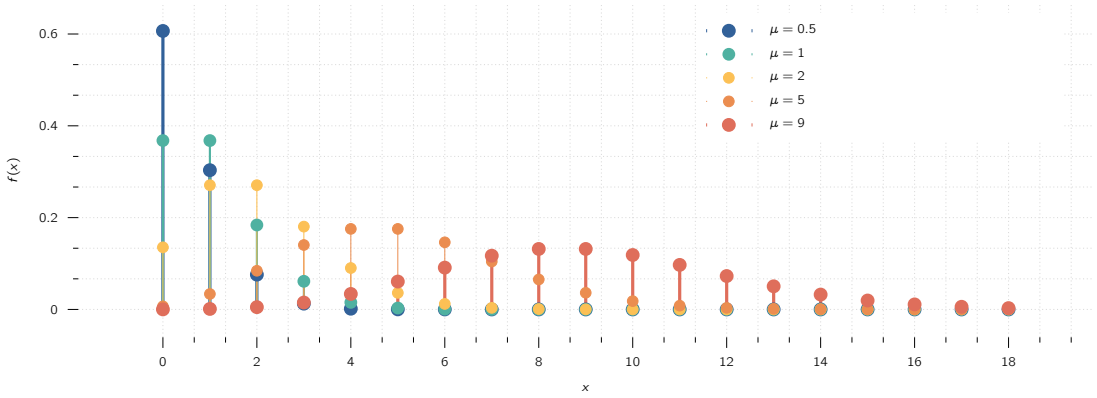


Figure 1.4: The Poisson distribution with different mean (μ) values.

Fig. 1.4 gives the impression that, with increasing mean, the spread of the distribution increases, thereby illustrating formula Eq. (1.54), and that the distribution becomes more and more symmetric[36] .

[36] approximately

Exercise 1.18: Poisson Distribution

If the probability of producing a defective screw is $p = 0.01$, what is the probability that a lot of 100 screws will contain more than 2 defectives?

Solution

The complementary event is A^c. No more than 2 defectives. For its probability we get, from the binomial distribution with mean $\mu = np = 1$, the value.

$$P\left(A^c\right) = \begin{pmatrix} 100 \\ 0 \end{pmatrix} 0.99^{100} + \begin{pmatrix} 100 \\ 1 \end{pmatrix} 0.01 \cdot 0.99^{100} + \begin{pmatrix} 100 \\ 2 \end{pmatrix} 0.01^2 \cdot 0.99^{100}.$$

Since p is very small, we can approximate this by the much more convenient Poisson distribution with mean $\mu = np = 100 \cdot 0.01 = 1$, obtaining.

$$P\left(A^c\right) = e^{-1}\left(1 + 1 + \frac{1}{2}\right) = 91.97\%.$$

Thus $P(A) = 8.03\%$. Show that the binomial distribution gives $P(A) = 7.94\%$, so that the Poisson approximation is quite good ■

Exercise 1.19: The Parking Problem

If on the average, 2 cars enter a certain parking lot per minute, what is the probability that during any given minute four (4) or more cars will enter the lot?

Solution

To understand that the Poisson distribution is a model of the situation, we imagine the minute to be divided into very many short time intervals. Let p be the (constant) probability that a car will enter the lot during any such short interval, and assume independence of the events that happen during those intervals. Then, we are dealing with a binomial distribution with very large n and very small p, which we can approximate by the Poisson distribution with

$$\mu = np = 2$$

because 2 cars enter on the average, the complementary event of the event "*4 cars or more during a given minute*" is "*3 cars or fewer enter the lot*" and has the probability

$$f(0) + f(1) + f(2) + f(3) = e^{-2}\left(\frac{2^0}{0} + \frac{2^1}{1!} + \frac{2^2}{2!} + \frac{2^3}{3!}\right) = 0.857.$$

Which means the result is 14.3% ■

1.7.1 Sampling with Replacement

This means that we draw things from a given set one by one, and after each trial we replace the thing drawn[37] before we draw the next thing. This guarantees independence of trials and leads to the **binomial distribution**. Indeed, if a box contains N things, for example, screws, M of which are defective, the probability of drawing a defective screw in a trial is $p = M/N$. Hence the probability of drawing a nondefective screw is $q = 1 - p = 1 - M/N$, and Eq. (1.52) gives the probability of drawing x defectives in n trials in the form:

[37] put it back to the given set and mix.

$$f(x) = \binom{n}{x} \left(\frac{M}{N}\right)^x \left(1 - \frac{M}{N}\right)^{n-x} \qquad (x = 0, 1, \cdots, n). \tag{1.55}$$

1.7.2 Sampling without Replacement: Hyper-geometric Distribution

Sampling without replacement means that we return no screw to the box. Then we no longer have independence of trials, and instead of Eq. (1.55) the probability of drawing x defectives in n trials is:

$$f(x) = \frac{\binom{M}{x}\binom{N-M}{n-x}}{\binom{N}{n}} \qquad \text{where} \qquad x = 1, 2, \ldots, n. \tag{1.56}$$

The distribution with this probability function is called the hyper-geometric distribution[38].

[38] because its moment generating function can be expressed by the hypergeometric function, which is a fact only useful to write it in a margin.

The hypergeometric distribution has the mean:

$$\mu = n\frac{M}{N},$$

and the variance

$$\sigma^2 = \frac{nM(N-M)(N-n)}{N^2(N-1)}.$$

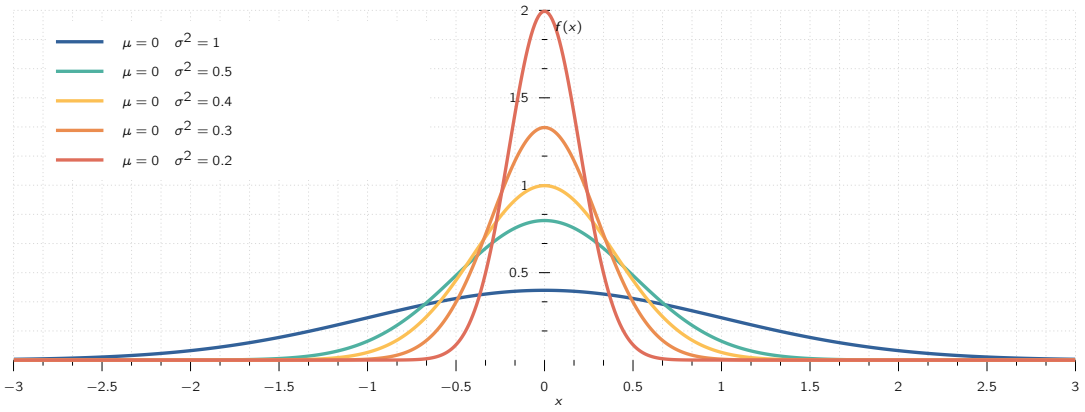


Figure 1.5: The poster child of probability and statistics, the normal distribution.

1.8 Normal Distribution

Turning from discrete to continuous distributions, in this section we discuss the normal distribution. This is the most important continuous distribution because in applications many random variables are normal random variables[39] or they are approximately normal or can be transformed into normal random variables in a relatively simple fashion. Furthermore, the normal distribution is a useful approximation of more complicated distributions, and it also occurs in the proofs of various statistical tests.

[39] that is, they have a normal distribution.

The **normal distribution** or *Gauss distribution* is defined as the distribution with the density:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) \tag{1.57}$$

where exp is the exponential function with base $e = 2.718\cdots$. This is simpler than it may at first look. $f(x)$ has these features (see also **Fig.** 1.5).

1. μ is the mean, and σ the standard deviation.
2. $1/(\sigma\sqrt{2\pi})$ is a constant factor that makes the area under the curve of $f(x)$ from $-\infty$ to ∞ equal to 1, as it must be[40].
3. The curve of $f(x)$ is symmetric with respect to $x = \mu$ because the exponent is quadratic. Hence for $\mu = 0$ it is symmetric with respect to the y-axis $x = 0$ [41].
4. The exponential function in Eq. (1.57) goes to zero very fast–the faster the smaller the standard deviation σ is, as it should be, as seen in **Fig.** 1.5.

[40] Having a probability higher than 1 does NOT make sense

[41] This distribution is also known as bell-shaped curves.

1.8.1 Distribution Function

From Eq. (1.55) and Eq. (1.57) we see that the normal distribution has the **distribution function** of the following form:

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left[-\frac{1}{2}\left(\frac{v-\mu}{\sigma}\right)^2\right] dv. \tag{1.58}$$

Here we needed x as the upper limit of integration and wrote v (instead of x) in the integrand.

For the corresponding **standardised normal distribution** with mean 0 and standard deviation 1 we denote $F(x)$ by $\Phi(z)$. Then we simply have from Eq. (1.58).

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-u^2/2} du. \tag{1.59}$$

> This integral cannot be integrated by one of the methods of calculus.

But this is no serious handicap because its values can be obtained from standardised tables. These values are needed in working with the normal distribution. The curve of $\Phi(z)$ is S-shaped. It increases monotone from 0 to 1 and intersects the vertical axis at $\frac{1}{2}$, as shown in **Fig.** 1.6.

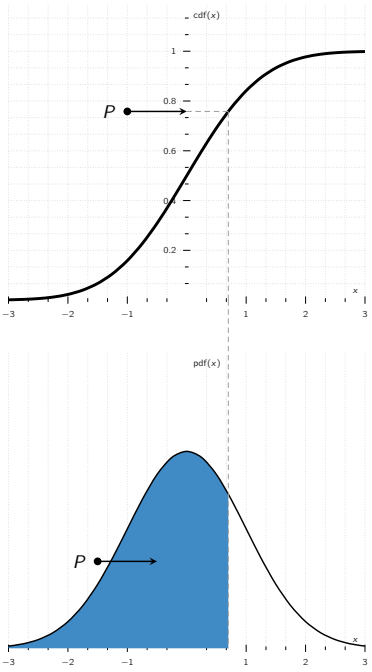


Figure 1.6: A visual representation between the relationship of PDF and CDF.

Theory 1.13: Relationship between PDF and CDF

The distribution function $F(x)$ of the normal distribution with any μ and σ is related to the standardised distribution function $\Phi(z)$ in Eq. (1.59) by the formula

$$F(x) = \Phi\left(\frac{x-\mu}{\sigma}\right).$$

Theory 1.14: Normal Probabilities for Intervals

The probability a normal random variable X with mean μ and standard deviation σ assume any value in an interval $a < x \equiv b$ is:

$$P(a < X \leq b) = F(b) - F(a) = \Phi\left(\frac{b-\mu}{\sigma}\right) - \Phi\left(\frac{a-\mu}{\sigma}\right).$$

1.8.2 Numeric Values

In practical work with the normal distribution it is good to remember that about 67% of all values of X to be observed will be between $\mu \pm \sigma$, about 95% between $\mu \pm 2\sigma$, and practically all between

the **three-sigma limits** $\mu \pm 3\sigma$:

$$P(\mu - \sigma < X \leq \mu + \sigma) \approx 68\% \tag{1.60a}$$
$$P(\mu - 2\sigma < X \leq \mu + 2\sigma) \approx 95.5\% \tag{1.60b}$$
$$P(\mu - 3\sigma < X \leq \mu + 3\sigma) \approx 99.7\%. \tag{1.60c}$$

The aforementioned formulas show that a value deviating from μ by more than σ, 2σ, or 3σ will occur in one of about 3, 20, and 300 trials, respectively.

[42] Which we shall cover in Chapter 2.

In tests[42], we shall ask, conversely, for the intervals that correspond to certain given probabilities; practically most important use the probabilities of 95%, 99%, and 99.9%. For these,the answers are $\mu \pm 2\sigma$, $\mu \pm 2.6\sigma$, and $\mu \pm 3.3\sigma$, respectively.

More precisely,

$$P(\mu - 1.96\sigma < X \leq \mu + 1.96\sigma) \approx 95\% \tag{1.61a}$$
$$P(\mu - 2.58\sigma < X \leq \mu + 2.58\sigma) \approx 99\% \tag{1.61b}$$
$$P(\mu - 3.29\sigma < X \leq \mu + 3.29\sigma) \approx 99.9\%. \tag{1.61c}$$

1.8.3 Normal Approximation of the Binomial Distribution

The probability function of the binomial distribution, as a reminder, is:

$$f(x) = \binom{n}{x} p^x q^{n-x} \qquad (x = 0, 1, \cdots, n). \tag{1.62}$$

[43] and theoretical

If n is large, the binomial coefficients and powers become very inconvenient. It is of great practical[43] importance that, in this case, the normal distribution provides a good approximation of the binomial distribution, according to the following theorem, one of the most important theorems in all probability theory.

Theory 1.15: Limit Theorem of De Moivre and Laplace

For large n,

$$f(x) \sim f^*(x) \qquad \text{where} \qquad x = 0, 1, \cdots n$$

Here f is given by Eq. (1.62). The function

$$f^*(\cdot) = \frac{1}{\sqrt{2\pi}\sqrt{npq}} \exp\left(-\frac{z^2}{2}\right), \qquad \text{and} \qquad z = \frac{x - np}{\sqrt{npq}}$$

is the density of the normal distribution with mean $\mu = np$ and variance $\sigma^2 = npq$ (the mean and variance of the binomial distribution). Furthermore, for any nonnegative integers a and b ($> a$):

$$P(a \leq X \leq b) = \sum_{x=a}^{b} \binom{n}{x} p^x q^{n-x} \sim \Phi(\beta) - \Phi(\alpha)$$

where,

$$\alpha = \frac{a - np - 0.5}{\sqrt{npq}} \qquad \text{and} \qquad \beta = \frac{b - np + 0.5}{\sqrt{npq}}$$

1.9 Distribution of Several Random Variables

Distributions of two (2) or more random variables are of interest for two (2) reasons:

1. They occur in experiments in which we observe several random variables, for example, carbon content X and hardness Y of steel, amount of fertiliser X and yield of corn Y, height X_1, weight X_2, and blood pressure X_3 of persons, and so on.

2. They will be needed in the mathematical justification of the methods of statistics in Chapter 2.

In this section we consider two (2) random variables X and Y or, as we also say, a **two-dimensional random variable** (X, Y). For (X, Y) the outcome of a trial is a pair of numbers $X = x$, $Y = y$, briefly $(X, Y) = (x, y)$, which we can plot as a point in the XY-plane.

The **two-dimensional probability distribution** of the random variable (X, Y) is given by the **distribution function**

$$F(x, y) = P\left(X \leqq x, Y \leqq y\right). \tag{1.63}$$

This is the probability that in a trial, X will assume any value not greater than x and in the same trial, Y will assume any value not greater than y. $F(x, y)$ determines the probability distribution uniquely, because extending the analogy we developed previously, $P(a < X \leq b) = F(b) - F(a)$, we now have for a rectangle defined using the following equation:

$$P\left(a_1 < X \leq b_1,\ a_2 < Y \leq b_2\right) = F\left(b_1, b_2\right) - F\left(a_1, b_2\right) - F\left(b_1, a_2\right) + F\left(a_1, a_2\right). \tag{1.64}$$

As before, in the two-dimensional case we shall also have discrete and continuous random variables and distributions.

1.9.1 Discrete Two-Dimensional Distribution

In analogy to the case of a single random variable, we call (X, Y) and its distribution **discrete** if (X, Y) can assume only finitely many or at most countably infinitely many pairs of values (x_1, y_1), $(x_2, y_2), \cdots$ with positive probabilities, whereas the probability for any domain containing none of those values of (X, Y) is zero.

Let (x_i, y_i) be any of those values and let $P\left(X = x_i,\ Y = y_j\right) = p_{ij}$ (where we admit that p_{ij} may be 0 for certain pairs of subscripts i). Then we define the **probability function** $f(x, y)$ of (X, Y) by:

$$f(x, y) = p_{ij} \qquad \text{if} \qquad x = x_i,\ y = y_j \qquad \text{and} \qquad f(x, y) = 0 \quad \text{otherwise;}$$

where, $i = 1, 2, \cdots$ and $j = 1, 2, \cdots$ independently. In analogy to Eq. (1.37), we now have for the distribution function the formula:

$$F(x, y) = \sum_{x_i \leq x} \sum_{y_j \leq y} f\left(x_i, y_j\right).$$

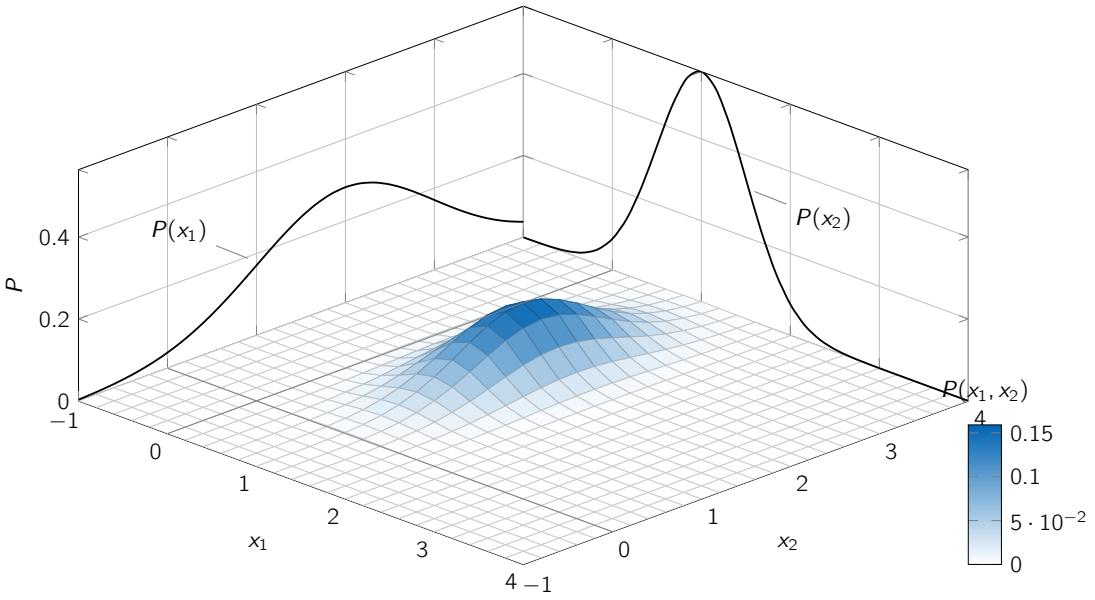


Figure 1.7: Many samples from a bivariate normal distribution. The marginal distributions are shown on the z-axis. The marginal distribution of X is also approximated by creating a histogram of the X coordinates without consideration of the Y coordinates.

Instead of Eq. (1.39), we now have the condition:

$$\sum_i \sum_j f\left(x_i, y_j\right) = 1.$$

1.9.2 Continuous Two-Dimensional Distribution

In analogy to the case of a single random variable, we call (X, Y) and its distribution continuous if the corresponding distribution function $F(x, y)$ can be given by a double integral:

$$F(x, y) = \int_{-\infty}^{y} \int_{-\infty}^{x} f\left(x^*, y^*\right) \, \mathrm{d}x^* \, \mathrm{d}y^* \tag{1.65}$$

whose integrand f, called the **density** of (X, Y), is non-negative everywhere, and is continuous, possibly except on finitely many curves.

From Eq. (1.65) we obtain the probability that (X, Y) assume any value in a rectangle (Fig. 523) given by the formula:

$$P\left(a_1 < X \leq b_1,\ a_2 < Y \leqq b_2\right) = \int_{a_1}^{b_1} \int_{a_1}^{b_1} f(x, y) \, \mathrm{d}x \, \mathrm{d}y$$

1.9.3 Marginal Distributions of a Discrete Distribution

This is a rather natural idea, without counterpart for a single random variable.

It amounts to being interested only in one of the two variables in (X, Y), say, X, and asking for its distribution, called the **marginal distribution** of X in (X, Y). So we ask for the probability $P(X = x,\ Y)$ arbitrary.

Since $(X,\ Y)$ is discrete, so is X. We get its probability function, call it $f_1(x)$, from the probability function $f(x,\ y)$ of $(X,\ Y)$ by summing over y:

$$f_1(x) = P(X = x,\ Y,\ \text{arbitrary}) = \sum_y f(x,\ y) \tag{1.66}$$

where we sum all the values of $f(x,\ y)$ that are not 0 for that x.

From Eq. (1.66) we see that the distribution function of the marginal distribution of X is

$$F_1(x) = P\left(X \leqq x,\ Y,\ \text{arbitrary}\right) = \sum_{x^a \leqq x} f_1(x^*).$$

Similarly, the probability function

$$f_2(y) = P(X\text{arbitrary},\ Y \equiv y) = \sum_x f(x, y)$$

determines the **marginal distribution** of Y in (X, Y). Here we sum all the values of $f(x, y)$ that are not zero for the corresponding y. The distribution function of this marginal distribution is

$$F_2(y) = P(X\text{arbitrary},\ Y \equiv y) = \sum_{y^\star \equiv y} f_2(y^\star).$$

Exercise 1.20: Marginal Distributions of a Discrete Two-Dimensional Random Variable

In drawing 3 cards with replacement from a bridge deck let us consider

$$(X,\ Y) \quad \text{where} \quad X = \text{Number of queens} \quad \text{and} \quad Y = \text{Number of kings or aces.}$$

The deck has 52 cards. These include 4 queens, 4 kings, and 4 aces. Therefore, in a single trial a queen has probability:

$$\frac{4}{52} = \frac{1}{13}$$

and a king or ace:

$$\frac{8}{52} = \frac{2}{13}$$

This gives the probability function of $(X,\ Y)$ as:

$$f(x,\ y) = \frac{3!}{x!y!\,(3-x-y)} \left(\frac{1}{13}\right)^x \left(\frac{2}{13}\right)^y \left(\frac{10}{13}\right)^{3-x-y} \quad \text{where} \quad (x + y \leq 3)$$

and $f(x,\ y) = 0$ otherwise.

1.9.4 Marginal Distributions of a Continuous Distribution

This is conceptually the same as for discrete distributions, with probability functions and sums replaced by densities and integrals. For a continuous random variable $(X,\ Y)$ with density $f(x,\ y)$ we now have the marginal distribution of X in $(X,\ Y)$, defined by the distribution function

$$F_1(x) = P\left(X \leqq x,\ -\infty < Y < \infty\right) = \int_{-\infty}^{x} f_1(x^*)\, dx^*$$

with the density f_1 of X obtained from $f(x, y)$ by integration over y,

$$f_1(x) = \int_{-\infty}^{\infty} f(x, y)\ dy.$$

Interchanging the roles of X and Y, we obtain the **marginal distribution** of Y in (X, Y) with the distribution function

$$F_2(y) = P\left(-\infty < X < \infty, Y \leqq\right) = \int_{-\infty}^{y} f_2(y^*)\ dy^*$$

and density

$$f_2(y) = \int_{-\infty}^{\infty} f(x, y)\ dx.$$

1.9.5 Independence of Random Variables

X and Y in a, discrete or continuous, random variable (X, Y) are said to be **independent** if

$$F(x, y) = F_1(x)F_2(y)$$

holds for all (x, y). Otherwise these random variables are said to be **dependent**. Necessary and sufficient for independence is

$$f(x, y) = f_1(x)f_2(y)$$

for all x and y. Here the f's are the above probability functions if (X, Y) is discrete or those densities if (X, Y) is continuous.

> **Exercise** 1.21: Independence and Dependence
>
> In tossing a 50 cent and a 20 cent coin, with X being the number of heads on the 50 cent, and Y number of heads on the 20 cent, we may assume the values 0 or 1 and are independent.

Extension of Independence to n-Dimensional Random Variables. This will be needed throughout Chapter 2. The distribution of such a random variable $\boldsymbol{X} = (X_1, \cdots, X_n)$ is determined by a **distribution function** of the form

$$F(x_1, \cdots, x_n) = P\left(X_1 \leqq x_1, \cdots, X_n \leqq x_n\right).$$

The random variables $X_1, \cdots, X_n$ are said to be **independent** if

$$F(x_1, \cdots, x_n) = F_1(x_1)F_2(x_2)\cdots F_n(x_n)$$

for all $(x_1, \cdots, x_n)$. Here $F_j(x_j)$ is the distribution function of the marginal distribution of X_j in $\boldsymbol{X}$, that is,

$$F_j(x_j) = P\left(X_j \leqq x_j,\ X_k \text{arbitrary}, k = 1, \cdots, n,\ k \neq j\right).$$

Otherwise these random variables are said to be **dependent**.

1.9.6 Functions of Random Variables

When $n = 2$, we write $X_1 = X$, $X_2 = Y$, $x_1 = x$, $x_2 = y$. Taking a non-constant continuous function $g(x, y)$ defined for all x, y, we obtain a random variable $Z = g(X, Y)$.

> For example, if we roll two (2) dice and X and Y are the numbers the dice turn up in a trial, then $Z = X + Y$ is the sum of those two (2) numbers.

In the case of a discrete random variable (X, Y) we may obtain the probability function $f(z)$ of $Z = g(X, Y)$ by summing all $f(x, y)$ for which $g(x, y)$ equals the value of z considered; thus

$$f(z) = P(Z = z) = \sum_{g(x,y)=z} \sum f(x, y) .$$

Hence the distribution function of Z is

$$F(z) = P(Z \leq z) = \sum_{g(x,y)\leq z} \sum f(x, y) ,$$

where we sum all values of $f(x, y)$ for which $g(x, y) \leqq z$.

In the case of a continuous random variable (X, Y) we similarly have

$$F(z) = P(Z \leqq z) = \iint_{g(x,y)\leq z} f(x, y)\, dx\, dy$$

where for each z we integrate the density $f(x, y)$ of (X, Y) over the region $g(x, y) \leq z$ in the xy-plane, the boundary curve of this region being $g(x, y) = z$.

1.9.7 Addition of Means

The number

$$E(g(X, Y)) = \begin{cases} \sum_x \sum_y g(x, y) f(x, y) & \text{where } X, Y \text{ are discrete} \\ \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y) f(x, y)\ dx\ dy & \text{where } X, Y \text{ are continuous} \end{cases} \tag{1.67}$$

is called the mathematical expectation or, briefly, the **expectation of** $g(X, Y)$. Here it is assumed that the double series converges absolutely and the integral of $|g(x, y)|/(x, y)$ over the y-plane exists[44]. Since summation and integration are linear processes, we have from Eq. (1.67):

[44] meaning it is finite.

$$E(ag(X, Y) + bh(X, Y)) = aE(g(X, Y)) + bE(h(X, Y))$$

An important special case is

$$E(X + Y) = E(X) + E(Y) ,$$

and by induction we have the following result.

Theory 1.16: Addition of Means

The mean (expectation) of a sum of random variables equals the sum of the means (expectations), that is,

$$E\left(X_1 + X_2 + \cdots + X_n\right) = E\left(X_1\right) + E\left(X_2\right) + \cdots + E\left(X_n\right).$$

We can also deduce the following statement:

Theory 1.17: Multiplication of Means

The mean (expectation) of the product of independent random variables equals the product of the means (expectations), that is,

$$E\left(X_1 X_2 \cdots X_n\right) = E\left(X_1\right) E\left(X_2\right) \cdots E\left(X_n\right).$$

and in the continuous case the proof of the relation is similar[45].

[45] This is left as an exercise to the reader.

1.9.8 Addition of Variances

A final matter to cover is how we can sum up variances. Similar to before, let $Z = X + Y$ and denote the mean and variance of Z by μ and σ^2.

Then we first have:

$$\sigma^2 = E\left(\left[Z - \mu\right]^2\right) = E\left(Z^2\right) - \left[E\left(Z\right)\right]^2$$

From (24) we see that the first term on the right equals

$$E\left(Z^2\right) = E\left(X^2 + 2XY + Y^2\right) = E\left(X^2\right) + 2E\left(XY\right) + E\left(Y^2\right).$$

For the second term on the right we obtain from Theorem 1

$$\left[E\left(Z\right)\right]^2 = \left[E\left(X\right) + E\left(Y\right)\right]^2 = \left[E\left(X\right)\right]^2 + 2E\left(X\right)E\left(Y\right) + \left[E\left(Y\right)\right]^2$$

By substituting these expressions into the formula for σ^2 we have

$$\begin{aligned}\sigma^2 = E\left(X^2\right) - \left[E\left(X\right)\right]^2 + E\left(Y^2\right) - \left[E\left(Y\right)\right]^2 \\ + 2\left[E\left(XY\right) - E\left(X\right)E\left(Y\right)\right].\end{aligned}$$

the expression in the first line on the right is the sum of the variances of X and Y, which we denote by σ_1^2 and σ_2^2, respectively.

The quantity in the second line (except for the factor 2) is:

$$\sigma_{XY} = E\left(XY\right) - E\left(X\right)E\left(Y\right), \tag{1.68}$$

and is called the **covariance** of X and Y. Consequently, our result is

$$\sigma^2 = \sigma_1^2 + \sigma_2^2 + 2\sigma_{XY}.$$

If X and Y are independent, then

$$E\left(XY\right) = E\left(X\right)E\left(Y\right);$$

hence $\sigma_{XY} = 0$, and

$$\sigma^2 = \sigma_1^2 + \sigma_2^2$$

Extension to more than two variables gives the basic

Theory 1.18: Addition of Variances

The variance of the sum of independent random variables equals the sum of the variances of these variables.

Chapter 2

Statistical Methods

Table of Contents

2.1 Introduction 41
2.2 Point Estimation of Parameters 44
2.3 Confidence Intervals 47
2.4 Testing of Hypotheses and Making Decisions 54
2.5 Goodness of Fit 60
2.6 Regression and Correlation 63

2.1 Introduction

Statistical[1] methods consists of a wide range of tools for designing and evaluating random experiments to obtain information about practical problems:

[1] The word is derived from New Latin *statistica* or *statisticus* ("of the state")

> such as exploring the relation between iron content and density of iron ore, the quality of raw material or manufactured products, the efficiency of air-conditioning systems, the performance of certain cars, the effect of advertising, the reactions of consumers to a new product, etc.

Therefore, the diameter of screws is a random variable X and we have non-defective screws, with diameter between given tolerance limits, and defective screws, with diameter outside those limits. We can ask for the distribution of X, for the percentage of defective screws to be expected, and for necessary improvements of the production process.

Samples are selected from populations:

> 20 screws from 1000 screws, 100 of 5000 voters, 8 behaviours in a wildlife observation.

as inspecting the entire sample, would be expensive, time-consuming, impossible or even senseless.[2]

[2] It would be inconceivable for a company who produces over a billion light bulbs to test all their products. That is why we have return policies.

To obtain a meaningful sense of information, samples must be **random selections**. Each of the 1000 screws must have the same chance of being sampled,[3] at least approximately. Only then will the sample mean:

[3] of being drawn when we sample.

$$\overline{x} = \frac{1}{20}\left(x_1 + \cdots x_{20}\right) \qquad \text{where} \qquad n = 20,$$

will be a good approximation of the population mean μ, and the accuracy of the approximation will generally improve with increasing n, as we shall see.

> This is also applicable to other statistical quantities such as standard deviation, variance, etc.

Independent sample values will be obtained in experiments with an infinite sample space S certainly for the normal distribution. This is also true in sampling with replacement. It is approximately true in drawing small samples from a large finite population.[4] However, if we sample without replacement from a small population, the effect of dependence of sample values may be considerable.

[4] for instance, 5 or 10 of 1000 items.

Random numbers help in obtaining samples that are in fact random selections. This is sometimes not easy to accomplish as there are numerous subtle factors which can bias sampling.[5] Random numbers can be obtained from a **random number generator**

[5] Such as by personal interviews, by poorly working machines, by the choice of non-typical observation conditions, etc.

> It is important to state that the numbers generated by a computer are **NOT** truly random, as are calculated by a tricky formula that produces numbers that do have practically all the essential features of true randomness. Because these numbers eventually repeat, they must not be used in cryptography, for example, where true randomness is required.

> **Exercise** 2.1: Generating Random Numbers
>
> To select a sample of size $n = 10$ from 80 given ball bearings, we number the bearings from 1 to 80. We then let the generator randomly produce 10 of the integers from 1 to 80 and include the bearings with the numbers obtained in our sample, for example,
>
> 44 55 57 03 61 51 68 22 34 77
>
> or whichever number pops up in your head.[6]

[6] Of course in a professional setting you can't just write numbers like that as there is also a pattern when we make successive random number. Before the prevalence of computers there used to be books containing random numbers which people consulted.

Representing and processing data were considered in the previous chapter in connection with **frequency distributions**. These are the empirical counterparts of probability distributions and helped motivating axioms and properties in probability theory. The new aspect in this chapter is **randomness**:

i.e., the data are samples selected **randomly** from a population.

Accordingly, we can already use the plots we have used in probability, such as stem-and-leaf plots, box plots, and histograms.

> In this chapter, the mean $\overline{x}$ we defined previously, will now be referred as **sample mean**.

$$\overline{x} = \frac{1}{n}\sum_{j=1}^{n} x_j = \frac{1}{n}\left(x_1 + x_2 + \cdots + x_n\right). \tag{2.1}$$

We call n the **sample size**, and similar to mean, the variance s^2 is called the **sample variance**:

$$s^2 = \frac{1}{n-1}\sum_{j=1}^{n}\left(x_j - \overline{x}\right)^2 = \frac{1}{n-1}\left[\left(x_1 - \overline{x}\right)^2 + \cdots + \left(x_n - \overline{x}\right)^2\right], \tag{2.2}$$

and its positive square root, s is the **sample standard deviation**.

> $\overline{x}$, s^2, s are called **sample parameters** of a dataset.

2.2 Point Estimation of Parameters

Before we dive deep into statistics, let's spend some time to learn the most basic practical tasks in statistics and corresponding statistical methods to accomplish them. The first is point estimation of parameters, that is, of quantities appearing in distributions:

such as p in the binomial distribution and μ and σ in the normal distribution.

A **point estimate** of a parameter is a number,[7] which is computed from a given sample and serves as an approximation of the unknown exact value of the parameter of the population. An interval estimate is an interval[8] obtained from a sample.

[7] which is a point on the real line.

[8] also known as confidence interval.

> Estimation of parameters is of great practical importance in many applications.

As an approximation[9] of the mean of a population we may take the mean $\overline{x}$ of a corresponding sample. This gives the estimate $\hat{\mu} = \overline{x}$ for μ, that is,

[9] to describe something which is an approximation or an educated guess, we use hat notation. This is applicable for fields in statistics, machine learning or data science.

$$\hat{\mu} = \overline{x} = \frac{1}{n}\left(x_1 + \cdots + x_n\right) \tag{2.3}$$

where n is the sample size. Similarly, an estimate $\hat{\sigma}^2$ for the variance of a population is the variance s^2 of a corresponding sample, that is:

$$\hat{\sigma}^2 = s^2 = \frac{1}{n-1}\sum_{j=1}^{n}\left(x_j - \overline{x}\right)^2. \tag{2.4}$$

As can be seen, Eq. (2.3) and Eq. (2.4) are estimates of parameters for distributions in which μ or σ^2 appear explicitly as parameters, such as the normal and Poisson distributions.

For the binomial distribution, $p = \mu/n$. From Eq. (2.3) we obtain for p the estimate:

$$\hat{p} = \frac{\overline{x}}{n}. \tag{2.5}$$

It is important to mention Eq. (2.3) is a special case of the so-called **method of moments**. Here, the parameters to be estimated are expressed in terms of the moments of the distribution. In the resulting formulas, those moments of the distribution are replaced by the corresponding moments of the sample, which gives the estimates. Here the k^{th} moment of a sample $x_1, \cdots, x_n$ is:

$$m_k = \frac{1}{n}\sum_{j=1}^{n} x_j^k. \tag{2.6}$$

2.2.1 Maximum Likelihood Method

Another method for obtaining estimates is

the so-called **maximum likelihood method** conceived by To explain it, we consider a discrete (or continuous) random variable X whose probability function (or density) $f(x)$ depends on a single parameter θ. We take a corresponding sample of n independent values $x_1, \cdots, x_n$. Then in the discrete case the probability given a sample of size n consists precisely of those n values is

$$I = f(x_1)\, f(x_2) \cdots f(x_n)\,. \tag{2.7}$$

In the continuous case the probability that the sample consists of values in the small intervals $x_j \leq x \leqq x_j + \Delta x\,(j = 1, 2, \cdots, n)$ is

$$f(x_1)\,\Delta x f(x_2)\,\Delta x \cdots f(x_n)\,\Delta x = I\,(\Delta x)^n \tag{2.8}$$

As $f\left(x_j\right)$ depends on θ, the function I in Eq. (2.8) given by Eq. (2.7) depends on $x_1, \cdots, x_n$ and θ.

> We imagine $x_1, \cdots, x_n$ to be given and fixed.

Then I is a function of θ, which is called the **likelihood function**. The basic idea of the maximum likelihood method is quite simple, as follows.

> We choose an approximation for the unknown value of θ for which I is as large as possible.

If I is a differentiable function of θ, a necessary condition for I to have a maximum in an interval[10] is

[10] not at the boundary.

$$\frac{\partial I}{\partial \theta} = 0 \tag{2.9}$$

A solution of Eq. (2.9) depending on $x_1, \cdots, x_n$ is called a **maximum likelihood estimate** for θ. We may replace Eq. (2.9) by:

$$\frac{\partial \ln I}{\partial \theta} = 0 \tag{2.10}$$

as $f\left(x_j\right) > 0$, a maximum of I is in general positive, and $\ln I$ is a monotone increasing function of I. This often simplifies calculations.

Several Parameters

If the distribution of X involves r parameters $\theta_1, \cdots, \theta_r$, then instead of Eq. (2.9) we have the r conditions $\partial \ln I/\partial \theta_1, \cdot, \partial \ln I/\partial \theta_r = 0$, and instead of Eq. (2.10) we have:

$$\frac{\partial \ln I}{\partial \theta_1} = 0, \cdots, \frac{\partial \ln I}{\partial \theta_r} = 0. \tag{2.11}$$

Exercise 2.2: Maximum Likelihood of Gaussian Distribution

Find maximum likelihood estimates for $\theta_1 = \mu$ and $\theta_2 = \sigma$ in the case of the normal distribution.

Solution

We obtain the likelihood function:

$$l = \left(\frac{1}{\sqrt{2\pi}}\right)^n \left(\frac{1}{\sigma}\right)^n e^{-h} \qquad \text{where} \qquad h = \frac{1}{2\sigma^2}\sum_{j=1}^{n}\left(x_j - \mu\right)^2.$$

Taking logarithms, we have

$$\ln l = -n\ln\sqrt{2\pi} - n\ln\sigma - h.$$

The first equation in Eq. (2.11) is $\frac{\partial \ln l}{\partial \mu} = 0$, written out:

$$\frac{\partial \ln l}{\partial \mu} = -\frac{\partial h}{\partial \mu} = \frac{1}{\sigma^2}\sum_{j=1}^{n}\left(x_j - \mu\right) = 0, \qquad \text{therefore} \qquad \sum_{j=1}^{n} x_j - n\mu = 0.$$

The solution is the desired estimate $\hat{\mu}$ for μ: we find

$$\hat{\mu} = \frac{1}{n}\sum_{j=1}^{n} x_j = \overline{x}.$$

The second equation in Eq. (2.11) is $\frac{\partial \ln l}{\partial \sigma} = 0$, written out

$$\frac{\partial \ln l}{\partial \sigma} = -\frac{n}{\sigma} - \frac{\partial h}{\partial \sigma} = -\frac{1}{\sigma} + \frac{1}{\sigma^3}\sum_{j=1}^{n}\left(x_j - \mu\right)^2 = 0.$$

Replacing μ by $\hat{\mu}$ and solving for σ^2, we obtain the estimate:

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{j=1}^{n}\left(x_j - \overline{x}\right)^2 \quad \blacksquare$$

Exercise 2.3: Estimating Poisson Parameters

Consider a Poisson distribution with Probability Mass Function (PMF):

$$f\left(x|\mu\right) = \frac{e^{-\mu}\mu^{x}}{x!}, \qquad \text{where} \qquad x = 1,\ 2,\ 3,\ \cdots$$

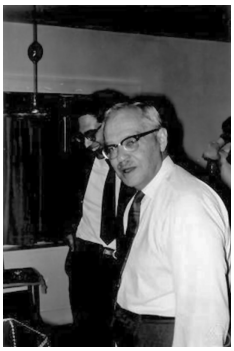
Suppose a random sample $x_1,\ x_2 \cdots x_n$ is taken from distribution. What is the maximum likelihood estimate of μ?

Solution

The likelihood function is:

$$L\left(x_1,\ x_2 \cdots x_n;\mu\right) = \prod_{i=1}^{n} f\left(x_i|\mu\right) = \frac{e^{-n\mu}\mu^{\sum_{i=1}^{n} x_i}}{\prod_{i=1}^{n} x_i!}$$

2.3 Confidence Intervals



Confidence intervals[11] for an unknown parameter θ of some distribution (e.g., $\theta = \mu$) are intervals $\theta_1 \leq \theta \leq \theta_2$ which contain θ, not with certainty but with a high probability γ, which we can choose.[12] Such an interval is calculated from a sample. $\gamma = 95\%$ means probability $1 - \gamma = 5\% = 1/20$ of being wrong.[13] Instead of writing $\theta_1 \leq \theta \leq \theta_2$, we denote this more **distinctly** by writing:

$$\text{CONF}_\gamma \ \{\theta_1 \leq \theta \leq \theta_2\} \tag{2.12}$$

Such a special symbol, CONF, seems worthwhile to avoid the misunderstanding that θ **must** lie between θ_1 and θ_2.

γ is called the **confidence level**, and θ_1 and θ_2 are called the **lower** and **upper confidence limits**, respectively and depend on the γ value. The larger we choose γ, the smaller is the error probability $1 - \gamma$, but the longer is the confidence interval.

If $\gamma \to 1$, then its length goes to infinity.

The choice of γ depends on the kind of application.

> In taking no umbrella, a 5% chance of getting wet is **NOT** a problem. In a medical decision of life or death, a 5% chance of being wrong may be too large and a 1% chance of being wrong ($\gamma = 99\%$) may be more desirable.

Confidence intervals are more valuable than point estimates. We can take the midpoint of Eq. (2.12) as an approximation of θ and half the length of Eq. (2.12) as an error bound.[14]

θ_1 and θ_2 in Eq. (2.12) are calculated from a sample $x_1, \cdots, x_n$. These are n observations of a random variable X. Now comes a **standard trick**.

We regard $x_1, \cdots, x_n$ as single observations of n random variables $X_1, \cdots, X_n$[15]. Then $\theta_1 = \theta_1(x_1, \cdots, x_n)$ and $\theta_2 = \theta_2(x_1, \cdots, x_n)$ in Eq. (2.12) are observed values of two random variables $\Theta_1 = \Theta_1(X_1, \cdots, X_n)$ and $\Theta_2 = \Theta_2(X_1, \cdots, X_n)$. The condition Eq. (2.12) involving γ can now be written

$$P\left(\Theta_1 \leqq \theta \leqq \Theta_2\right) = \gamma. \tag{2.13}$$

Let us see what all this means in concrete practical cases.

In each case in this section we shall first state the steps of obtaining a confidence interval in the form of a table, then consider a typical example, and finally justify those steps theoretically.

[11] Established by Jerzy Neyman. He proposed and studied randomised experiments in 1923. Furthermore, his paper *On the Two Different Aspects of the Representative Method: The Method of Stratified Sampling and the Method of Purposive Selection*, given at the Royal Statistical Society on 19 June 1934, was the groundbreaking event leading to modern scientific sampling. He introduced the confidence interval in his paper in 1937. Another noted contribution is the Neyman-Pearson lemma, the basis of hypothesis testing.

[12] 95% and 99% are popular

[13] one of about 20 such intervals will **NOT** contain θ.

[14] not in the strict sense of numerical means, but except for an error whose probability we know.

[15] with the same distribution, namely, that of X

For Mean with known Variance in Normal Distribution

The method of tackling is this problem is as follows:

[16] 95%, 99%, depending on the application.

1. Choose a confidence level for γ .[16]
2. Determine the corresponding c:

γ	0.90	0.95	0.99	0.999
c	1.645	1.960	2.576	3.291

3. Compute the mean $\overline{x}$ of the sample $x_1, \cdots, x_n$.
4. Compute $k = c\sigma/\sqrt{n}$. The confidence interval for μ is

$$\text{CONF}_\gamma \{\overline{x} - k \leqq \mu \leqq \overline{x} + k\}. \tag{2.14}$$

Exercise 2.4: Confidence Interval for mean with known variance in Normal Distribution

Determine 95% confidence interval for the mean of a normal distribution with variance $\sigma^2 = 9$, using a sample of $n = 100$ values with mean $\overline{x} = 5$.

Solution

1. First we define γ as 0.95.
2. Then looking at the table find the corresponding c which equals 1.960.
3. $\hat{x} = 5$ is given.
4. We need:

$$k = c\frac{\sigma}{\sqrt{n}} = 1.960\frac{3}{\sqrt{100}} = 0.588$$

Therefore

$$\hat{x} - k = 4.412 \quad \text{and} \quad \hat{x} + k = 5.588$$

and the confidence interval is:

$$\text{CONF}_{0.95}\{4.412 \leq \mu \leq 5.588\}$$ ■

Theory 2.19: Sum of Independent Normal Random Variables

Let $X_1, \cdots, X_n$ be independent normal random variables each of which has mean μ and variance σ^2.

Then the following holds:

a. The sum $X_1 + \cdots + X_n$ is normal with mean $n\mu$ and variance $n\sigma^2$.

b. The following random variable $\overline{X}$ is normal with mean μ and variance σ^2/n.

$$\overline{X} = \frac{1}{n}\left(X_1 + \cdots + X_n\right)$$

c. The following random variable Z is normal with mean 0 and variance 1.

$$Z = \frac{\overline{X} - \mu}{\sigma/\sqrt{n}}$$

Exercise 2.5: Sample Size Needed for a Confidence Interval of Prescribed Length

How large must n be in Example 2.3 if we want to obtain a 95% confidence interval of length $L = 0.4$?

Solution

The interval in Eq. (2.14) has the length:

$$L = 2k = 2c\sigma/\sqrt{n}$$

Solving for n, we obtain

$$n = \left(\frac{2c\sigma}{L}\right)^2$$

In the present case the answer is:

$$n = \left(\frac{2 \times 1.96 \times 3}{0.4}\right)^2 \approx 870 \quad \blacksquare$$

For Mean of the Normal Distribution with Unknown Variance

In practice σ^2 is frequently unknown. Then the method described previously does not help and the whole theory changes, although the steps of determining a confidence interval for μ remain quite similar. They will be explained in the following section. We see that k differs from previous method, namely, the sample standard deviation s has taken the place of the unknown standard deviation σ of the population. And c now depends on the sample size n and must be determined from Table 49 given in the Appendix. That table lists values z for given values of the distribution function.

$$F(z) = K_m \int_{-\infty}^{x} \left(1 + \frac{u^2}{m}\right)^{-(m+1)/2} du \tag{2.15}$$

of the t-distribution. Here, m $(= 1, 2, \cdots)$ is a parameter, called the **number of degrees of freedom** of the distribution.[17] In the present case, $m = n - 1$; see Table 25.2. The constant K_m is such that $F(\infty) = 1$. By integration it turns out that

[17] abbreviated d.f.

$$K_m = \frac{\Gamma\left(\frac{1}{2}m + \frac{1}{2}\right)}{\sqrt{m\pi}\,\Gamma\left(\frac{1}{2}m\right)},$$

where Γ is the gamma function.

The method of tackling is this problem is as follows:

1. Choose a confidence level γ .[18]

[18] 95%, 99%, or the like.

2. Determine the solution c of the equation,

$$F(c) = \frac{1}{2}(1 + \gamma)$$

   from the table of the t-distribution with $n - 1$ degrees of freedom

3. Compute the mean $\overline{x}$ and the variance s^2 of the sample $x_1, \cdots, x_n$.

4. Compute $k = cs/\sqrt{n}$. The confidence interval is:

$$\text{CONF}_\gamma\{\overline{x} - k \leqq \mu \leqq \overline{x} + k\}.$$

This illustrates that Table 25.1 (which uses more information, namely, the known value of σ^2) yields shorter confidence intervals than Table 25.2. This is confirmed in, which also gives an idea of the gain by increasing the sample size.

Exercise 2.6: Confidence Interval for Mean of Normal Distribution with Unknown Variance

The five (5) independent measurements of flash point of Diesel oil (D-2) gave the values (in °F):

144 147 146 142 144

If we assume normality, determine a 99% confidence interval for the mean.

Solution

1. $\gamma = 0.99$ is required.
2. $F(c) = \frac{1}{2}(1+\gamma) = 0.99$ and looking at the reference table with $n-1 = 4$ d.f. giving $c = 4.60$.
3. $\overline{x} = 144.6$ and $s = 3.8$,
4. $k = \sqrt{3.8} \times 4.60/\sqrt{5} = 4.01$. Therefore the confidence interval is:

$$\text{CONF}_{0.99}\{140.5 \leq \mu \leq 148.7\} \quad \blacksquare$$

If the variance σ^2 were known and equal to the sample variance s^2, thus $\sigma^2 = 3.8$, then the Reference Table would give:

$$k = \frac{c\sigma}{\sqrt{n}} = 2.576\frac{\sqrt{3.8}}{\sqrt{3}} = 2.25$$

and

$$\text{CONF}_{0.99}\{140.5 \leq \mu \leq 148.7\}$$

We see that the present interval is almost twice as long as that with a known variance $\sigma^2 = 3.8$.

[19] William Gosset (13 June 1876 - 16 October 1937) was an English statistician, chemist and brewer who served as Head Brewer of Guinness and Head Experimental Brewer of Guinness and was a pioneer of modern statistics. He published is results under the pen name student.

Theory 2.20: Student's t-Distribution

Let $X_1, \cdots, X_n$ be independent normal random variables with the same mean μ and the same variance σ^2. Then the random variable:

$$T = \frac{\overline{X} - \mu}{S/\sqrt{n}}$$

has a t-distribution[19] with $n-1$ degrees of freedom (d.f.); here $\overline{X}$ is given by (4) and

$$S^2 = \frac{1}{n-1}\sum_{j=1}^{n}\left(X_j - \overline{X}^2\right).$$

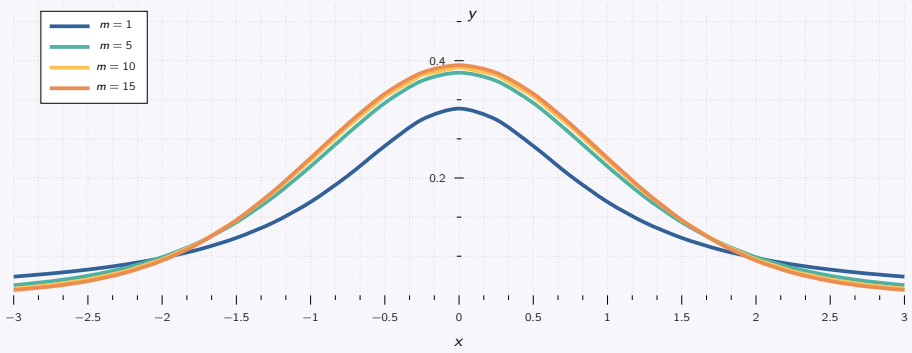


Figure 2.1: The student-t distribution with different degrees of freedom m.

For the Variance of the Normal Distribution

The method for calculating the confidence interval is similar to the previous methods, with slight change in some steps which are as follows:

1. Choose a confidence level γ.[20]

[20] as usual this can be 95%, 99%, or the like.

2. Determine solutions c_1 and c_2 of the equations:

$$F(c_1) = \frac{1}{2}(1-\gamma) \qquad \text{and} \qquad F(c_2) = \frac{1}{2}(1+\gamma).$$

where the necessary values are calculated from the table of the chi-square distribution with $n-1$ degrees of freedom.

3. Compute $(n-1)\,s^2$, where s^2 is the variance of the sample $x_1, \cdots, x_n$.

4. Compute $k_1 = (n-1)s^2/c_1$ and $k_2 = (n-1)s^2/c_2$. The confidence interval is

$$\text{CONF}_\gamma\{k_2 \cong \sigma^2 \cong k_1\}. \tag{2.16}$$

Exercise 2.7: Confidence Interval for the Variance of the Normal Distribution

Determine a 95% confidence interval Eq. (2.16) for the variance, using Table 25.3 and a sample (tensile strength of sheet steel in $\mathrm{kg\,mm^{-2}}$, rounded to integer values)

89 84 87 81 89 86 91 90 78 89 87 99 83 89

Solution

1. $\gamma = 0.95$ is required.
2. For $n-1 = 13$ we find
$$c_1 = 5.01 \qquad \text{and} \qquad c_2 = 24.74.$$
3. $13s^2 = 326.9$
4. $13s^2/c_1 = 65.25 \qquad \text{and} \qquad 13s^2/c_2 = 13.21$
5. This makes the confidence interval as:
$$\text{CONF}_{0.09}\left\{13.21 \le \sigma^2 \le 65.25\right\}.$$

This is rather large, and for obtaining a more precise result, one would need a much larger sample ■.

Theory 2.21: Chi-Square Distribution

Under the assumptions in Theorem 2 the random variable

$$Y = (n-1)\,\frac{S^2}{\sigma^2}$$

with S_2 given by (12) has a chi-square distribution with $n-1$ degrees of freedom.

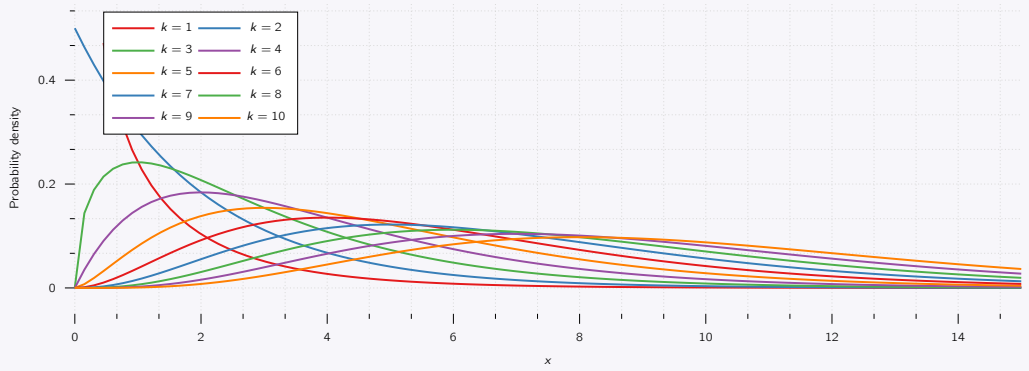


Figure 2.2: Chi-square distribution with different degrees of freedom.

The chi-squared distribution, which can be seen in **Fig.** 2.2 is used primarily in **hypothesis testing**, and to a lesser extent for confidence intervals for population variance when the underlying distribution is normal. Unlike more widely known distributions such as the normal distribution and the exponential distribution, the chi-squared distribution is not as often applied in the direct modelling of natural phenomena.

The primary reason for which the chi-squared distribution is extensively used in hypothesis testing is its relationship to the normal distribution. Many hypothesis tests use a test statistic, such as the t-statistic in a t-test. For these hypothesis tests, as the sample size n increases, the sampling distribution of the test statistic approaches the normal distribution.[21] Because the test statistic (t) is asymptotically normally distributed, provided the sample size is sufficiently large, the distribution used for hypothesis testing may be approximated by a normal distribution.

[21] This is the result of the central limit theorem.

So wherever a normal distribution could be used for a hypothesis test, a chi-squared distribution could be used.

Confidence Internals for Parameters of Other Distributions

The methods in mentioned previously for confidence intervals for μ and σ^2 are designed for the normal distribution. We will see it here that they can also be applied to other distributions if we **use large samples**.

We know that if $X_1, \cdots, X_n$ are independent random variables with the same mean μ and the same variance σ^2, then their sum $Y_n = X_1 + \cdots + X_n$ has the following properties:

- Y_n has the mean $n\mu$ and the variance $n\sigma^2$,
- If those variables are normal, then Y_n is normal.

If those random variables are **not normal**, then second property is **NOT** applicable. However, for large n the random variable Y_n is still approximately normal.

This follows from the central limit theorem, which is one of the most fundamental results in probability theory.

Theory 2.22: Central Limit Theorem

Let $X_1, \cdots, X_n$ be independent random variables having the same distribution function and therefore the same mean μ and

variance σ^2. Let $Y_n = X_1 + \cdots + X_n$, then the random variable

$$Z_n = \frac{Y_n - n\mu}{\sigma\sqrt{n}}$$

is **asymptotically normal** with mean 0 and variance 1. That is, the distribution function $F(x)$ of Z_n satisfies:

$$\lim_{n\to\infty} F(x) = \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-u^2/2}\, du.$$

This theorem basically boils down to the following statement:

> Under appropriate conditions, the distribution of a normalised version of the sample mean converges to a standard normal distribution. This holds even if the original variables themselves are not normally distributed.

Therefore, when applying the previous confidence interval methods to a non-normal distribution, we must use sufficiently large samples.

As a rule of thumb, if the sample indicates that the skewness of the distribution is small, use at least $n = 20$ for the mean and at least $n = 50$ for the variance.

[22] The modern development of tests are generally attributed to Egon Sharpe Pearson and Neymar whom was mentioned previously. Egon Sharpe was one of three children of Karl Pearson and Maria, and, like his father, a British statistician. He is known throughout the world as co-author of the Neyman-Pearson theory of testing statistical hypotheses, and responsible for many important contributions to problems of statistical inference and methodology, especially in the development and use of the likelihood ratio criterion.

[23] also called null hypothesis.

[24] or alternative hypothesis

2.4 Testing of Hypotheses and Making Decisions

The ideas of confidence intervals and of tests[22] are the two (2) most important ideas in modern statistics. In a statistical test we make inference from sample to population through testing a **hypothesis**, resulting from experience or observations, from a theory or a quality requirement, and so on.

In many cases the result of a test is used as a basis for a **decision**:

> to buy, or not to buy a certain model of car, depending on a test of the fuel efficiency ($\mathrm{km\,L^{-1}}$), or, to apply some medication, depending on a test of its effect; to proceed with a marketing strategy, depending on a test of consumer reactions, etc.

As with most abstract mathematical concepts, it is better to explain such a test in terms of a typical example and then introduce the corresponding standard notions of statistical testing.

Exercise 2.8: Test of a Hypothesis

Let's say we want to buy 100 coils of a certain kind of wire, provided we can verify the manufacturer's claim that the wire has a specific strength of $\mu = \mu_0 = 200$ kN m kg^{-1}, or more.

This is a test of the hypothesis:[23] $\mu = \mu_0 = 200$. We shall **NOT** buy the wire if the statistical tests shows that actually $\mu = \mu_1 < \mu_0$, the wire is weaker, the claim does **NOT** hold. μ_1 is called the **alternative** of the test.[24] We shall **accept** the hypothesis if the test suggests that it is true, except for a small error probability α, called the **significance level** of the test.

Otherwise we reject the hypothesis.

Hence α is the probability of rejecting a hypothesis although it is true. The choice of α is up to us, 5% and 1% are popular values.

For the test we need a sample. We randomly select 25 coils of the wire, cut a piece from each coil, and determine the breaking limit experimentally. Suppose that this sample of $n = 25$ values of the breaking limit has the mean $\overline{x} = 197$ kN m kg^{-1}, which is somewhat less than the claim, and the standard deviation $s = 6$ kN m kg^{-1}.

At this point we could only speculate when this difference $197 - 200 = -3$ is due to randomness, is a chance effect, or whether it is **significant**, due to the actual inferior quality of the wire. To continue beyond speculation requires probability theory, as follows.

We assume that the blocking limit is normally distributed. Then

$$T = \frac{\overline{X} - \mu_0}{S/\sqrt{n}}$$

with $\mu = \mu_0$ has a **t-distribution** with $n - 1$ degrees of freedom ($n - 1 = 24$ for our sample). Also $\overline{x} = 197$ and $s = 6$ are observed values of $\overline{X}$ and S to be used later. We can now choose a significance level, say, $\alpha = 95\%$ From the Reference Table, we then obtain a critical value c such that $P(T \leq c) = \alpha = 5\%$. For $P(T \leq \overline{c}) = 1 - \alpha = 95\%$ the table gives $\overline{c} = 1.71$, so that $c = -\overline{c} = -1.71$ because of the symmetry of the distribution shown in **Fig.** 2.3.

We now reason as follows–this is the crucial idea of the test. If the hypothesis is true, we have a chance of only α (= 5%) that we observe a value t of T (calculated from a sample) that will fall between $-\infty$ and -1.71. Hence, if we nevertheless do observe such a t, we start that the hypothesis cannot be true and we reject it.

A simple calculation gives:

$$T = \frac{(107 - 200)}{6/\sqrt{25}} = -2.5,$$

as an observed value of T. Since $-2.5 < -1.71$, we reject the hypothesis, the manufacturer's claim, and accept the alternative result of $\mu = \mu_4 < 200$, which means the wire seems to be weaker than claimed ■

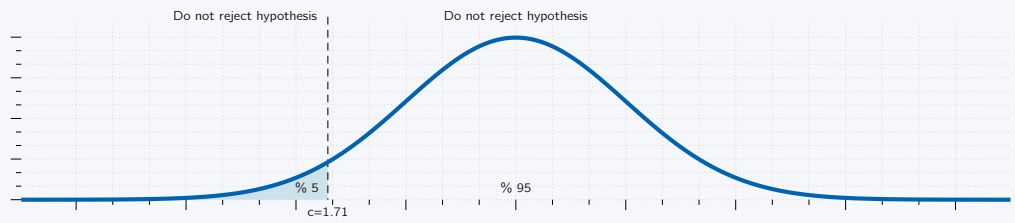


Figure 2.3: The t-distribution used in Example 2.8.

This aforementioned example perfectly captures the steps of a test:

1. Formulate the **hypothesis** $\theta = \theta_0$ to be tested. In our previous example it is $\theta_0 = \mu_0$.
2. Formulate an **alternative** $\theta = \theta_1$, which in our example is $\theta_1 = \mu_1$.
3. Choose a **significance level** α with values such as 5%, 1%, or, 0.1%.
4. Use a random variable $\hat{\Theta} = g(X_1, \cdots, X_n)$ whose distribution depends on the hypothesis and on the alternative, and this distribution is known in both cases.

   Determine a critical value c from the distribution of $\hat{\Theta}$, assuming the hypothesis to be true. In the example, $\hat{\Theta} = T$, and c is, obtained from $P(T \leq c) = \alpha$.
5. Use a sample $x_1, \cdots, x_n$ to determine an observed value $\hat{\theta} = g(x_1, \cdots, x_n)$ of $\hat{\Theta}$, where in our example it is t.
6. Accept or reject the hypothesis, depending on the size of $\hat{\theta}$ relative to c.

There are two (2) important facts require further discussion and careful attention.

1. The choice of an alternative. In the example, $\mu_1 < \mu_0$, but other applications may require $\mu_1 > \mu_0$ or $\mu_1 \neq \mu_0$.
2. Addressing errors. We know that α, the significance level of the test, is the probability of reflecting a true hypothesis. And we shall discuss the probability β of accepting a false hypothesis.

One-Sided and Two-Sided Alternatives

Let θ be an unknown parameter in a distribution, and suppose we want to test the hypothesis $\theta = \theta_0$.

Then there are three (3) main kinds of alternatives, namely,

$$\theta > \theta_0 \tag{2.17}$$

$$\theta < \theta_0 \tag{2.18}$$

$$\theta \neq \theta_0 \tag{2.19}$$

Here Eq. (2.17), and Eq. (2.18) are **one-sided alternatives**, and Eq. (2.19) is a two-sided alternative.

[25] or called the critical region

We call rejection region[25] the region such that we reject the hypothesis if the observed value in the test falls in this region. In [1] the critical c lies to the right of θ_0 because so does the alternative. Hence the rejection region extends to the right. This is called a right-sided test. In [2] the critical c lies to the left of θ_0 (as in Example 1), the rejection region extends to the left, and we have a left-sided test. These are one-sided tests. In [3]

All three kinds of alternatives occur in practical problems. For example, Eq. (2.17) may arise if θ_0 is the maximum tolerable inaccuracy of a voltmeter or some other instrument. Alternative Eq. (2.18) may occur in testing strength of material, as in Example A. Finally, θ_0 in Eq. (2.19) may be the diameter of axle-shafts, and shafts that are too thin or too thick are equally undesirable, so that we have to watch for deviations in both directions.

2.4.1 Errors in Tests

Tests always involve risks of making false decisions:

I Rejecting a true hypothesis (Type I error)

- α = Probability of making a Type I error.

II Accepting a false hypothesis (Type II error).

- β = Probability of making a Type II error.

Clearly, we cannot avoid these errors.

> No absolutely certain conclusions about populations can be drawn from samples.

[26] e.g., a small risk $\alpha = \%1$ is used if it is a matter of life or death.

But we show there are ways and means of choosing suitable levels of risks, that is, of values α and β. The choice of α depends on the nature of the problem.[26]

Let us discuss this systematically for a test of a hypothesis $\theta = \theta_0$ against an alternative that is a single number θ_1, for simplicity. We let $\theta_1 > \theta_0$, so that we have a right-sided test. For a left-sided or a two-sided test the discussion is quite similar.

[27] as in the upper part of Fig. 533, by methods discussed below

We choose a critical $c > \theta_0$[27]. From a given sample $x_1, \cdots, x_n$ we then compute a value:

$$\hat{\theta} = g\left(x_1, \cdots, x_n\right)$$

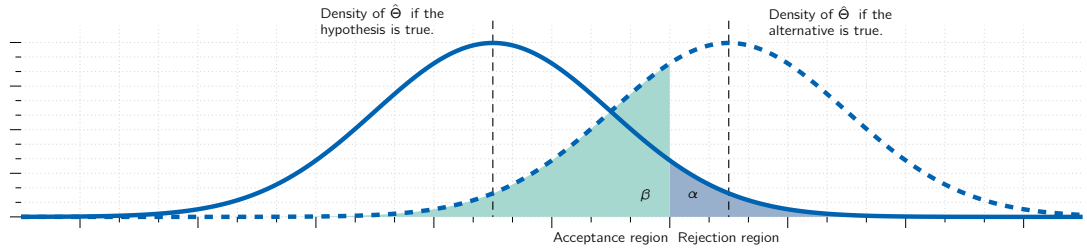


Figure 2.4: Illustration of Type I and II errors in testing a hypothesis $\theta = \theta_0$ against an alternative $\theta = \theta_0$.

with a suitable g.

> whose choice will be a main point of our further discussion; for instance, take $g = (x_1 + \cdots + x_n)/n$ in the case in which θ is the mean.

If $\hat{\theta} > c$, we reject the hypothesis. If $\hat{\theta} \leq c$, we accept it. Here, the value $\hat{\theta}$ can be regarded as an observed value of the random variable

$$\hat{\Theta} = g(X_1, \cdots, X_n)$$

because x_j may be regarded as an observed value of X_j where $j = 1, \cdots, n$. In this test there are two (2) possibilities of making an error, as follows.

Type I Error The hypothesis is true but is rejected[28] because Θ assumes a value $\hat{\theta} > c$. Obviously, the probability of making such an error equals

[28] hence the alternative is accepted.

$$P\left(\hat{\Theta} > c\right)_{\theta} = \theta_0 = \alpha. \tag{2.20}$$

α is called the **significance level** of the test, as mentioned before.

Type II Error The hypothesis is false but is accepted because $\hat{\Theta}$ assumes a value $\hat{\theta} \equiv c$. The probability of making such an error is denoted by β; thus

$$P\left(\hat{\Theta} \leqq c\right)_{\theta=\theta_0} = \beta. \tag{2.21}$$

$\eta = 1 - \beta$ is called the **power** of the test. Obviously, the power η is the probability of avoiding a Type II error.

Formulas Eq. (2.20) and Eq. (2.21) show that both α and β depend on c, and we would like to choose c so that these probabilities of making errors are as small as possible. But the important **Fig.** 2.4 shows that these are conflicting requirements because to let α decrease we must shift c to the right, but then β increases. In practice we first choose α (5%, sometimes 1%), then determine c, and finally compute β. If β is large so that the power $\eta = 1 - \beta$ is small, we should repeat the test, choosing a larger sample, for reasons that will appear shortly. If the alternative is **NOT** a single

number but is of the form Eq. (2.17)-Eq. (2.19), then β becomes a function of θ. This function $\beta(\theta)$ is called the operating characteristic (OC) of the test and its curve the OC curve. Clearly, in this case $\eta = 1 - \beta$ also depends on θ. This function $\eta(\theta)$ is called the power function of the test.

> Of course, from a test that leads to the acceptance of a certain hypothesis θ_0, it does **NOT** follow that this is the only possible hypothesis or the best possible hypothesis. Hence the terms "not reject" or "fail to reject" are perhaps better than the term "accept".

The following example explains the three (3) kinds of hypotheses.

Exercise 2.9: Test for the Mean of the Normal Distribution with Known Variance

Let X be a normal random variable with variance $\sigma^2 = 9$. Using a sample of size $n = 10$ with mean $\overline{x}$, test the hypothesis $\mu = \mu_0 = 24$ against the three (3) kinds of alternatives, namely,

$$\text{(a) } \mu > \mu_0 \qquad \text{(b) } \mu < \mu_0 \qquad \text{(c) } \mu \neq \mu_0$$

Solution

We choose the significance level $\alpha = 0.05$. An estimate of the mean will be obtained from:

$$\overline{X} = \frac{1}{n}\left(X_1 + \cdots + X_n\right).$$

If the hypothesis is true, $\hat{X}$ is normal with mean $\mu = 24$ and variance $\sigma^2/n = 0.9$. Hence we may obtain the critical value c from Table 48 in App. 5.

a. Right-Sided Test We determine c from

$$P\left(\overline{X} > c\right)_{\mu=24} = \alpha = 0.05$$

that is,

$$P\left(\hat{X} \leq c\right)_{\mu=24} = \Phi\left(\frac{c-24}{\sqrt{0.9}}\right) = 1 - \alpha = 0.95.$$

Using Table A8 in App. 5 gives $(c-24)/\sqrt{0.9} = 1.645$, and $c = 25.56$, which is greater than μ_0. If $\overline{x} \leqq 25.56$, the hypothesis is accepted. If $\overline{x} > 25.56$, it is rejected.

The power function of the test is:

$$\eta(\mu) = P(\overline{X} > 25.56)_\mu = 1 - P(\overline{X} \leq 25.56)_\mu$$
$$= 1 - \Phi\left(\frac{25.56-\mu}{\sqrt{0.9}}\right) = 1 - \Phi(26.94 - 1.05\mu)$$

b. Left-Sided Test The critical value c is obtained from the equation

$$P\left(\overline{X} \leq c\right)_{\mu-24} = \Phi\left(\frac{c-24}{\sqrt{0.9}}\right) = \alpha = 0.05.$$

Table A8 in App.5 yields $c = 24 - 1.56 = 22.44$. If $\overline{x} \cong 22.44$, we accept the hypothesis. If $\overline{x} < 22.44$, we reject it. The power function of the test is

$$\eta(\mu) = P\left(\overline{x} \cong 22.44\right)_\mu = \Phi\left(\frac{22.44-\mu}{\sqrt{0.9}}\right) = \Phi(23.65 - 1.05\mu).$$

c. Two-Sided Test Since the normal distribution is symmetric, we choose c_1 and c_2 equidistant from $\mu = 24$, say, $c_1 = 24 - k$ and $c_2 = 24 + k$, and determine k from

$$P\left(24 - k \leqq \overline{X} \leqq 24 + k\right)_{\mu=24} = \Phi\left(\frac{k}{\sqrt{0.9}}\right) - \Phi\left(-\frac{k}{\sqrt{0.9}}\right) = 1 - \alpha = 0.95.$$

Table A8 in App. 5 gives $k/\sqrt{0.9} = 1.960$, hence $k = 1.86$. This gives the values $c_1 = 24 - 1.86 = 22.14$ and $c_2 = 24 + 1.86 = 25.86$. If $\overline{x}$ is not smaller than c_1 and not greater than c_2, we accept the hypothesis. Otherwise, we reject it. The power function of the test is (Fig. 535)

$$\eta(\mu) = P\left(\overline{x} < 22.14\right)_\mu + P\left(\overline{x} > 25.86\right)_\mu = P\left(\overline{x} < 22.14\right)_\mu + 1 - P\left(\overline{x} \leq 25.86\right)_\mu$$

$$= 1 + \Phi\left(\frac{22.14 - \mu}{\sqrt{0.5}}\right) - \Phi\left(\frac{25.86 - \mu}{\sqrt{0.5}}\right)$$

$$= 1 + \Phi(23.34 - 1.05\mu) - \Phi(27.26 - 1.05\mu).$$

Consequently, the operating characteristic $\beta(\mu) = 1 - \eta(\mu)$ (see before) is (Fig. 536).

Exercise 2.10: Comparison of the Means of Two Normal Distributions

Using a sample $x_1, \cdots, x_m$ from a normal distribution with unknown mean μ_x and a sample $y_1, \cdots, y_m$ from another normal distribution with unknown mean μ_y, we want to test the hypothesis that the means are equal, $\mu_x = \mu_y$, again an alternative, say, $\mu_x > \mu_y$. The variances need not be known but are assumed to be equal.

Solution

Two cases of comparing means are of practical importance:

Case A

The sample have the same size. Furthermore, each value of the first sample corresponds to precisely one value of the other, because corresponding values result from the same person or thing (**paired comparison**)

> for example, two measurements of the same thing by two different methods or two measurements from the two cycles of the same person.

More generally, they may result from a circular individual or things, for example, the lower reason is that the lower reason is that the lower value of the second sum of the differences of corresponding values used as the previous that the population corresponds to the differences has mean 0. using the method in Example 3. If we have a choice, this method is better than the following.

Case B

The two samples are independent and not necessarily of the same size. Then we may proceed as follows. Suppose that the alternative is $\frac{1}{2}\ u^n > u_{PV}$ we choose a significance level α. Then we compute the sample means $\overline{x}$ and $\overline{y}$ as well as $(n_1 - 1)u_{PV}^2$ and $(n_2 - 1)u_{PV}^2$ where $\overline{x}^2$ and $\overline{y}^2$ are the sample variances. Using Table 9 in App.5 with $n_1 + n_2 - 2$ degrees of freedom, we now determine c from

2.5 Goodness of Fit

To test for goodness of fit[29] means that we wish to test that a certain function $F(x)$ is the distribution function of a distribution from which we have a sample $x_1, \cdots, x_n$. Then we test whether the **sample distribution function** $\widetilde{F}(x)$ defined as:

$$\widetilde{F}(x) = \text{Sum of the relative frequencies of all sample values } x_j \text{ not exceeding } x, \quad (2.22)$$

fits $\widetilde{F}(x)$ sufficiently well. If this is so, we shall accept the hypothesis that $\widetilde{F}(x)$ is the distribution function of the population; else, we shall **reject the hypothesis**.

> This test is of considerable practical importance, and it differs in character from the tests for parameters (μ, σ^2, etc.) considered thus far.

To test in that fashion, we have to know how much $\widetilde{F}(x)$ can differ from $F(x)$ if the hypothesis is **true**. Hence we must first introduce a quantity which measures the deviation of $\widetilde{F}(x)$ from $F(x)$, and we must know the probability distribution of this quantity under the assumption that the hypothesis is true.

Then we proceed as follows.

We determine a number, lets use c, such that, if the hypothesis is **true**, a deviation greater than c has a small preassigned probability. If, nevertheless, a deviation greater than c occurs, we have reason to doubt that the hypothesis is true and we reject it. On the other hand, if the deviation does not exceed c, so that $\widetilde{F}(x)$ approximates $F(x)$ sufficiently well, we accept the hypothesis. Of course, if we accept the hypothesis, this means that we have insufficient evidence to reject it, and this does not exclude the possibility that there are other functions that would not be rejected in the test.

In this respect the situation is quite similar to hypothesis testing we talked previously.

The following text-block shows a test of that type, which was introduced by *R. A. Fisher*. This test is justified by the fact that if the hypothesis is true, then χ_0^2 is an observed value of a random variable whose distribution function approaches that of the chi-square distribution with $K-1$ degrees of freedom[30] as n approaches infinity. The requirement that at least five (5) sample values lie in each interval results from the fact that for finite n that random variable has only approximately a chi-square distribution.

> If the sample is so small that the requirement cannot be satisfied, one may continue with the test, but then use the result **with caution**.

[29] In literature, this method also means χ^2-Test.

Historical Anecdote

During the 19th century, statistical analytical methods were mainly applied to biological data and it was customary for researchers to assume observations followed a normal distribution, such as Sir George Airy and Mansfield Merriman, whose works were criticized by Karl Pearson in his 1900 paper.

At the end of the 19th century, Pearson noticed the existence of significant skewness within some biological observations. To model the observations regardless of being normal or skewed, Pearson, in a series of articles published from 1893 to 1916,[3][4][5][6] devised the Pearson distribution, a family of continuous probability distributions, which includes the normal distribution and many skewed distributions, and proposed a method of statistical analysis consisting of using the Pearson distribution to model the observation and performing a test of goodness of fit to determine how well the model really fits to the observations.

[30] or $K-r-1$ degrees of freedom if r parameters are estimated.

Chi-square Test for F(x) being the Distribution Function of a Population

1. Subdivide the x-axis into n intervals $I_1, \cdots, I_n$ such that each interval contains at least five (5) values of the given sample $x_1, \cdots, x_n$.

   Determine the number b_j of sample values in the interval I_j, where $j = 1, \cdots, K$. If a sample value lies at a common boundary point of two (2) intervals, add 0.5 to each of the two (2) corresponding b_j.

2. Using $F(x)$, calculate the probability p_j that the random variable X under consideration assumes any value in the interval I_j, where $j = 1, \cdots, K$. Then, calculate

   $$e_j = np_j.$$

   This is the number of sample values **theoretically expected** in I_j if the hypothesis is true.

3. Compute the deviation:

   $$\chi_0^2 = \sum_{j=1}^{K} \frac{(b_j - e_j)^2}{e_j}.$$

4. Choose a significance level such as 5%, 1%, or the like.

5. Determine the solution c of the equation

   $$P\left(\chi^2 \leqq c\right) = 1 - \alpha.$$

   from the table of the chi-square distribution with $K - 1$ degrees of freedom (Table A10 in App. 5).

If r parameters of $F(x)$ are unknown and their maximum likelihood estimates are used, then use $K - r - 1$ degrees of freedom, instead of $K - 1$.

If $\chi_0^2 \leqq c$, accept the hypothesis. If $\chi_0^2 > c$, reject the hypothesis.

Exercise 2.11: Test of Normality

Test whether the population from which the sample in table given below was taken is normal.

320	380	340	410	380	340	360	350	320	370
350	340	350	360	370	350	380	370	300	420
370	390	390	440	330	390	330	360	400	370
320	350	360	340	340	350	350	390	380	340
400	360	350	390	400	350	360	340	370	420
420	400	350	370	330	320	390	380	400	370
390	330	360	380	350	330	360	300	360	360
360	390	350	370	370	350	390	370	370	340
370	400	360	350	380	380	360	340	330	370
340	360	390	400	370	410	360	400	340	360

Solution

The table given in the question shows the values, column by column, in the order obtained in the experiment. The next table gives the frequency distribution and Fig. 542 the histogram.

The maximum likelihood estimates for μ and σ^2 are $\hat{\mu} = \overline{x} = 364.7$ and $\overline{\sigma}^2 = 712.9$. The computation in Table 25.10 yields $\overline{x}_0^2 = 2.688$. It is very interesting that the interval $375 \cdots 385$ contributes over 50% of $\overline{x}_0^2$. From the histogram we see that the corresponding frequency looks much too small. The second largest contribution comes from $395 \cdots 405$, and the histogram shows that the frequency seems somewhat too large, which is perhaps not obvious from inspection.

Tensile Strength	Absolute Freq.	Relative Freq.	Cumulative Absolute Freq.	Cumulative Relative Freq.
300	2	0.02	2	0.02
310	0	0.00	2	0.02
320	4	0.04	6	0.06
330	6	0.06	12	0.12
340	11	0.11	23	0.23
350	14	0.14	37	0.37
360	16	0.16	53	0.53
370	15	0.15	68	0.68
380	8	0.08	76	0.76
390	10	0.10	86	0.86
400	8	0.08	94	0.94
410	2	0.02	96	0.96
420	3	0.03	99	0.99
430	0	0.00	99	0.99
440	1	0.01	100	1.00

Table 2.1: Frequency table of the sample given in the question.

We choose $\alpha = 5\%$. Since $K = 10$ and we estimated $r = 2$ parameters we have to use Table A10 in App. 5 with $K - r - 1 = 7$ degrees of freedom. We find $c = 14.07$ as the solution of $P(\chi^2 \cong c) = 95\%$. Since $\chi_0^2 < c$, we accept the hypothesis that the population is normal.

2.6 Regression and Correlation

Up to this points, we were only concerned with random experiments in which we observed a single quantity[31] and got samples whose values were single numbers. In this section we discuss experiments in which we observe or measure two (2) quantities simultaneously, so we get samples of pairs of values:

[31] In this case it is a random variable.

$$(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$$

Most applications involve one of two kinds of experiments, which are as follows:

Regression one (1) of the two (2) variables, call it x, can be regarded as an ordinary variable because we can measure it without substantial error or we can even give it values we want. x is called the **independent variable**, or sometimes the **controlled variable** as we can control it.[32] The other variable, Y, is a random variable, and we are interested in the dependence of Y on x.

[32] set it at values we choose

Examples include the dependence of the blood pressure Y on the age x of a person or, as we shall now say, the regression of Y on x. The regression of the gain of weight Y of certain animals on the daily ratio of food x, the regression of the heat conductivity Y of work on the specific weight x of the rock, etc.

Correlation both quantities are random variables and we are interested in relations between them.

Examples are the relation[33] between user X and year Y of the front tires of cars, between grades X and Y of students in mathematics and in physics, respectively, between the hardness X of steel plates in the centre and the hardness Y near the edges of the plates, etc.

[33] we say correlation

2.6.1 Regression Analysis

In regression analysis the dependence of Y on x is a dependence of the mean μ of Y on x, so that $\mu = \mu(x)$ is a function in the ordinary sense. The curve of $\mu(x)$ is called the **regression curve** of Y on x.

Let's look into the simplest case, namely, that of a straight regression line:

$$\mu(x) = \kappa_0 + \kappa_1 x. \tag{2.23}$$

Then we may want to graph the sample values as n points in the xY-plane, fit a straight line through them, and use it for estimating $\mu(x)$ at values of x that interest us, so we know what values of Y we can expect for those x.

Fitting line by eye would not be good because it would be subjective as people would come up with different estimations, particularly if the points are scattered. So we need a mathematical method

which gives a unique result depending **only** on the n points. A widely used procedure is the method of least squares by Gauss and Legendre.

For our task we may formulate it as follows.

Theory 2.23: Least Square Principle

The straight line should be fitted through the given points so that the sum of the squares of the distances of those points from the straight line is **minimum**, where the distance is measured in the vertical direction, which is the y-direction.

To get uniqueness of the straight line, we need some extra condition.

To see this, lets look at a small example and take the sample $(0,\,1)\,,\,(0,-1)$. Then all the lines $y = k_1 x$ with any k_1 satisfy the principle.

The following assumption will imply uniqueness, as we shall find out.

Theory 2.24: Assumption A1

The x-values $x_1, \cdots, x_n$ in our sample $(x_1, y_1), \cdots, (x_n, y_n)$ are not all equal.

From a given sample $(x_1,\,y_1)\,,\,(x_2,\,y_2)\,,\,\cdots,\,(x_n,\,y_n)$ we shall now determine a straight line by least squares. We write the line as:

$$y = k_0 + k_1 x, \tag{2.24}$$

and call it the sample regression line as it will be the counterpart of the population regression line given in Eq. (2.23).

[34] distance measured in the y-direction

Now a sample point $\left(x_j,\,y_j\right)$ has the vertical distance[34] from Eq. (2.24) given by:

$$\left|y_j - \left(k_0 + k_1 x_j\right)\right| \tag{2.25}$$

This gives the sum of the squares of these distances as:

$$q = \sum_{j=1}^{n} \left(y_j - k_0 - k_1 x_j\right)^2 \tag{2.26}$$

In the method of least squares we now have to determine k_0 and k_1 such that q is minimum. From calculus we know that a necessary condition for this is:

$$\frac{\partial q}{\partial k_0} = 0 \qquad \text{and} \qquad \frac{\partial q}{\partial k_1} = 0. \tag{2.27}$$

We shall see that from this condition we obtain for the sample regression line the formula

$$y - y \;= k_1\left(x - \overline{x}\right). \tag{2.28}$$

Here $\overline{x}$ and $\overline{y}$ are the means of the x- and the y-values in our sample, that is,

$$\overline{x} = \frac{1}{n}\left(x_1 + \cdots + x_n\right) \qquad \text{and} \qquad \overline{y} = \frac{1}{n}\left(y_1 + \cdots + y_n\right). \tag{2.29}$$

The slope k_1 in Eq. (2.28) is called the **regression coefficient** of the sample and is given by:

$$k_1 = \frac{s_{xy}}{s_x^2}. \tag{2.30}$$

Here the **sample covariance** s_{xy} is:

$$s_{xy} = \frac{1}{n-1}\sum_{j=1}^{n}\left(x_j - \overline{x}\right)\left(y_j - \overline{y}\right) = i\frac{1}{n-1}\left[\sum_{j=1}^{n} x_j y_j - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)\left(\sum_{j=1}^{n} y_j\right)\right] \tag{2.31}$$

and s_x^2 is given by

$$s_x^2 = \frac{1}{n-1}\sum_{j=1}^{n}\left(x_j - \overline{x}\right)^2 = \frac{1}{n-1}\left[\sum_{j=1}^{n} x_j^2 - \frac{1}{n}\left(\sum_{j=1}^{n} x_j\right)^2\right]. \tag{2.32}$$

From Eq. (2.28) we see that the sample regression line passes through the point $(\overline{x}, \overline{y})$, by which it is determined, together with the regression coefficient Eq. (2.30). We may call s_x^2 the *variance* of the x-values, but keep in mind that x is an ordinary variable, and **NOT** a random variable.

We shall soon also need:

$$s_y^2 = \frac{1}{n-1}\sum_{j=1}^{n}\left(y_j - \overline{y}\right)^2 = \frac{1}{n-1}\left[\sum_{j=1}^{n} y_j^2 - \frac{1}{n}\left(\sum_{j=1}^{n} y_j\right)^2\right]. \tag{2.33}$$

Now, let's try to derive Eq. (2.28) and Eq. (2.30). Differentiating Eq. (2.26) and using Eq. (2.27), we first obtain:

$$\frac{\partial q}{\partial k_0} = -2\sum\left(y_j - k_0 - k_1 x_j\right) = 0,$$
$$\frac{\partial q}{\partial k_1} = -2\sum x_j\left(y_j - k_0 - k_1 x_j\right) = 0.$$

where we sum over j from 1 to n. We now divide by 2, write each of the two sums as three sums, and take the sums containing y_j and $x_j y_j$ over to the right.

Then we get the normal equations:

$$\begin{aligned} k_0 n + k_1 \sum x_j &= \sum y_j \\ k_0 \sum x_j + k_1 \sum x_j^2 &= \sum x_j y_j. \end{aligned} \tag{2.34}$$

This is a linear system of two (2) equations with two unknowns k_0, k_1, with coefficient determinant being:

$$\begin{vmatrix} n & \sum x_j \\ \sum x_j & \sum x_j^2 \end{vmatrix} = n\sum x_j^2 - \left(\sum x_j\right)^2 = n(n-1)\,s_x^2 = n\sum\left(x_j - \overline{x}\right)^2,$$

and is **NOT** zero due to the first assumption (A1) we made prior. Hence the system has a unique solution. Dividing the first equation of Eq. (2.34) by n and using Eq. (2.29), we get $k_0 = \overline{y} - k_1\overline{x}$.

Together with $y = k_0 + k_1 x$ in Eq. (2.24) this gives Eq. (2.28). To get Eq. (2.30), we solve the system Eq. (2.34) by Cramer's rule or elimination, finding

$$k_1 = \frac{n\sum x_j y_j - \sum x_i \sum y_j}{n\,(n-1)\,s_x^2}. \tag{2.35}$$

Which completes the derivation.

Exercise 2.12: Regression Line

The decrease of volume y (in %) of leather for certain fixed values of high pressure x (atmosphere) was measured. The results are shown in the first two columns of the table below.

Given Values		Auxiliary Values	
x_j	y_j	x_j^2	$x_j y_j$
4000	2.3	16,000,000	9200
6000	4.1	36,000,000	24,600
8000	5.7	64,000,000	45,600
10,000	6.9	100,000,000	69,000
28,000	19.0	216,000,000	148,400

Table 2.2: Dataset

Find the regression line of y on x.

Solution

We see that the sample count is $n = 4$ and obtain the values

$$\overline{x} = \frac{28000}{4} = 7000 \quad \text{and} \quad \overline{y} = \frac{19.0}{4} = 4.75,$$

and from Eq. (2.32), Eq. (2.33) and Eq. (2.31)

$$s_x^2 = \frac{1}{3}\left(216,000,000 - \frac{28,000^2}{4}\right) = \frac{20,000,000}{3}$$
$$s_{xy} = \frac{1}{3}\left(148,400 - \frac{28,000 \cdot 19}{4}\right) = \frac{15,400}{3}.$$

Hence $k_1 = 15,400/20,000,000 = 0.00077$ from Eq. (2.30), and the regression line is

$$y - 4.75 = 0.00077\,(x - 7000)$$
$$y = 0.00077x - 0.64.$$

With both options being valid. Note $y\,(0) = -10.65$, which is physically meaningless, but typically indicates that a linear relation is merely an approximation valid on some restricted interval ■

2.6.2 Confidence Intervals

[35] which we have not made so far; least squares is a *geometric principle*, not involving probabilities.

If we want to get confidence intervals, we have to make assumptions about the distribution of Y.[35] We assume normality and independence in sampling:

Theory 2.25: Assumption A2

For each fixed x the random variable Y is normal with mean Eq. (2.23), that is,

$$\mu(x) = \kappa_0 + \kappa_1 x \tag{2.36}$$

and variance σ^2 **independent** of x.

Theory 2.26: Assumption A3

The n performances of the experiment by which we obtain a sample

$$(x_1, y_1), \quad (x_2, y_2), \quad \cdots, \quad (x_n, y_n) \tag{2.37}$$

are independent

κ_1 given in Eq. (2.36) is called the **regression coefficient** of the population because it can be shown that, under the assumptions given in A1 to A3, the maximum likelihood estimate of κ_1 is the sample regression coefficient k_1 given by Eq. (2.35).

Following with the assumptions from A1 to A3, we may now obtain a confidence interval for κ_1, as shown below.

Determination of Regression Coefficient under Assumptions A1 to A3

1. Choose a confidence level γ which can take values of 95%, 99%, or the like.
2. Determine the solution c of the equation,

$$F(c) = \frac{1}{2}(1+\gamma) \tag{2.38}$$

   from the table of the t-distribution with $n-2$ degrees of freedom (Table A9 in App. 5; n = sample size)

3. Using a sample $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$, calculate $(n-1)\,s_x^2$ from Eq. (2.32), $(n-1)s_{xy}$ from Eq. (2.31), k_1 from Eq. (2.30),

$$(n-1)\,s_y^2 = \sum_{j=1}^{n} y_j^2 - \frac{1}{n}\left(\sum_{j=1}^{n} y_j\right)^2$$

   which was described in Eq. (2.33), and

$$q_0 = (n-1)\left(s_y^2 - k_1^2 s_x^2\right)$$

4. Calculate:

$$K = c\sqrt{\frac{q_0}{(n-2)\,(n-1)\,s_x^2}}.$$

5. The confidence interval is then defined to be:

$$\mathrm{CONF}_\gamma\left\{k_1 - K \leqq \kappa_1 \leqq k_1 + K\right\}. \tag{2.39}$$

Exercise 2.13: Confidence Interval for the Regression Coefficient

Using the sample in **Table** 2.2, determine a confidence interval for κ_1 by the method described just previously.

Solution

1. We choose $\gamma = 0.95$.
2. Eq. (2.38) takes the form $F(c) = 0.975$, and Table A9 in App. 5 with $n - 2 = 2$ degrees of freedom gives $c = 4.30$.
3. From Example 1 we have $3s_x^2 = 20{,}000{,}000$ and $k_1 = 0.00077$. From **Table** 2.2 we compute:

$$3s_y^2 = 102.0 - \frac{19^2}{4} = 11.95.$$
$$q_0 = 11.95 - 20,000,000 \cdot 0.00077^2 = 0.092.$$

4. We therefore obtain:

$$K = 4.30\sqrt{\frac{0.092}{(2 \cdot 20,000,000}}$$
$$= 0.000206$$
$$\text{CONF}_{0.95}\left\{0.00056 \leqq \kappa_1 \leqq 0.00098\right\} \quad \blacksquare$$

2.6.3 Correlation Analysis

Time to give an introduction to the basic facts in correlation analysis.

The topic of **correlation analysis** is concerned with the relation between X and Y in a two-dimensional random variable (X, Y). A sample consists of n ordered pairs of values:

$$(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n),$$

as before. The interrelation between the x and y values in the sample is measured by the sample covariance s_{xy} in Eq. (2.31) or by the sample correlation coefficient:

$$r = \frac{s_{xy}}{s_x s_y} \tag{2.40}$$

with s_x and s_y given in Eq. (2.32) and Eq. (2.33). Here r has the advantage that it does not change under a multiplication of the x and y values by a factor.[36]

[36] Such as the unit changing from g to kg.

Theory 2.27: Sample Correlation Coefficient

The sample correlation coefficient r satisfies $-1 \leqq r \leqq 1$.

In particular, $r = \pm 1$ if and only if the sample values lie on a straight line which can be seen in **Fig.** 2.5.

The theoretical counterpart of r is the correlation coefficient ρ of X and Y,

$$\rho = \frac{\sigma_{XY}}{\sigma_X \sigma_Y} \tag{2.41}$$

where:

$$\mu_X = E(X) \qquad \mu_Y = E(Y) \qquad \sigma_X^2 = E\left(\left[X - \mu_X\right]^2\right) \qquad \sigma_Y^2 = E\left(\left[Y - \mu_Y\right]^2\right)$$

which are the means and variances of the marginal distributions of X and Y. The covariance (σ_{XY}), on the other hand, is defined as:

$$\sigma_{XY} = E\left(\left[X - \mu_X\right]\left[Y - \mu_Y\right]\right) = E(XY) - E(X)\,E(Y) \tag{2.42}$$

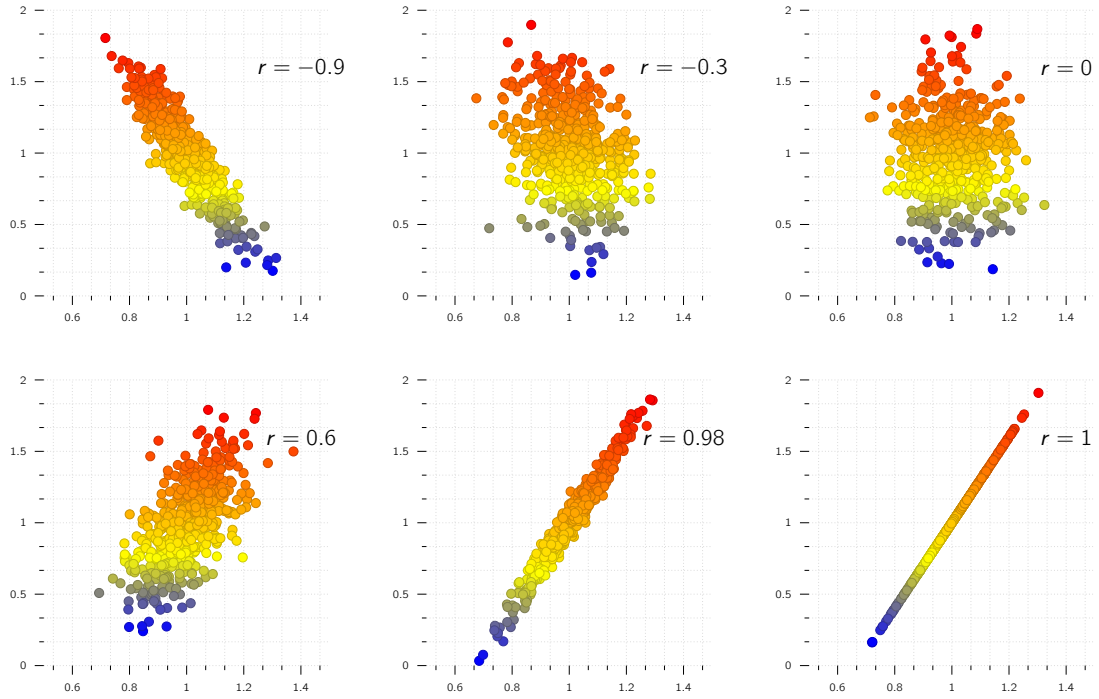


Figure 2.5: Samples with various values of the correlation coefficient r.

Theory 2.28: Correlation Coefficient

The correlation coefficient ρ satisfies $-1 \leqq \rho \leqq 1$.

In particular, $\rho = \pm 1$ if and only if X and Y are **linearly related**, that is,

$$Y = \gamma X + \delta, X = \gamma_* Y + \delta_*.$$

X and Y are **uncorrelated** if $\rho = 0$.

Theory 2.29: Independence and Relation to Normal Distribution

a. If X and Y are independent, they are uncorrelated.

b. If (X, Y) is normal, then uncorrelated X and Y are **independent**.

Here the two-dimensional normal distribution can be introduced by taking two (2) independent standardised normal random variables X^*, Y^*, whose joint distribution thus has the density:

$$f^*\left(x^*, y^*\right) = \frac{1}{2\pi} e^{-\left(x^{*2} + y^{*2}\right)/2}$$

and setting

$$X = \mu_X + \sigma_X X^*$$
$$Y = \mu_Y + \rho\sigma_Y X^* + \sqrt{1-\rho^2}\sigma_Y Y^*$$

This gives the general two-dimensional normal distribution with the density:

$$f(x, y) = \frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1-\rho^2}} e^{-h(x, y)/2} \tag{2.43}$$

where

$$h(x, y) = \frac{1}{1-\rho^2}\left[\left(\frac{x-\mu_X}{\sigma_X}\right)^2 - 2\rho\left(\frac{x-\mu_X}{\sigma_X}\right)\left(\frac{y-\mu_Y}{\sigma_Y}\right) + \left(\frac{y-\mu_Y}{\sigma_Y}\right)^2 x\right] \tag{2.44}$$

In Theorem 3(b), normality is important, as we can see from the following example.

Exercise 2.14: Uncorrelated but Dependent Random Variables

If X assumes the value of -1, 0, 1 with probability $\frac{1}{3}$ and $Y = X^2$, then $EX = 0$ and in Eq. (2.26)

$$\sigma_{XY} = E(XY) = E\left(X^3\right) = (-1)^3 \cdot \frac{1}{3} + 0^3 \cdot \frac{1}{3} + 1^2 \cdot \frac{1}{3} = 0,$$

so that $\rho = 0$ and X and Y are uncorrelated. But they are certainly **NOT** independent since they are even functionally related.

2.6.4 Test for the Correlation Coefficient

The following text-block shows a test for ρ in the case of the two-dimensional normal distribution. t is an observed value of a random variable that has a t-distribution with $n-2$ degrees of freedom.[37]

[37] This was shown by R. A. Fisher.

Testing the Hypothesis against the Alternative in case of Two-Dimensional Normal Distribution

1. Choose a significance level α (5%, 1%, or the like).
2. Determine the solution c of the equation:

$$P\left(T \leqq c\right) = 1 - \alpha,$$

   from the t-distribution (Table A9 in App. 5) with $n-2$ degrees of freedom.
3. Calculate r from Eq. (2.40), using a sample $(x_1, y_1), \cdots, (x_n, y_n)$.
4. Calculate

$$t = r\left(\sqrt{\frac{n-2}{1-r^2}}\right).$$

5. If $t \leqq c$, accept the hypothesis. If $t > c$, reject the hypothesis.

Exercise 2.15: Test for the Correlation Coefficient

Test the hypothesis $\rho = 0$ (independence of X and Y, because of Theorem 3) against the alternative $\rho > 0$, using the data when $r = 0.6$ (normal soliding errors on 10 two-sided circuit boards done by 10 workers $x =$ front, $y =$ back of the bonds).

Solution

We choose $\alpha = 5\%$; which makes $1 - \alpha = 95\%$. Since $n = 10$, and $n - 2 = 8$, the table gives $c = 1.86$. Also,

$$t = 0.6\sqrt{\frac{8}{0.64}} = 2.12 > c.$$

We reject the hypothesis and search that there is a positive correlation. A worker making few errors on the front side also tends to make few errors on the reverse side of the bond ■

Part II

Scientific Python Essentials

Chapter 3

Numpy

Table of Contents

3.1 Introduction 75
3.2 Why use Numpy 77
3.3 An Introductionary Example - Calculating Grades 78
3.4 Getting Into Shape: Array Shapes and Axes 80
3.5 Data Science Operations: Filter, Order, Aggregate 85
3.6 Masking and Filtering 87
3.7 Transposing, Sorting, and Concatenating 93
3.8 Aggregating 96
3.9 Practical Exercise Implementing Maclauring Series 97
3.10 Optimizing Storage: Data Types 99
3.11 Numerical Types 100

Figure 3.1: The logo of numpy.

3.1 Introduction

NumPy is a Python library that provides a simple yet powerful data structure:

the n-dimensional array.

This is the foundation on which almost all the power of Python's data science toolkit is built, and learning NumPy is the first step on any Python data scientist's journey. This tutorial will provide you with the knowledge you need to use NumPy and the higher-level libraries that rely on it.

3.2 Why use Numpy

Since you already know Python, you may be asking yourself if you really have to learn a whole new paradigm to do data science.

Here are the top four benefits that NumPy can bring to your code:

- **More speed:** Algorithms written in C that complete in nanoseconds rather than seconds.
- **Fewer loops:** Reduces loops and keep from getting tangled up in iteration indices.
- **Clearer code:** Without loops, your code will look more like the equations you're trying to calculate.
- **Better quality:** Thousands of contributors working to keep NumPy fast, friendly, and bug free.

Because of these benefits, NumPy is the de facto[1] standard for multidimensional arrays in Python data science, and many of the most popular libraries are built on top of it.

[1] De facto means it is an unofficial standard, while *de jure* means it is a standard with a legal backing

Learning NumPy is a great way to set down a solid foundation as we expand our knowledge into more specific areas of data science.

3.3 An Introductionary Example - Calculating Grades

This first example introduces a few core concepts in NumPy that we'll use throughout the rest of the tutorial:

- Creating arrays using `numpy.array()`
- Treating complete arrays like individual values to make vectorised calculations more readable
- Using built-in NumPy functions to modify and aggregate the data

These concepts are the core of using NumPy effectively.

Imagine a scenario: You're a teacher who has just graded your students on a recent test. Unfortunately, you may have made the test too challenging, and most of the students did worse than expected. To help everybody out, you're going to curve everyone's grades[2].

[2] Grading on a curve is a term that describes a variety of different methods that a teacher uses to adjust the scores their students received on a test in some way. Most of the time, grading on a curve boosts the students' grades by moving their actual scores up a few notches, perhaps increasing the letter grade

It'll be a relatively rudimentary curve, though. You'll take whatever the average score is and declare that a **C**. Additionally, you'll make sure that the curve doesn't accidentally hurt your students' grades or help so much that the student does better than 100%.

C.R. 1

```python
import numpy as np
CURVE_CENTER = 80
grades = np.array([72, 35, 64, 88, 51, 90, 74, 12])
def curve(grades):
    average = grades.mean()
    change = CURVE_CENTER - average
    new_grades = grades + change
    return np.clip(new_grades, grades, 100)

curve(grades)
```

> The original scores have been increased based on where they were in the pack, but none of them were pushed over 100%.

Here are the important highlights:

- Line 1 imports NumPy using the `np` alias, which is a common convention[3] that saves you a few keystrokes.
- Line 3 creates your first NumPy array, which is one-dimensional and has a shape of (8,) and a data type of `int64`. Don't worry too much about these details yet. You'll explore them in more detail later in the tutorial.
- Line 5 takes the average of all the scores using `.mean()`. Arrays have a lot of methods.

[3] It is a convention and not a rule, but all documentations use np so you might as well

On line 7, you take advantage of two (2) important concepts at once:

1. **Vectorization:** process of performing the same operation in the same way for each element in an array. This removes for loops from your code but achieves the same result.

2. **Broadcasting:** process of extending two arrays of different shapes and figuring out how to perform a vectorized calculation between them

Remember, grades is an array of numbers of shape (8,) and change is a scalar, or single number, essentially with shape (1,). In this case, NumPy adds the scalar to each item in the array and returns a new array with the results.

Finally, on line 8, you limit, or clip, the values to a set of minimums and maximums. In addition to array methods, NumPy also has a large number of built-in functions. You don't need to memorize them all-that's what documentation is for. Anytime you get stuck or feel like there should be an easier way to do something, take a peek at the documentation and see if there isn't already a routine that does exactly what you need.

In this case, you need a function that takes an array and makes sure the values don't exceed a given minimum or maximum. `clip()` does exactly that.

Line 8 also provides another example of broadcasting. For the second argument to `clip()`, you pass grades, ensuring that each newly curved grade doesn't go lower than the original grade. But for the third argument, you pass a single value: 100.

NumPy takes that value and broadcasts it against every element in `new_grades`, ensuring that none of the newly curved grades exceeds a perfect score.

3.4 Getting Into Shape: Array Shapes and Axes

Now that you've seen some of what NumPy can do, it's time to firm up that foundation with some important theory. There are a few concepts that are important to keep in mind, especially as you work with arrays in higher dimensions.

Vectors, which are one-dimensional arrays of numbers, are the least complicated to keep track of. Two dimensions aren't too bad, either, because they're similar to spreadsheets.

But things start to get tricky at three dimensions, and visualizing four? At this point geometrical thinking of arrays becomes unfeasible and its better to think them **ONLY** as data points.

3.4.1 Understanding Shapes

Shape is an important concept when you're using multidimensional arrays. At a certain point, it's easier to forget about visualizing the shape of your data and to instead follow some mental rules and trust NumPy to tell you the correct shape.

All arrays have a property called `.shape` which returns a tuple of the size in each dimension. It's less important which dimension is which, but it's critical that the arrays you pass to functions are in the shape that the functions expect. A common way to confirm that your data has the proper shape is to print the data and its shape until you're sure everything is working like you expect.

This next example will show this process. You'll create an array with a complex shape, check it, and reorder it to look like it's supposed to:

C.R. 2

python
```python
import numpy as np

temperatures = np.array([
    29.3, 42.1, 18.8, 16.1, 38.0, 12.5,
    12.6, 49.9, 38.6, 31.3, 9.2, 22.2
]).reshape(2, 2, 3)

print(temperatures.shape)
print(temperatures)
print(np.swapaxes(temperatures, 1, 2))
```

text
```
(2, 2, 3)

[[[29.3 42.1 18.8]
  [16.1 38.  12.5]]
 [[12.6 49.9 38.6]
  [31.3  9.2 22.2]]]

[[[29.3 16.1]
```

```
    [42.1 38. ]
    [18.8 12.5]]
   [[12.6 31.3]
    [49.9  9.2]
    [38.6 22.2]]]
```

Here, you use a `numpy.ndarray` method called `.reshape()` to form a 2-by-2-by-3 block of data. When you check the shape of your array in input 3, it's exactly what you told it to be. However, you can see how printed arrays quickly become hard to visualize in three or more dimensions. After you swap axes with `.swapaxes()`, it becomes little clearer which dimension is which. You'll see more about axes in the next section.

Shape will come up again in the section on broadcasting. For now, just keep in mind that these little checks don't cost anything. You can always delete the cells or get rid of the code once things are running smoothly.

3.4.2 Understanding Axes

The example above shows how important it is to know not only what shape your data is in but also which data is in which axis. In NumPy arrays, axes are zero-indexed and identify which dimension is which. For example, a two-dimensional array has a vertical axis (axis 0) and a horizontal axis (axis 1).

> Lots of functions and commands in NumPy change their behavior based on which axis you tell them to process.x

This example will show how .max() behaves by default, with no axis argument, and how it changes functionality depending on which axis you specify when you do supply an argument:

C.R. 3

python

```
import numpy as np

table = np.array([
    [5, 3, 7, 1],
    [2, 6, 7 ,9],
    [1, 1, 1, 1],
    [4, 3, 2, 0],
])

print(table.max())

print(table.max(axis=0))

print(table.max(axis=1))
```

```
9                                                                    text
[5 6 7 9]
[7 9 1 4]
```

By default, `.max()` returns the largest value in the entire array, no matter how many dimensions there are. However, once you specify an axis, it performs that calculation for each set of values along that **particular axis**.

For example, with an argument of `axis=0`, `.max()` selects the maximum value in each of the four vertical sets of values in table and returns an array that has been flattened, or aggregated into a one-dimensional array.

In fact, many of NumPy's functions behave this way:

> If no axis is specified, then they perform an operation on the entire dataset. Otherwise, they perform the operation in an axis-wise fashion.

3.4.3 Broadcasting

So far, you've seen a couple of smaller examples of broadcasting, but the topic will start to make more sense the more examples you see. Fundamentally, it functions around one (1) rule:

> arrays can be broadcast against each other if their dimensions match or if one of the arrays has a size of 1.

If the arrays match in size along an axis, then elements will be operated on element-by-element, similar to how the built-in Python function `zip()`[4] works.

[4] an iterator that will aggregate elements from two or more iterables. You can use the resulting iterator to quickly and consistently solve common programming problems, like creating dictionaries.

If one of the arrays has a size of 1 in an axis, then that value will be broadcast along that axis, or duplicated as many times as necessary to match the number of elements along that axis in the other array.

Here's a quick example. Array `A` has the shape `(4, 1, 8)`, and array `B` has the shape `(1, 6, 8)`. Based on the rules above, you can operate on these arrays together:

- In axis 0, A has a 4 and B has a 1, so B can be broadcast along that axis.
- In axis 1, A has a 1 and B has a 6, so A can be broadcast along that axis.
- In axis 2, the two arrays have matching sizes, so they can operate successfully.

All three (3) axes successfully follow the rule. You can set up the arrays like this:

C.R. 4
```python
import numpy as np

A = np.arange(32).reshape(4, 1, 8)

print(A)

B = np.arange(48).reshape(1, 6, 8)

print(B)
```

```text
[[[ 0  1  2  3  4  5  6  7]]
 [[ 8  9 10 11 12 13 14 15]]
 [[16 17 18 19 20 21 22 23]]
 [[24 25 26 27 28 29 30 31]]]
[[[ 0  1  2  3  4  5  6  7]
  [ 8  9 10 11 12 13 14 15]
  [16 17 18 19 20 21 22 23]
  [24 25 26 27 28 29 30 31]
  [32 33 34 35 36 37 38 39]
  [40 41 42 43 44 45 46 47]]]
```

A has 4 planes, each with 1 row and 8 columns. B has only 1 plane with 6 rows and 8 columns. Watch what NumPy does for you when you try to do a calculation between them.

Add the two (2) arrays together:

C.R. 5
```python
print(A + B)
```

```text
[[[ 0  2  4  6  8 10 12 14]
  [ 8 10 12 14 16 18 20 22]
  [16 18 20 22 24 26 28 30]
  [24 26 28 30 32 34 36 38]
  [32 34 36 38 40 42 44 46]
  [40 42 44 46 48 50 52 54]]
 [[ 8 10 12 14 16 18 20 22]
  [16 18 20 22 24 26 28 30]
  [24 26 28 30 32 34 36 38]
  [32 34 36 38 40 42 44 46]
  [40 42 44 46 48 50 52 54]
  [48 50 52 54 56 58 60 62]]
 [[16 18 20 22 24 26 28 30]
  [24 26 28 30 32 34 36 38]
  [32 34 36 38 40 42 44 46]
  [40 42 44 46 48 50 52 54]
  [48 50 52 54 56 58 60 62]
  [56 58 60 62 64 66 68 70]]
 [[24 26 28 30 32 34 36 38]
  [32 34 36 38 40 42 44 46]
  [40 42 44 46 48 50 52 54]
```

```
22    [48 50 52 54 56 58 60 62]
23    [56 58 60 62 64 66 68 70]
24    [64 66 68 70 72 74 76 78]]]
```

The way broadcasting works is that NumPy duplicates the plane in B three times so that you have a total of four, matching the number of planes in A. It also duplicates the single row in A five times for a total of six, matching the number of rows in B. Then it adds each element in the newly expanded A array to its counterpart in the same location in B. The result of each calculation shows up in the corresponding location of the output[5].

[5] It also provides a means of vectorizing array operations so that looping occurs in C instead of Python

> This is a good way to create an array from a range using arange()!

Once again, even though you can use words like "plane," "row," and "column" to describe how the shapes in this example are broadcast to create matching three-dimensional shapes, things get more complicated at higher dimensions. A lot of times, you'll have to simply follow the broadcasting rules and do lots of print-outs to make sure things are working as planned.

Understanding broadcasting is an important part of mastering vectorized calculations, and vectorized calculations are the way to write clean, idiomatic NumPy code.

3.5 Data Science Operations: Filter, Order, Aggregate

That wraps up a section that was heavy in theory but a little light on practical, real-world examples. In this section, you'll work through some examples of real, useful data science operations:

filtering, sorting, and aggregating data.

3.5.1 Indexing

Indexing uses many of the same idioms that normal Python code uses. You can use positive or negative indices to index from the front or back of the array. You can use a colon (`:`) to specify "the rest" or "all," and you can even use two colons to skip elements as with regular Python lists.

Here's the difference: NumPy arrays use commas between axes, so you can index multiple axes in one set of square brackets. An example is the easiest way to show this off. It's time to confirm Dürer's magic square!

> **Dürer's magic square** - a magic square with magic constant 34 used in an engraving entitled *Melencolia I* by *Albrecht Dürer*. The engraving shows a disorganized jumble of scientific equipment lying unused while an intellectual sits absorbed in thought.
> Dürer's magic square is located in the upper right-hand corner of the engraving. The numbers 15 and 14 appear in the middle of the bottom row, indicating the date of the engraving, 1514.
>
> | | | | |
> |---|---|---|---|
> | 16 | 3 | 2 | 13 |
> | 5 | 10 | 11 | 8 |
> | 9 | 6 | 7 | 12 |
> | 4 | 15 | 14 | 1 |

If you add up any of the rows, columns, or diagonals, then you'll get the same number, 34. That's also what you'll get if you add up each of the four quadrants, the center four squares, the four corner squares, or the four corner squares of any of the contained 3-by-3 grids.

Let's prove this statement:

C.R. 6

python

```python
import numpy as np

square = np.array([
    [16, 3, 2, 13],
    [5, 10, 11, 8],
    [9, 6, 7, 12],
    [4, 15, 14, 1]
])

for i in range(4):
```

C.R. 7 python

```
    assert square[:, i].sum() == 34
    assert square[i, :].sum() == 34


assert square[:2, :2].sum() == 34
assert square[2:, :2].sum() == 34
assert square[:2, 2:].sum() == 34
assert square[2:, 2:].sum() == 34
```

Inside the `for` loop, you verify that all the rows and all the columns add up to 34. After that, using selective indexing, you verify that each of the quadrants also adds up to 34.

The keyword assert - exists in almost every programming language. It has two (2) main uses:

- It helps detect problems early in your program, where the cause is clear, rather than later when some other operation fails. A type error in Python, for example, can go through several layers of code before actually raising an Exception if not caught early on.
- It works as documentation for other developers reading the code, who see the assert and can confidently say that its condition holds from now on.

One last thing to note is that you're able to take the sum of any array to add up all of its elements globally with `square.sum()`. This method can also take an axis argument to do an axis-wise summing instead.

3.6 Masking and Filtering

Index-based selection is great, but what if you want to filter your data based on more complicated non-uniform or non-sequential criteria? This is where the concept of a mask comes into play.

A mask is an array that has the exact same shape as your data, but instead of your values, it holds Boolean values: either `True` or `False`. You can use this mask array to index into your data array in nonlinear and complex ways. It will return all of the elements where the Boolean array has a `True` value.

Here's an example showing the process, first in slow motion and then how it's typically done, all in one line:

C.R. 8

python
```python
import numpy as np

numbers = np.linspace(5, 50, 24, dtype=int).reshape(4, -1)

print(numbers)

mask = numbers % 4 == 0

print(mask)

numbers[mask]

by_four = numbers[numbers % 4 == 0]

print(by_four)
```

text
```
[[ 5  6  8 10 12 14]
 [16 18 20 22 24 26]
 [28 30 32 34 36 38]
 [40 42 44 46 48 50]]

[[False False  True False  True False]
 [ True False  True False  True False]
 [ True False  True False  True False]
 [ True False  True False  True False]]

 [ 8 12 16 20 24 28 32 36 40 44 48]
```

You'll see an explanation of the new array creation in line 3 in a moment, but for now, focus on the meat of the example. These are the important parts:

- Line 7 creates the mask by performing a vectorized Boolean computation, taking each element and checking to see if it divides evenly by four. This returns a mask array of the same shape with the element-wise results of the computation.
- Line 13 uses this mask to index into the original numbers array. This causes the array to lose

its original shape, reducing it to one dimension, but you still get the data you're looking for.

- Line 13 provides a more traditional, idiomatic masked selection that you might see in the wild, with an anonymous filtering array created inline, inside the selection brackets.

> This syntax is similar to usage in the R programming language. A language used predominantly in data science.

Coming back to line 3, you encounter three (3) new concepts:

- Using `np.linspace()` to generate an evenly spaced array
- Setting the `dtype` of an output
- Reshaping an array with `-1`

np.linspace() generates n numbers evenly distributed between a minimum and a maximum, which is useful for evenly distributed sampling in scientific plotting.

Because of the particular calculation in this example, it makes life easier to have integers in the numbers array. But because the space between 5 and 50 doesn't divide evenly by 24, the resulting numbers would be floating-point numbers. You specify a dtype of int to force the function to round down and give you whole integers. You'll see a more detailed discussion of data types later on.

Finally, array.reshape() can take -1 as one of its dimension sizes. That signifies that NumPy should just figure out how big that particular axis needs to be based on the size of the other axes. In this case, with 24 values and a size of 4 in axis 0, axis 1 ends up with a size of 6.

> **Exercise** 3.1: A Simple Filter Exercise
>
> Write a NumPy program to extract all numbers from a given array less and greater than a specified number.
>
> **Solution**

C.R. 9

python

```python
# Importing the NumPy library with an alias 'np'
import numpy as np

# Creating a NumPy array 'nums' containing values in a 3x3 matrix
nums = np.array([[5.54, 3.38, 7.99],
                 [3.54, 4.38, 6.99],
                 [1.54, 2.39, 9.29]])

# Printing a message indicating the original array
print("Original array:")
print(nums)

# Assigning value 5 to the variable 'n' and printing elements greater than 'n' in the array
n = 5
print("\nElements of the said array greater than", n)
```

C.R. 10
python
```
print(nums[nums > n])

# Assigning value 6 to the variable 'n' and printing elements less than 'n' in the array
n = 6
print("\nElements of the said array less than", n)
print(nums[nums < n])
```

text
```
Original array:
[[5.54 3.38 7.99]
 [3.54 4.38 6.99]
 [1.54 2.39 9.29]]
Elements of the said array greater than 5
[5.54 7.99 6.99 9.29]
Elements of the said array less than 6
[5.54 3.38 3.54 4.38 1.54 2.39]
```

Here's one more example to show off the power of masked filtering. The normal distribution is a probability distribution in which roughly 95.45% of values occur within two standard deviations of the mean.

You can verify that with a little help from NumPy's random module for generating random values:

C.R. 11
python
```
import numpy as np

from numpy.random import default_rng

rng = default_rng()

values = rng.standard_normal(10000)

print(values[:5])

std = values.std()

print(std)

filtered = values[(values > -2 * std) & (values < 2 * std)]

print(filtered.size)

print(values.size)

filtered.size / values.size
```

text
```
[-0.7239233   1.71462297 -1.21091617  0.22540724  0.89326428]
1.005225298547061
9545
10000
```

Here you use a potentially strange-looking syntax to combine filter conditions: a binary & operator.

Why would that be the case? It's because NumPy designates & and | as the vectorized, element-wise operators to combine Booleans. If you try to do A and B, then you'll get a warning about how the truth value for an array is weird, because the and is operating on the truth value of the whole array, not element by element.

Exercise 3.2: Creating a Numpy Array

Write a NumPy program to create an array of integers from 30 to 70.

Solution

C.R. 12

python

```python
# Importing the NumPy library with an alias 'np'
import numpy as np

# Creating an array of integers from 30 to 70 using np.arange()
array = np.arange(30, 71)

# Printing a message indicating an array of integers from 30 to 70
print("Array of the integers from 30 to 70")

# Printing the array of integers from 30 to 70
print(array)
```

text

```
Array of the integers from 30 to 70

[30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70]
```

Exercise 3.3: An Identity Matrix

Write a 3-by-3 identity matrix.

Solution

C.R. 13

python

```python
# Importing the NumPy library with an alias 'np'
import numpy as np

# Creating a 3x3 identity matrix using np.identity()
array_2D = np.identity(3)

# Printing a message indicating a 3x3 matrix
print('3x3 matrix:')

# Printing the 3x3 identity matrix
print(array_2D)

```

text

```
3x3 matrix:
[[1. 0. 0.]
 [0. 1. 0.]
```

```
[0. 0. 1.]]
```

Exercise 3.4: A Random Value

Write a NumPy program to generate a random number between 0 and 1.

Solution

C.R. 14

```python
# Importing the NumPy library with an alias 'np'
import numpy as np

# Generating a random number from a normal distribution with mean 0 and standard deviation 1
↪ using np.random.normal()
rand_num = np.random.normal(0, 1, 1)

# Printing a message indicating a random number between 0 and 1
print("Random number between 0 and 1:")

# Printing the generated random number
print(rand_num)

```

```text
Random number between 0 and 1:
[0.805393]
```

Exercise 3.5: A Random Array

Write a NumPy program to generate an array of 15 random numbers from a standard normal distribution.

Solution

C.R. 15

```python
# Importing the NumPy library with an alias 'np'
import numpy as np

# Generating an array of 15 random numbers from a standard normal distribution using
↪ np.random.normal()
rand_num = np.random.normal(0, 1, 15)

# Printing a message indicating 15 random numbers from a standard normal distribution
print("15 random numbers from a standard normal distribution:")

# Printing the array of 15 random numbers
print(rand_num)

```

```text
15 random numbers from a standard normal distribution:

[ 0.83730584  0.0962134   0.77050723  1.03140118 -0.67267708  1.84883332
  0.70835831 -2.30154701 -0.00770623 -0.7585212  -1.3338401  -0.47316322
```

```
5    0.20031869 -0.63787389  1.25788111]
```

3.7 Transposing, Sorting, and Concatenating

Other manipulations, while not quite as common as indexing or filtering, can also be very handy depending on the situation you're in. You'll see a few examples in this section.

Here's transposing an array:

C.R. 16

python

```python
import numpy as np

a = np.array([
    [1, 2],
    [3, 4],
    [5, 6],
])

print(a.T)

print(a.transpose())
```

text

```
[[1 3 5]
 [2 4 6]]

[[1 3 5]
 [2 4 6]]
```

When you calculate the transpose of an array, the row and column indices of every element are switched. Item [0, 2], for example, becomes item [2, 0]. You can also use a.T as an alias for a.transpose().

The following code block shows sorting, but you'll also see a more powerful sorting technique in the coming section on structured data:

C.R. 17

python

```python
import numpy as np

data = np.array([
    [7, 1, 4],
    [8, 6, 5],
    [1, 2, 3]
])

print(np.sort(data))

print(np.sort(data, axis=None))

print(np.sort(data, axis=0))
```

```text
[[1 4 7]
 [5 6 8]
 [1 2 3]]
[1 1 2 3 4 5 6 7 8]
[[1 1 3]
 [7 2 4]
 [8 6 5]]
```

Omitting the axis argument automatically selects the last and innermost dimension, which is the rows in this example. Using None flattens the array and performs a global sort. Otherwise, you can specify which axis you want. In output 5, each column of the array still has all of its elements but they have been sorted low-to-high inside that column.

Finally, here's an example of concatenation. While there's a np.concatenate() function, there are also a number of helper functions that are sometimes easier to read.

Here are some examples:

C.R. 18

```python
import numpy as np

a = np.array([
    [4, 8],
    [6, 1]
])

b = np.array([
    [3, 5],
    [7, 2],
])

print(np.hstack((a, b)))

print(np.vstack((b, a)))

print(np.concatenate((a, b)))

print(np.concatenate((a, b), axis=None))
```

```text
[[4 8 3 5]
 [6 1 7 2]]
[[3 5]
 [7 2]
 [4 8]
 [6 1]]
[[4 8]
 [6 1]
 [3 5]
 [7 2]]
[4 8 6 1 3 5 7 2]
```

Inputs 4 and 5 show the slightly more intuitive functions hstack() and vstack(). Inputs 6 and 7 show the more generic concatenate(), first without an axis argument and then with axis=None. This flattening behavior is similar in form to what you just saw with sort().

One important stumbling block to note is that all these functions take a tuple of arrays as their first argument rather than a variable number of arguments as you might expect. You can tell because there's an extra pair of parentheses.

Exercise 3.6: Sum of an Array

Write a NumPy program to compute the sum of all elements, the sum of each column and the sum of each row in a given array.

Solution

C.R. 19

python

```python
# Importing the NumPy library with an alias 'np'
import numpy as np

# Creating a NumPy array 'x' with a 2x2 shape
x = np.array([[0, 1], [2, 3]])

# Printing a message indicating the original array 'x'
print("Original array:")
print(x)

# Calculating and printing the sum of all elements in the array 'x' using np.sum()
print("Sum of all elements:")
print(np.sum(x))

# Calculating and printing the sum of each column in the array 'x' using np.sum() with axis=0
print("Sum of each column:")
print(np.sum(x, axis=0))

# Calculating and printing the sum of each row in the array 'x' using np.sum() with axis=1
print("Sum of each row:")
print(np.sum(x, axis=1))

```

text

```
Original array:
[[0 1]
 [2 3]]
Sum of all elements:
6
Sum of each column:
[2 4]
Sum of each row: [1 5]
```

3.8 Aggregating

Your last stop on this tour of functionality before diving into some more advanced topics and examples is aggregation. You've already seen quite a few aggregating methods, including .sum(), .max(), .mean(), and .std(). You can reference NumPy's larger library of functions to see more. Many of the mathematical, financial, and statistical functions use aggregation to help you reduce the number of dimensions in your data.

3.9 Practical Exercise Implementing Maclauring Series

Now it's time to see a realistic use case for the skills introduced in the sections above: implementing an equation.

One of the hardest things about converting mathematical equations to code without NumPy is that many of the visual similarities are missing, which makes it hard to tell what portion of the equation you're looking at as you read the code. Summations are converted to more verbose for loops, and limit optimizations end up looking like while loops.

Using NumPy allows you to keep closer to a one-to-one representation from equation to code.

In this next example, you'll encode the Maclaurin series[6] for ex. Maclaurin series are a way of approximating more complicated functions with an infinite series of summed terms centered about zero.

[6] Computers actually use infinite sums like these to approximate functions like sin, cos

For e^x, the Maclaurin series is the following summation:

$$e^x = \sum_{n=0}^{\infty} = \frac{x^n}{n!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6}$$

You add up terms starting at zero and going theoretically to infinity. Each nth term will be x raised to n and divided by n!, which is the notation for the factorial operation.

Now it's time for you to put that into NumPy code.

C.R. 20

python

```python
from math import e, factorial

import numpy as np

fac = np.vectorize(factorial)

def e_x(x, terms=10):
    """Approximates e^x using a given number of terms of
    the Maclaurin series
    """
    n = np.arange(terms)
    return np.sum((x ** n) / fac(n))

if __name__ == "__main__":
    print("Actual:", e ** 3)  # Using e from the standard library

    print("N (terms)\tMaclaurin\tError")

    for n in range(1, 14):
        maclaurin = e_x(3, terms=n)
        print(f"{n}\t\t{maclaurin:.03f}\t\t{e**3 - maclaurin:.03f}")
```

When you run this, you should see the following result:

As you increase the number of terms, your Maclaurin value gets closer and closer to the actual value, and your error shrinks smaller and smaller.

The calculation of each term involves taking x to the n power and dividing by n!, or the factorial of n. Adding, summing, and raising to powers are all operations that NumPy can vectorize automatically and quickly, but not so for factorial().

To use factorial() in a vectorized calculation, you have to use np.vectorize() to create a vectorized version. The documentation for np.vectorize() states that it's little more than a thin wrapper that applies a for loop to a given function. There are no real performance benefits from using it instead of normal Python code, and there are potentially some overhead penalties. However, as you'll see in a moment, the readability benefits are huge.

Once your vectorized factorial is in place, the actual code to calculate the entire Maclaurin series is shockingly short. It's also readable. Most importantly, it's almost exactly one-to-one with how the mathematical equation looks:

C.R. 21

```python
n = np.arange(terms)
return np.sum((x ** n) / fac(n))
```

This is such an important idea that it deserves to be repeated. With the exception of the extra line to initialize n, the code reads almost exactly the same as the original math equation. No for loops, no temporary i, j, k variables. Just plain, clear, math.

1 Just like that, you're using NumPy for mathematical programming! For extra practice, try picking one of the other Maclaurin series and implementing it in a similar way.

3.10 Optimizing Storage: Data Types

Now that you have a bit more practical experience, it's time to go back to theory and look at **data types**. Data types don't play a central role in a lot of Python code. Numbers work like they're supposed to, strings do other things, Booleans are true or false, and other than that, you make your own objects and collections.

In NumPy, though, there's a little more detail that needs to be covered. NumPy uses C code under the hood to optimize performance, and it can't do that unless all the items in an array are of the same type. That doesn't just mean the same Python type. They have to be the same underlying C type, with the same shape and size in bits!

3.11 Numerical Types

Since most of your data science and numerical calculations will tend to involve numbers, they seem like the best place to start. There are essentially four numerical types in NumPy code, and each one can take a few different sizes.

The table below breaks down the details of these types:

Name	Number of Bites	Python Type	Numpy Types
Integer	64	`int`	`np.int_`
Booleans	8	`bool`	`np.bool_`
Float	64	`float`	`np.float_`
Complex	128	`complex`	`np.complex_`

Table 3.1: The data types supported by Python and Numpy.

These are just the types that map to existing Python types. NumPy also has types for the smaller-sized versions of each, like 8-, 16-, and 32-bit integers, 32-bit single-precision floating-point numbers, and 64-bit single-precision complex numbers. The documentation lists them in their entirety.

To specify the type when creating an array, you can provide a dtype argument

Exercise 3.7: Find the Missing Values

Write a NumPy program to find missing data in a given array.

Solution

C.R. 22

python

```python
# Importing the NumPy library with an alias 'np'
import numpy as np

# Creating a NumPy array 'nums' with provided values, including NaN (Not a Number)
nums = np.array([[3, 2, np.nan, 1],
                 [10, 12, 10, 9],
                 [5, np.nan, 1, np.nan]])

# Printing a message indicating the original array 'nums'
print("Original array:")
print(nums)

# Printing a message indicating finding the missing data (NaN) in the array using np.isnan()
# This function returns a boolean array of the same shape as 'nums', where True represents NaN values
print("\nFind the missing data of the said array:")
print(np.isnan(nums))
```

```text
Original array:
[[ 3.  2. nan  1.]
 [10. 12. 10.  9.]
 [ 5. nan  1. nan]]
Find the missing data of the said array:
[[False False  True False]
 [False False False False]
 [False  True False  True]]
```

Exercise 3.8: Array Value Parity

Write a NumPy program to check whether two arrays are equal (element wise) or not.

Solution

C.R. 23

```python
# Importing the NumPy library with an alias 'np'
import numpy as np

# Creating NumPy arrays 'nums1' and 'nums2' with floating-point values
nums1 = np.array([0.5, 1.5, 0.2])
nums2 = np.array([0.4999999999, 1.500000000, 0.2])

# Setting print options to display floating-point precision up to 15 decimal places
np.set_printoptions(precision=15)

# Printing a message indicating the original arrays 'nums1' and 'nums2'
print("Original arrays:")
print(nums1)
print(nums2)

# Printing a message asking whether the two arrays are equal element-wise or not
print("\nTest said two arrays are equal (element wise) or not:?")
print(nums1 == nums2)

# Reassigning new values to arrays 'nums1' and 'nums2'
nums1 = np.array([0.5, 1.5, 0.23])
nums2 = np.array([0.4999999999, 1.5000000001, 0.23])

# Printing a message indicating the original arrays 'nums1' and 'nums2'
print("\nOriginal arrays:")
np.set_printoptions(precision=15)
print(nums1)
print(nums2)

# Printing a message asking whether the two arrays are equal element-wise or not
print("\nTest said two arrays are equal (element wise) or not:?")
print(np.equal(nums1, nums2))
```

```text
Original arrays:
[0.5 1.5 0.2]
[0.4999999999 1.5          0.2        ]
```

```
Test said two arrays are equal (element wise) or not:?
[False  True  True]
Original arrays:
[0.5  1.5  0.23]
[0.4999999999 1.5000000001 0.23       ]
Test said two arrays are equal (element wise) or not:?
[False False  True]
```

Chapter 4

Scipy

Table of Contents

4.1 Introduction 103
4.2 Special Functions 105
4.3 Integration 107
4.4 Ordinary Differential Equations 110
4.5 Fourier Transform 115
4.6 Linear Algebra 118
4.7 Interpolation 121
4.8 Statistics 123

4.1 Introduction

The SciPy framework builds on top of the low-level NumPy framework for multidimensional arrays, and provides a large number of higher-level scientific algorithms. Some of the topics that SciPy covers are shown in **Table** 4.1.

Each of these submodules provides a number of functions and classes that can be used to solve problems in their respective topics. In this chapter we will look at how to use some of these subpackages. However, the reader is encouraged to look at other aspects of the package.

To access the SciPy package in a Python program, we start by importing everything from the scipy module.

C.R. 1
```python
from scipy import *
```

Topic	Module
Special functions	`scipy.special`
Integration	`scipy.integrate`
Optimization	`scipy.optimize`
Interpolation	`scipy.interpolate`
Fourier Transforms	`scipy.fftpack`
Signal Processing	`scipy.signal`
Linear Algebra	`scipy.linalg`
Sparse Eigenvalue Problems	`scipy.sparse`
Statistics	`scipy.stats`
Multi-dimensional image processing	`scipy.ndimage`
File IO	`scipy.io`

Table 4.1: Scipy offers a wide variety of packages the user can work with. Above are some of them but the reader is recommended to look at other packages described within the documentation.

If we only need to use part of the SciPy framework we can selectively include only those modules we are interested in. For example, to include the linear algebra package under the name `la` [1], we can do:

[1] As usual, it is up to the programmer to pick their alias, but some aliases have standard notation which is commonly accepted such as np for numpy.[1]

C.R. 2
```python
import scipy.linalg as la
```

4.2 Special Functions

A large number of mathematical special functions are important for many computational physics problems. SciPy provides implementations of a very extensive set of special functions. For more information on the list of functions, the reference documentation is at

> Special functions are particular mathematical functions that have more or less established names and notations due to their importance in mathematical analysis, functional analysis, geometry, physics, or other applications.
>
> The term is defined by consensus, and thus lacks a general formal definition.

To demonstrate the typical usage of special functions we will look in more detail at the Bessel functions:

4.2.1 The Bessel Function

Bessel functions, first defined by the mathematician Daniel Bernoulli and then generalised by *Friedrich Bessel*, are canonical solutions $y(x)$ of Bessel's differential equation:

$$x^2 \frac{d^2y}{dx^2} + x\frac{dy}{dx} + \left(x^2 - \alpha^2\right) y = 0$$

where for an arbitrary complex number α, represents the **order** of the Bessel function. Although α and $-\alpha$ produce the same differential equation, it is conventional to define different Bessel functions for these two values in such a way that the Bessel functions are mostly smooth functions of α. The Bessel functions of the first kind (J_{ff}) with different orders can be seen in **Fig.** 4.1.

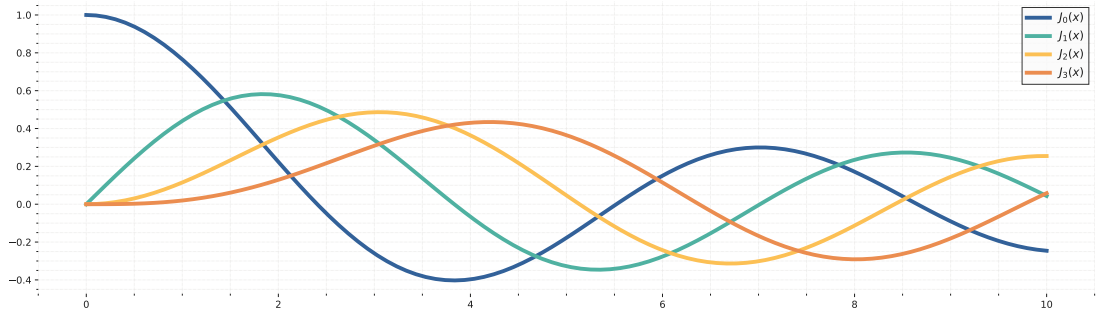


Figure 4.1: The Bessel functions of the first kind (J_{ff}) with different orders.

The most important cases are when α is an integer or half-integer. Bessel functions for integer α are also known as cylinder functions or the cylindrical harmonics because they appear in the solution to Laplace's equation in cylindrical coordinates. Spherical Bessel functions with half-integer α are obtained when solving the Helmholtz equation in spherical coordinates.

C.R. 3 python

```python
#
# The scipy.special module includes a large number of Bessel-functions
# Here we will use the functions jn and yn, which are the Bessel functions
# of the first and second kind and real-valued order. We also include the
# function jn_zeros and yn_zeros that gives the zeroes of the functions jn
# and yn.
#
from scipy.special import jn, yn, jn_zeros, yn_zeros

```

C.R. 4 python

```python
n = 0    # order
x = 0.0

# Bessel function of first kind
print("J_%d(%f) = %f" % (n, x, jn(n, x)))

x = 1.0
# Bessel function of second kind
print("Y_%d(%f) = %f" % (n, x, yn(n, x)))
```

text

```
J_0(0.000000) = 1.000000
Y_0(1.000000) = 0.088257
```

C.R. 5 python

```python
# zeros of Bessel functions
n = 0 # order
m = 4 # number of roots to compute
jn_zeros(n, m)

```

4.3 Integration

In mathematics, an integral is the continuous analog of a sum, which is used to calculate areas, volumes, and their generalizations. Integration, the process of computing an integral, is one of the two fundamental operations of calculus, the other being differentiation. Integration was initially used to solve problems in mathematics and physics, such as finding the area under a curve, or determining displacement from velocity. Usage of integration expanded to a wide variety of scientific fields thereafter.

4.3.1 Numerical Integration

Numerical integration comprises a broad family of algorithms for calculating the numerical value of a definite integral. The term numerical quadrature (often abbreviated to quadrature) is more or less a synonym for *numerical integration*, especially as applied to one-dimensional integrals. Some function have no integration (i.e, no anti-derivative) and it is necessary to approximate the results using numerical methods. The application of numerical integration is an indispensable tool for an engineer.

Numerical evaluation of a function of the type

$$\int_a^b f(x)\,\mathrm{d}x$$

is called numerical quadrature, or simply quadature. SciPy provides a series of functions for different kind of quadrature, for example the `quad`, `dblquad` and `tplquad` for single, double and triple integrals, respectively.

C.R. 6
```python
from scipy.integrate import quad, dblquad, tplquad
```

The quad function takes a large number of optional arguments, which can be used to fine-tune the behaviour of the function [2].

[2] To get more information you can use `help(quad)`.

The basic usage is as follows. We first define a function as such:

C.R. 7
```python
# define a simple function for the integrand
def f(x):
    return x
```

Then we define our limit for the integral and invoke the `quad()` function.

C.R. 8
```python
x_lower = 0 # the lower limit of x
x_upper = 1 # the upper limit of x

val, abserr = quad(f, x_lower, x_upper)
```

C.R. 9 python
```
print("integral value =", val, ", absolute error =", abserr)
```

text
```
integral value = 0.5 , absolute error = 5.551115123125783e-15
```

If we need to pass extra arguments to integrand function we can use the `args` keyword argument:

C.R. 10 python
```
def integrand(x, n):
    """
    Bessel function of first kind and order n.
    """
    return jn(n, x)


x_lower = 0  # the lower limit of x
x_upper = 10 # the upper limit of x

val, abserr = quad(integrand, x_lower, x_upper, args=(3,))

print(val, abserr)
```

text
```
0.7366751370811071 9.389256877192047e-13
```

[3] A lambda function is a small anonymous function. A lambda function can take any number of arguments, but can only have one expression.

For simple functions we can use a lambda[3] function (name-less function) instead of explicitly defining a function for the integrand:

C.R. 11 python
```
val, abserr = quad(lambda x: np.exp(-x ** 2), -np.Inf, np.Inf)

print("numerical  =", val, abserr)

analytical = sqrt(pi)

print("analytical =", analytical)
```

text
```
numerical  = 1.7724538509055159 1.4202637059452923e-08
```

As show in the example above, we can also use '`Inf`' or '`-Inf`' as integral limits. Higher-dimensional integration works in the same way:

C.R. 12 python
```
def integrand(x, y):
    return np.exp(-x**2-y**2)

x_lower = 0
x_upper = 10
y_lower = 0
y_upper = 10
```

C.R. 13

python

```python

val, abserr = dblquad(integrand, x_lower, x_upper, lambda x : y_lower, lambda x: y_upper)

print(val, abserr)

```

text

```
0.7853981633974476 1.3753098510206357e-08
```

we had to pass lambda functions for the limits for the y integration, since these in general can be functions of x.

4.4 Ordinary Differential Equations

An ordinary differential equation (ODE) is a differential equation dependent on only a single independent variable. As with other DE, its unknown(s) consists of one (or more) function(s) and involves the derivatives of those functions.

SciPy provides two (2) different ways to solve ODEs:

1. An API based on the function `odeint`,
2. object-oriented API based on the class `ode`.

> Usually `odeint` is easier to get started with, but the `ode` class offers some finer level of control.

Here we will use the `odeint` functions. For more information about the class ode, try `help(ode)`. It does pretty much the same thing as odeint, but in an object-oriented fashion. To use `odeint`, first import it from the `scipy.integrate` module.

C.R. 14

```python
from scipy.integrate import odeint, ode
import numpy as np
```

A system of ODEs are usually formulated on standard form before it is attacked numerically. The standard form is:

$$y' = f(x, y)$$

where

$$y = \left[y_1(t), y_2(t), \ldots, y_n(t)\right]$$

and f is some function that gives the derivatives of the function $y_i(t)$. To solve an ODE we need to know the function f and an initial condition $y(0)$.

> Note that higher-order ODEs can always be written in this form by introducing new variables for the intermediate derivatives

Once we have defined the Python function f and array y_0, we can use the `odeint` function as:

```
y_t = odeint(f, y_0, t)
```

where t is and array with time-coordinates for which to solve the ODE problem, `y_t` is an array with one row for each point in time in t, where each column corresponds to a solution `y_i(t)` at that point in time.

We will see how we can implement f and `y_0` in Python code in the examples below.

Let's consider a physical examples.

4.4.1 Double Pendulum

a double pendulum, also known as a chaotic pendulum, is a pendulum with another pendulum attached to its end, forming a simple physical system that exhibits rich dynamic behavior with a strong sensitivity to initial conditions. The motion of a double pendulum is governed by a pair of coupled ordinary differential equations and is chaotic. A model showcasing the setup can be seen in **Fig.** 4.1.

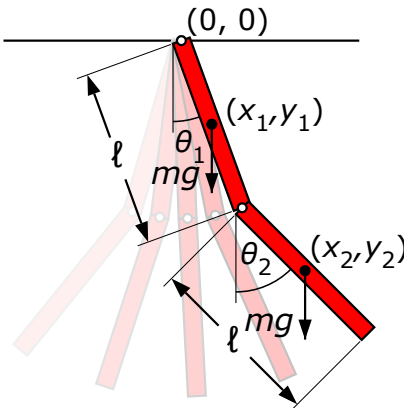


Figure 4.2: A model of the double compound pendulum with physical description of its parameters.

We are not going to bother deriving the solution and instead use the resources on the wiki page to derive the solution:

$$
\begin{aligned}
\dot{\theta_1} &= \frac{6}{m\ell^2}\frac{2p_{\theta_1} - 3\cos(\theta_1 - \theta_2)p_{\theta_2}}{16 - 9\cos^2(\theta_1 - \theta_2)}, \\
\dot{\theta_2} &= \frac{6}{m\ell^2}\frac{8p_{\theta_2} - 3\cos(\theta_1 - \theta_2)p_{\theta_1}}{16 - 9\cos^2(\theta_1 - \theta_2)}, \\
\dot{p_{\theta_1}} &= -\frac{1}{2}m\ell^2\left[\dot{\theta}_1\dot{\theta}_2\sin(\theta_1 - \theta_2) + 3\frac{g}{\ell}\sin\theta_1\right], \\
\dot{p_{\theta_2}} &= -\frac{1}{2}m\ell^2\left[-\dot{\theta}_1\dot{\theta}_2\sin(\theta_1 - \theta_2) + \frac{g}{\ell}\sin\theta_2\right].
\end{aligned}
$$

To make the Python code simpler to follow, let's introduce new variable names and the vector notation $x = \left[\theta_1, \theta_2, p_{\dot{}_1}, p_{\dot{}_2}\right]$. We can now write our function describing the problem.

$$
\begin{aligned}
\dot{x_1} &= \frac{6}{m\ell^2}\frac{2x_3 - 3\cos(x_1 - x_2)x_4}{16 - 9\cos^2(x_1 - x_2)}, \\
\dot{x_2} &= \frac{6}{m\ell^2}\frac{8x_4 - 3\cos(x_1 - x_2)x_3}{16 - 9\cos^2(x_1 - x_2)}, \\
\dot{x_3} &= -\frac{1}{2}m\ell^2\left[\dot{x_1}\dot{x_2}\sin(x_1 - x_2) + 3\frac{g}{\ell}\sin x_1\right], \\
\dot{x_4} &= -\frac{1}{2}m\ell^2\left[-\dot{x_1}\dot{x_2}\sin(x_1 - x_2) + \frac{g}{\ell}\sin x_2\right].
\end{aligned}
$$

C.R. 15 python

```python
g = 9.82
L = 0.5
m = 0.1

def dx(x, t):
    """
    The right-hand side of the pendulum ODE
    """
    x1, x2, x3, x4 = x[0], x[1], x[2], x[3]

    dx1 = 6.0/(m*L**2) * (2 * x3 - 3 * np.cos(x1-x2) * x4)/(16 - 9 * np.cos(x1-x2)**2)
    dx2 = 6.0/(m*L**2) * (8 * x4 - 3 * np.cos(x1-x2) * x3)/(16 - 9 * np.cos(x1-x2)**2)
    dx3 = -0.5 * m * L**2 * ( dx1 * dx2 * np.sin(x1-x2) + 3 * (g/L) * np.sin(x1))
    dx4 = -0.5 * m * L**2 * (-dx1 * dx2 * np.sin(x1-x2) + (g/L) * np.sin(x2))

    return [dx1, dx2, dx3, dx4]
```

We then need to tell the system where we are going to start with our simulation:

C.R. 16 python

```python
# choose an initial state
x0 = [np.pi/4, np.pi/2, 0, 0]
```

Continuing on, we need to describe the time-scale in which the simulation will take place and solve the differential equation:

C.R. 17 python

```python
# time coordinate to solve the ODE for: from 0 to 10 seconds
t = np.linspace(0, 10, 250)
```

C.R. 18 python

```python
# solve the ODE problem
x = odeint(dx, x0, t)
```

We then can finally plot our result and see for ourselves.

C.R. 19 python

```python
# plot the angles as a function of time

fig, axes = plt.subplots(1,2, figsize=(36,10))
axes[0].plot(t, x[:, 0], 'r', label="theta1")
```

C.R. 20

```python
axes[0].plot(t, x[:, 1], 'b', label="theta2")

x1 = + L * np.sin(x[:, 0])
y1 = - L * np.cos(x[:, 0])

x2 = x1 + L * np.sin(x[:, 1])
y2 = y1 - L * np.cos(x[:, 1])

axes[1].plot(x1, y1, label="pendulum1")
axes[1].plot(x2, y2, label="pendulum2")
axes[1].set_ylim([-1, 0])
axes[1].set_xlim([1, -1]);
```

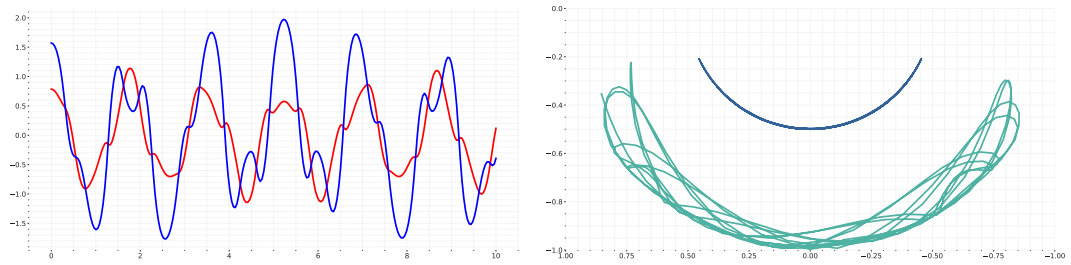


Figure 4.3: The results of the double pendulum experiment. **(a)** The angles of θ_1 and θ_2 with respect to time **(b)** The spatial position of the pendulum as it swings around its pivot point.

4.4.2 Damped Harmonic Oscillator

ODE problems are important in computational physics, so we will look at one more example: the damped harmonic oscillation. This problem is well described on the wiki page:

The equation of motion for the damped oscillator is:

$$\frac{d^2x}{dt^2} + 2\zeta\omega_0\frac{dx}{dt} + \omega_0^2 x = 0$$

where x is the position of the oscillator, ω_0 is the frequency, and ζ is the damping ratio. To write this second-order ODE on standard form we introduce $p = dx/dt$.

$$\frac{dp}{dt} = -2\zeta\omega_0 p - \omega_0^2 x$$
$$\frac{dx}{dt} = p$$

In the implementation of this example we will add extra arguments to the right-hand side function for the ODE, rather than using global variables as we did in the previous example. As a consequence of the extra arguments to the right-hand size, we need to pass an keyword argument `args` to the `odeint` function.

We first define a function which takes the ODE defined above and convert to their state space representation:

C.R. 21
python
```
def dy(y, t, zeta, w0):
    """
    The right-hand side of the damped oscillator ODE
    """
    x, p = y[0], y[1]

    dx = p
    dp = -2 * zeta * w0 * p - w0**2 * x

    return [dx, dp]

```

We then write our initial state value for our equation.

C.R. 22
python
```
# initial state:
y0 = [1.0, 0.0]
```

We then define the time axis for our solution to be calculated in.

C.R. 23
python
```
# time coordinate to solve the ODE for
t = np.linspace(0, 10, 1000)
w0 = 2*np.pi*1.0
```

After this we calculate the solutions with different damping ratios:

C.R. 24
python
```
# solve the ODE problem for three different values of the damping ratio
y1 = odeint(dy, y0, t, args=(0.0, w0)) # undamped
y2 = odeint(dy, y0, t, args=(0.2, w0)) # under damped
y3 = odeint(dy, y0, t, args=(1.0, w0)) # critical damping
y4 = odeint(dy, y0, t, args=(5.0, w0)) # over damped
```

Once the calculations are done we can plot our results (we will discuss the plotting in the matplotlib section)

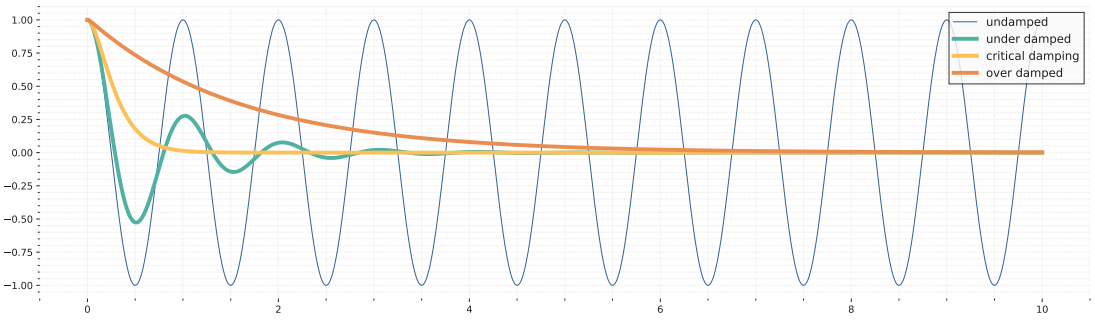


Figure 4.4: In classical mechanics, a harmonic oscillator is a system that, when displaced from its equilibrium position, experiences a restoring force F proportional to the displacement x with k being a positive constant.

4.5 Fourier Transform

In physics, engineering and mathematics, the Fourier transform is an integral transform that takes a function as input and outputs another function that describes the extent to which various frequencies are present in the original function. The output of the transform is a complex-valued function of frequency.

The term Fourier transform refers to both this complex-valued function and the mathematical operation. When a distinction needs to be made, the output of the operation is sometimes called the frequency domain representation of the original function. The Fourier transform is analogous to decomposing the sound of a musical chord into the intensities of its constituent pitches.

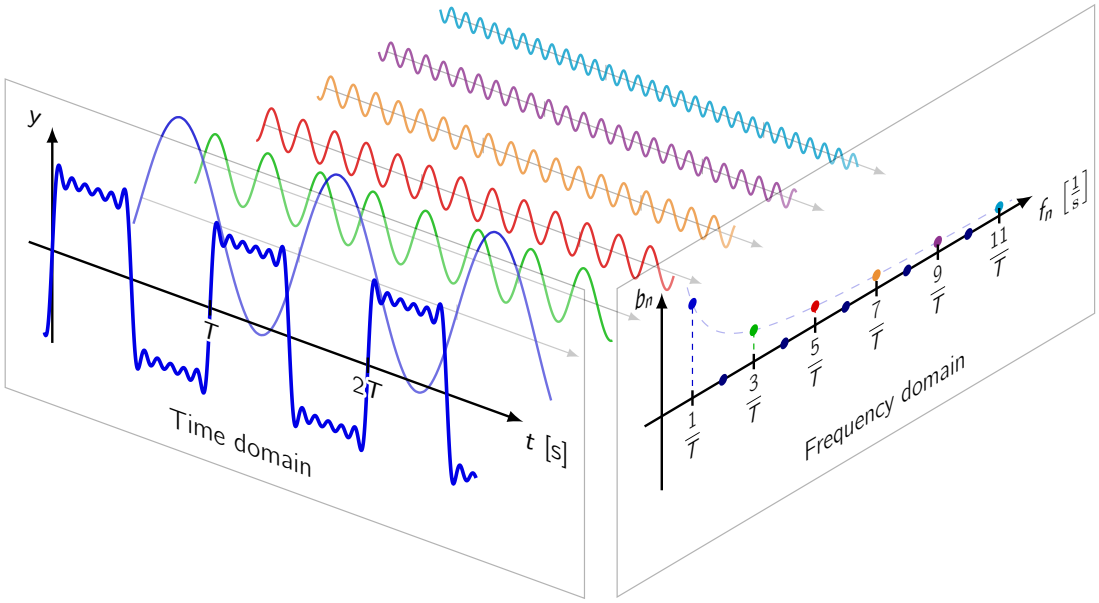


Figure 4.5: Fourier transform can be taught of the deconstruction of a signal to their sine and cosine components with a bias term which is generally known as DC term. The transform takes a signal in its time domain and transform it to their frequency components.

Fourier transforms are one of the **universal tools** in computational physics, which appear over and over again in different contexts. SciPy provides functions for accessing the classic FFTPACK library from NetLib (a repository of software for scientific computing maintained by AT&T, Bell Laboratories, the University of Tennessee and Oak Ridge National Laboratory), which is an efficient and well tested FFT library written in Fortran[4]. The SciPy API has a few additional convenience functions, but overall the API is closely related to the original Fortran library.

[4] A 3rd generation, compiled, imperative programming language that is especially suited to numeric computation and scientific computing.

To use the `fftpack` module in a python program, include it using:

C.R. 25
```python
from numpy.fft import fftfreq
from scipy.fftpack import *
```

To demonstrate how to do a fast Fourier transform with SciPy, let's look at the FFT[5] of the solution to the damped oscillator from the previous section:

[5] allows the decomposition of the signal to its frequency components with high speed. It has been called one of the most important numerical algorithms ever designed

C.R. 26
```python
N = len(t)
dt = t[1]-t[0]

# calculate the fast fourier transform
# y2 is the solution to the under-damped oscillator from the previous section
F = fft(y2[:,0])

# calculate the frequencies for the components in F
w = fftfreq(N, dt)
```

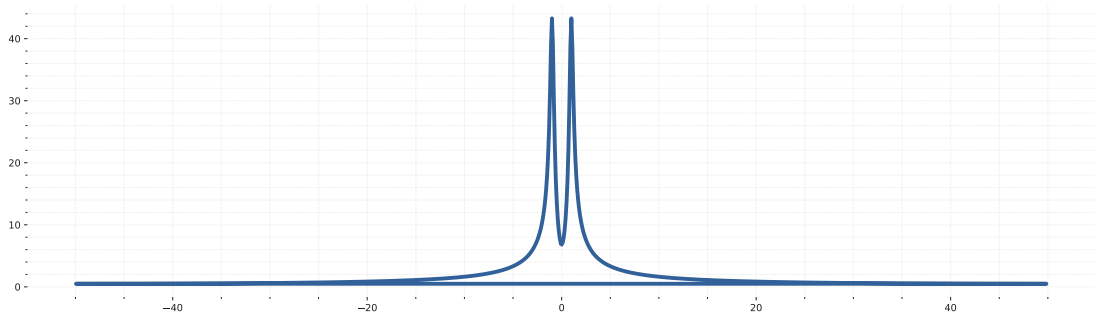


Figure 4.6: The FFT analysis of the solution to the under-damped oscillator from the previous section.

Since the signal is real, the spectrum is symmetric. We therefore only need to plot the part that corresponds to the positive frequencies. To extract that part of the `w` and `F` we can use some of the indexing tricks for NumPy arrays.

C.R. 27
```python
plt.figure().set_figwidth(16)
plt.plot(w, abs(F))
```

C.R. 28
```python
# select only indices for elements that corresponds to positive frequencies
indices = np.where(w > 0)
w_pos = w[indices]
F_pos = F[indices]
```

C.R. 29
```python
plt.figure().set_figwidth(16)
plt.plot(w_pos, abs(F_pos))
cp.store_fig("fft-plot-pos", close=True)
```

As expected, we now see a peak in the spectrum that is centered around 1, which is the frequency we used in the damped oscillator example.

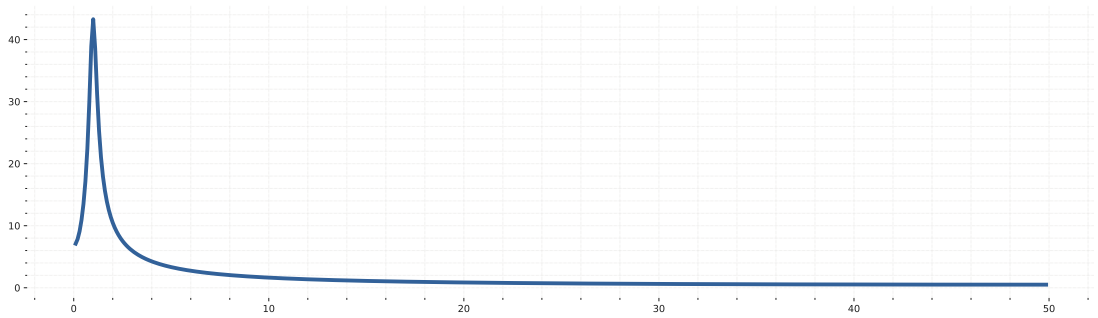


Figure 4.7: As the signal was symmetric, we only need to see the positive side of the spectrum.

4.6 Linear Algebra

The linear algebra module contains a lot of matrix related functions, including:

- linear equation solvers,
- eigenvalue solvers,
- matrix functions (for example matrix-exponentiation),
- a number of different decompositions (SVD, LU, cholesky)

Detailed documentation is available at:

Here we will look at how to use some of these functions:

4.6.1 Linear Equation Systems

Linear equations systems on the matrix form

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

where A is a matrix and x, b are vector can be solved like:

C.R. 30 python
```
from scipy.linalg import *
from numpy import *
```

C.R. 31 python
```
A = array([[1,2,3], [4,5,6], [7,8,9]])
b = array([1,2,3])
```

C.R. 32 python
```
x = solve(A, b)

print(x)
```

text
```
[-0.23333333  0.46666667  0.1       ]
```

We can also do a check to see if our calculations are correct by using numpy:

C.R. 33 python
```
print(dot(A, x) - b)
```

text
```
[0. 0. 0.]
```

We can also do the same with:

$$AX = B$$

where A, B, C are matrices:

C.R. 34 python
```
A = random.rand(3,3)
B = random.rand(3,3)
```

C.R. 35 python
```
X = solve(A, B)

print(X)
```

text
```
[[ 0.68046294  1.66422795  1.26428226]
 [-0.87939673 -2.24202469 -2.59472149]
 [ 1.09333012  1.85637217  1.51844923]]
```

C.R. 36 python
```
print(norm(dot(A, X) - B))
```

C.R. 37 python
```
print(norm(dot(A, X) - B))
```

4.6.2 Eigenvalues and Eigenvectors

The eigenvalue problem for a matrix A:

$$Ax_n = \lambda_n x_n$$

where x is the eigenvector and λ is the eigenvalue. To calculate eigenvalues of a matrix, use the `eigvals` and for calculating both eigenvalues and eigenvectors, use the function `eig`:

C.R. 38 python
```
evals = eigvals(A)
print(evals)
```

text
```
[ 1.37593026+0.j -0.61877061+0.j -0.14902867+0.j]
```

C.R. 39 python
```
evals, evecs = eig(A)
print(evals)
```

```
[ 1.37593026+0.j -0.61877061+0.j -0.14902867+0.j]
```
text

```
print(evecs)
```
C.R. 40
python

```
[[-0.5500703  -0.71043268  0.42417581]
 [-0.45803692  0.52634612 -0.88002878]
 [-0.6983014   0.46716716  0.21359829]]
```
text

The eigenvectors corresponding to the n^{th} eigenvalue (stored in `evals[n]`) is the n^{th} column in `evecs`, i.e., `evecs[:,n]`. To verify this, let's try multiplying eigenvectors with the matrix and compare to the product of the eigenvector and the eigenvalue:

```
n = 1

print(norm(dot(A, evecs[:,n]) - evals[n] * evecs[:,n]))
```
C.R. 41
python

```
1.3597399555105182e-16
```
text

There are also more specialized eigensolvers, like the `eigh` for Hermitian matrices.

4.7 Interpolation

In the mathematical field of numerical analysis, interpolation is a type of estimation, a method of constructing (finding) new data points based on the range of a discrete set of known data points.

In engineering and science, one often has a number of data points, obtained by sampling or experimentation, which represent the values of a function for a limited number of values of the independent variable. It is often required to interpolate; that is, estimate the value of that function for an intermediate value of the independent variable.

A closely related problem is the approximation of a complicated function by a simple function. Suppose the formula for some given function is known, but too complicated to evaluate efficiently. A few data points from the original function can be interpolated to produce a simpler function which is still fairly close to the original. The resulting gain in simplicity may outweigh the loss from interpolation error and give better performance in calculation process.

Interpolation is simple and convenient in scipy: The `interp1d` function, when given arrays describing `X` and `Y` data, returns and object that behaves like a function that can be called for an arbitrary value of x (in the range covered by X), and it returns the corresponding interpolated y value:

C.R. 42 python
```python
from scipy.interpolate import *
```

C.R. 43 python
```python
def f(x):
    return np.sin(x)
```

C.R. 44 python
```python
n = np.arange(0, 10)
x = np.linspace(0, 9, 100)

y_meas = f(n) + 0.1 * np.random.randn(len(n)) # simulate measurement with noise
y_real = f(x)

linear_interpolation = interp1d(n, y_meas)
y_interp1 = linear_interpolation(x)

cubic_interpolation = interp1d(n, y_meas, kind='cubic')
y_interp2 = cubic_interpolation(x)
```

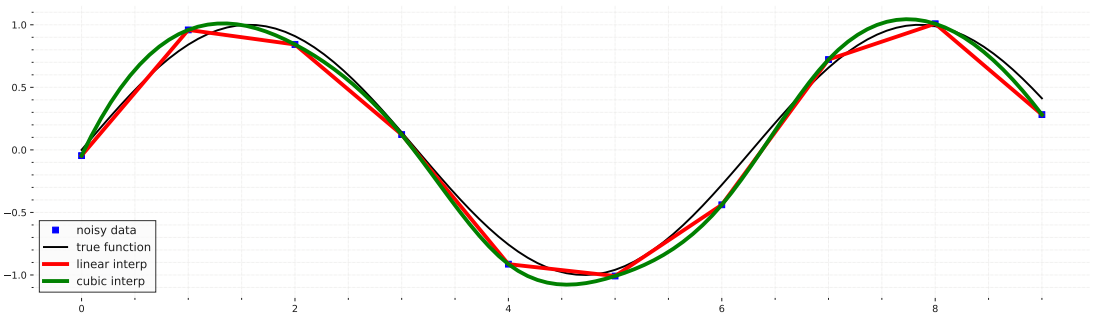


Figure 4.8: An example of interpolation of a function using linear, and cubic interpolation.

4.8 Statistics

The `scipy.stats` module contains a large number of statistical distributions, statistical functions and tests. There is also a very powerful python package for statistical modelling called statsmodels.

C.R. 45 python
```python
from scipy import stats
import matplotlib.pyplot as plt
```

C.R. 46 python
```python
# create a (discrete) random variable with Poissonian distribution

X = stats.poisson(3.5) # photon distribution for a coherent state with n=3.5 photons
```

C.R. 47 python
```python
n = arange(0,15)

fig, axes = plt.subplots(3,1, sharex=True, figsize=(18, 10))

# plot the probability mass function (PMF)
axes[0].step(n, X.pmf(n))

# plot the cumulative distribution function (CDF)
axes[1].step(n, X.cdf(n))

# plot histogram of 1000 random realizations of the stochastic variable X
axes[2].hist(X.rvs(size=1000));

```

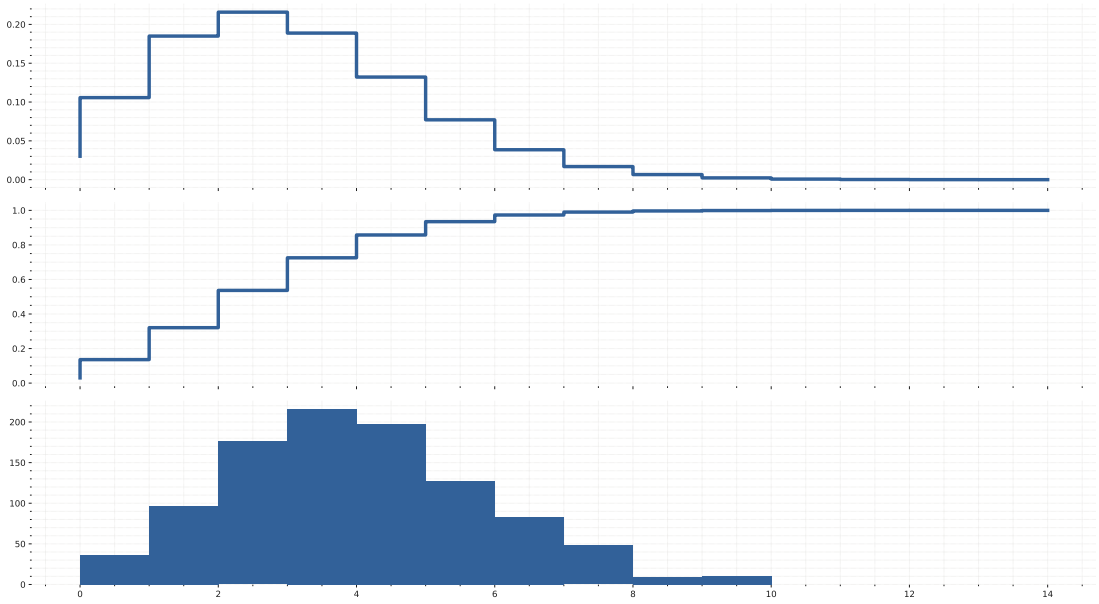


Figure 4.9: he plots of three statistical properties for a discrete distribution. From top to bottom **(a)** Probability density function **(b)** Cumulative density function **(c)** Histogram.

C.R. 48
```python
# create a (continuous) random variable with normal distribution
Y = stats.norm()
```

C.R. 49
```python
x = linspace(-5,5,100)

fig, axes = plt.subplots(3,1, sharex=True, figsize=(18, 10))

# plot the probability distribution function (PDF)
axes[0].plot(x, Y.pdf(x))

# plot the cumulative distribution function (CDF)
axes[1].plot(x, Y.cdf(x));

# plot histogram of 1000 random realizations of the stochastic variable Y
axes[2].hist(Y.rvs(size=1000), bins=50);

```

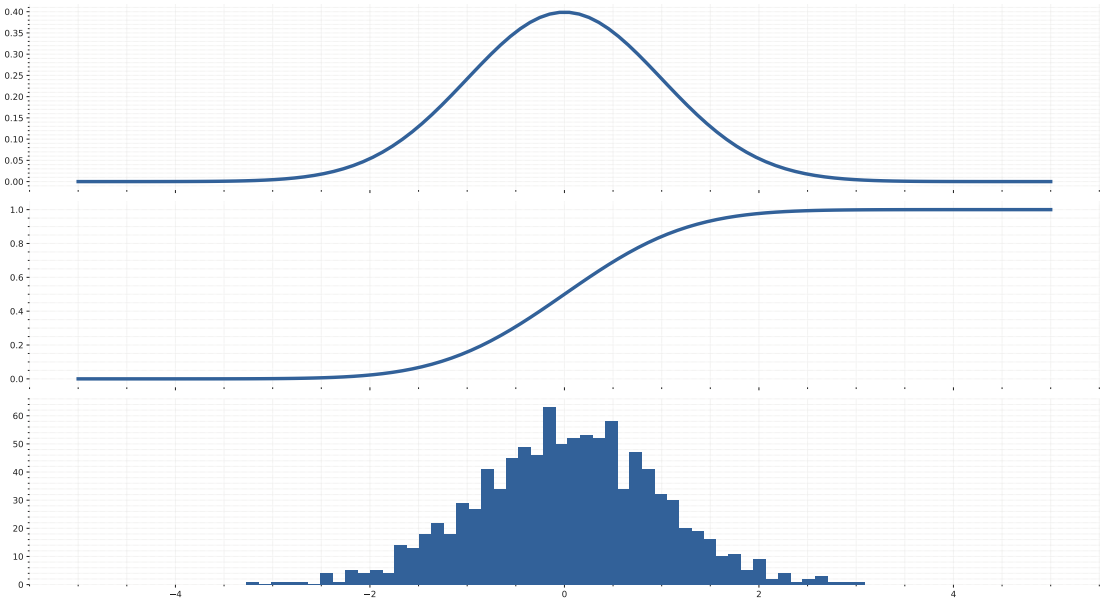


Figure 4.10: The plots of three statistical properties for a continuous distribution. From top to bottom **(a)** Probability density function **(b)** Cumulative density function **(c)** Histogram.

C.R. 50
```python
print(X.mean(), X.std(), X.var()) # Poisson distribution
```

```text
3.5 1.8708286933869707 3.5
```

C.R. 51
```python
print(Y.mean(), Y.std(), Y.var()) # normal distribution
```

```text
0.0 1.0 1.0
```

Chapter 5

SymPy

Table of Contents

5.1 Symbolic Variables 126
5.2 Numerical Evaluation 129
5.3 Algebraic manipulations 131
5.4 Calculus 133
5.5 Linear Algebra 137
5.6 Solving equations 138

There are two (2) notable Computer Algebra Systems (CAS) for Python:

1. **SymPy** A python module that can be used in any Python program, or in an IPython session, that provides powerful CAS features.

2. **Sage** A full-featured and very powerful CAS environment that aims to provide an open source system that competes with Mathematica and Maple.

> Sage is not a regular Python module, but rather a CAS environment that uses Python as its programming language.

> Sage is in some aspects more powerful than SymPy, but both offer very comprehensive CAS functionality. The advantage of SymPy is that it is a regular Python module and integrates well with the IPython notebook.

In this chapter we will therefore look at how to use SymPy with IPython notebooks. If you are interested in an open source CAS environment I also recommend to read more about Sage as it is a great tool to study linear algebra and differential equations.

To get started using SymPy in a Python program or notebook, import the module `sympy`:

```
import sympy as sp # for symbolic calculations
import matplotlib.pyplot as plt # for plotting applications
```

To print them in LaTeX format, we can invoke the following command:

```
sp.init_printing(use_latex=True)
```

5.1 Symbolic Variables

In SymPy we need to create symbols for the variables we want to work with. We can create a new symbol using the `Symbol` class:

```
x = sp.Symbol('x')
```

```
print(sp.latex((sp.pi + x)**2))
```

$$(x + \pi)^2$$

```
# alternative way of defining symbols
a, b, c = sp.symbols("a, b, c")
```

And if we want to look at the type of a data in Python, remember to just use `type()` which will give you the data type of the variable in question.

```
print(type(a))
```

```
<class 'sympy.core.symbol.Symbol'>
```

We can add assumptions to symbols when we create them just write it in the paranthesis as so:

```
x = sp.Symbol('x', real=True)
```

And if we assume a false assumption, we should get `False`.

```
print(x.is_imaginary)
```

```
False
```

We can also do other assumptions:

```
x = sp.Symbol('x', positive=True)
```

And testing this with a `True` statement, we find:

```
print(x > 0)
```

```
True
```

5.1.1 Complex Numbers

The imaginary unit is denoted `I` in SymPy.

Imaginary Numbers An imaginary number is the product of a real number and the imaginary unit i, which is defined by its property $i^2 = -1$. The square of an imaginary number bi is $-b^2$. For example, $5i$ is an imaginary number, and its square is -25. The number zero is considered to be both real and imaginary.

Originally coined in the 17th century by René Descartes as a derogatory term and regarded as fictitious or useless, the concept gained wide acceptance following the work of Leonhard Euler (in the 18th century) and Augustin-Louis Cauchy and Carl Friedrich Gauss (in the early 19th century).

An imaginary number bi can be added to a real number a to form a complex number of the form $a + bi$, where the real numbers a and b are called, respectively, the real part and the imaginary part of the complex number.

```
print(sp.latex(1+1*sp.I))
```

$$1 + i$$

```
print(sp.latex(sp.I**2))
```

$$-1$$

```
print(sp.latex((x * sp.I + 1)**2))
```

$$(ix + 1)^2$$

5.1.2 Rational Numbers

There are three different numerical types in SymPy: `Real`, `Rational`, `Integer`:

Real - Rational - Integer **Real Numbers:** A number that can be used to measure a continuous one-dimensional quantity such as a distance, duration or temperature. Here, continuous means that pairs of values can have arbitrarily small differences.

Rational Number: a rational number is a number that can be expressed as the quotient or fraction $\frac{p}{q}$ of two integers, a numerator p and a non-zero denominator q.

Integer: is the number zero (0), a positive natural number (1, 2, 3, . . .), or the negation of a positive natural number (-1, -2, -3, . . .).

```
r1 = sp.Rational(4,5)
r2 = sp.Rational(5,4)
```

```
print(sp.latex(r1))
```

$$\frac{4}{5}$$

```
print(sp.latex(r1 + r2))
```

$$\frac{41}{20}$$

```
print(sp.latex(r1 / r2))
```

$$\frac{16}{25}$$

5.2 Numerical Evaluation

SymPy uses a library for arbitrary precision as numerical backend, and has predefined SymPy expressions for a number of mathematical constants, such as: `pi`, `e`, `oo` for infinity.

To evaluate an expression numerically we can use the evalf function (or `N`). It takes an argument `n` which specifies the number of significant digits.

```
print(sp.latex(sp.pi.evalf(n=50)))
```

$$3.1415926535897932384626433832795028841971693993751$$

```
y = (x + sp.pi)**2
```

```
print(sp.latex(sp.N(y, 5))) # same as evalf
```

$$9.8696\,(0.31831x + 1)^2$$

When we numerically evaluate algebraic expressions we often want to substitute a symbol with a numerical value. In SymPy we do that using the subs function

```
print(sp.latex(y.subs(x, 1.5)))
```

$$(1.5 + \pi)^2$$

```
print(sp.latex(sp.N(y.subs(x, 1.5))))
```

$$21.5443823618587$$

The `subs` function can of course also be used to substitute Symbols and expressions:

```
print(sp.latex(y.subs(x, a+sp.pi)))
```

$$(a + 2\pi)^2$$

We can also combine numerical evaluation of expressions with NumPy arrays:

```
import numpy
```

```
x_vec = numpy.arange(0, 10, 0.1)
```

```
y_vec = numpy.array([sp.N(((x + sp.pi)**2).subs(x, xx)) for xx in x_vec])
```

```
fig, ax = plt.subplots()
ax.plot(x_vec, y_vec);
plt.savefig("images/sympy/fplot.pdf")
```

However, this kind of numerical evaluation can be very slow, and there is a much more efficient way to do it: Use the function `lambdify` to "compile" a SymPy expression into a function that is much more efficient to evaluate numerically:

```
f = sp.lambdify([x], (x + sp.pi)**2, 'numpy')
# the first argument is a list of variables that
# f will be a function of: in this case only x -> f(x)
```

```
y_vec = f(x_vec)  # now we can directly pass a numpy array and f(x) is efficiently
↪  evaluated
```

5.3 Algebraic manipulations

One of the main uses of an CAS is to perform algebraic manipulations of expressions. For example, we might want to expand a product, factor an expression, or simplify an expression. The functions for doing these basic operations in SymPy are demonstrated in this section.

5.3.1 Expand and Factor

The first steps in an algebraic manipulation

```
print(sp.latex((x+1)*(x+2)*(x+3)))
```

$$(x+1)\,(x+2)\,(x+3)$$

```
print(sp.latex(sp.expand((x+1)*(x+2)*(x+3))))
```

$$x^3 + 6x^2 + 11x + 6$$

The `expand` function takes a number of keywords arguments which we can tell the functions what kind of expansions we want to have performed. For example, to expand trigonometric expressions, use the `trig=True` keyword argument:

```
print(sp.latex(sp.sin(a+b)))
```

$$\sin(a+b)$$

```
print(sp.latex(sp.expand(sp.sin(a+b), trig=True)))
```

$$\sin(a)\cos(b) + \sin(b)\cos(a)$$

See `help(expand)` for a detailed explanation of the various types of expansions the expand functions can perform.

The opposite of product expansion is of course factoring. To factor an expression in SymPy use the `factor` function:

```
print(sp.latex(sp.factor(x**3 + 6 * x**2 + 11*x + 6)))
```

$$(x+1)\,(x+2)\,(x+3)$$

5.3.2 Simplify

The `simplify` tries to simplify an expression into a nice looking expression, using various techniques. More specific alternatives to the `simplify` functions also exists: `trigsimp`, `powsimp`, `logcombine`,

etc. The basic usages of these functions are as follows:

```
# simplify (sometimes) expands a product
print(sp.latex(sp.simplify((x+1)*(x+2)*(x+3))))
```

$$(x+1)(x+2)(x+3)$$

```
# simplify uses trigonometric identities
print(sp.latex(sp.simplify(sp.sin(a)**2 +sp. cos(a)**2)))
```

$$1$$

```
print(sp.latex(sp.simplify(sp.cos(x)/sp.sin(x))))
```

$$\frac{1}{\tan(x)}$$

5.3.3 Apart and Together

```
f1 = 1/((a+1)*(a+2))
```

```
print(sp.latex(f1))
```

$$\frac{1}{(a+1)(a+2)}$$

```
print(sp.latex(sp.apart(f1)))
```

$$-\frac{1}{a+2}+\frac{1}{a+1}$$

```
f2 = 1/(a+2) + 1/(a+3)
```

```
print(sp.latex(f2))
```

$$\frac{1}{a+3}+\frac{1}{a+2}$$

```
print(sp.latex(sp.together(f2)))
```

$$\frac{2a+5}{(a+2)(a+3)}$$

Simplify usually combines fractions but **does not factor**:

```
print(sp.latex(sp.simplify(f2)))
```

$$\frac{2a+5}{(a+2)(a+3)}$$

5.4 Calculus

In addition to algebraic manipulations, the other main use of CAS is to do calculus, like derivatives and integrals of algebraic expressions.

5.4.1 Differentiation

Differentiation is usually simple. Use the `diff` function. The first argument is the expression to take the derivative of, and the second argument is the symbol by which to take the derivative:

```
print(sp.latex(y))
```

$$(x + \pi)^{2}$$

```
print(sp.latex(sp.diff(y**2, x)))
```

$$4\,(x + \pi)^{3}$$

For higher order derivatives we can do:

```
print(sp.latex(sp.diff(y**2, x, x)))
```

$$12\,(x + \pi)^{2}$$

```
print(sp.latex(sp.diff(y**2, x, 2))) # same as above
```

$$12\,(x + \pi)^{2}$$

To calculate the derivative of a multivariate expression, we can do:

```
x, y, z = sp.symbols("x,y,z")
```

```
f = sp.sin(x*y) + sp.cos(y*z)
```

```
print(sp.latex(sp.diff(f, x, 1, y, 2)))
```

$$-x\left(xy\cos\left(xy\right) + 2\sin\left(xy\right)\right)$$

5.4.2 Integration

Integration is done in a similar fashion:

```
print(sp.latex(f))
```

$$\sin(xy) + \cos(yz)$$

```
print(sp.latex(sp.integrate(f, x)))
```

$$x\cos(yz) + \begin{cases} -\frac{\cos(xy)}{y} & \text{for } y \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

By providing limits for the integration variable we can evaluate definite integrals:

```
print(sp.latex(sp.integrate(f, (x, -1, 1))))
```

$$2\cos(yz)$$

and also improper integrals

```
print(sp.latex(sp.integrate(sp.exp(-x**2), (x, -sp.oo, sp.oo))))
```

$$\sqrt{\pi}$$

> Remember, oo is the SymPy notation for inifinity.

5.4.3 Sums and Products

We can evaluate sums using the function Sum:

```
n = sp.Symbol("n")
```

```
print(sp.latex(sp.Sum(1/n**2, (n, 1, 10))))
```

$$\sum_{n=1}^{10} \frac{1}{n^2}$$

```
print(sp.latex(sp.Sum(1/n**2, (n,1, 10)).evalf()))
```

$$1.54976773116654$$

```
print(sp.latex(sp.Sum(1/n**2, (n, 1, sp.oo)).evalf()))
```

$$1.64493406684823$$

Products work much the same way:

```
print(sp.latex(sp.Product(n, (n, 1, 10)))) # 10!
```

$$\prod_{n=1}^{10} n$$

5.4.4 Limits

Limits can be evaluated using the `limit` function. For example,

```
print(sp.latex(sp.limit(sp.sin(x)/x, x, 0)))
```

$$1$$

We can use 'limit' to check the result of derivation using the `diff` function:

```
print(sp.latex(f))
```

$$\sin(xy) + \cos(yz)$$

```
print(sp.latex(sp.diff(f, x)))
```

$$y\cos(xy)$$

```
h = sp.Symbol("h")
```

```
print(sp.latex(sp.limit((f.subs(x, x+h) - f)/h, h, 0)))
```

$$y\cos(xy)$$

We can change the direction from which we approach the limiting point using the `dir` keyword argument:

```
print(sp.latex(sp.limit(1/x, x, 0, dir="+")))
```

$$\infty$$

```
print(sp.latex(sp.limit(1/x, x, 0, dir="-")))
```

$$-\infty$$

5.4.5 Series

Series expansion is also one of the most useful features of a CAS. In SymPy we can perform a series expansion of an expression using the series function:

```
print(sp.latex(sp.series(sp.exp(x), x)))
```

$$1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + O\left(x^6\right)$$

By default it expands the expression around $x = 0$, but we can expand around any value of x by explicitly include a value in the function call:

```
print(sp.latex(sp.series(sp.exp(x), x, 1)))
```

$$e + e\left(x-1\right) + \frac{e\left(x-1\right)^{2}}{2} + \frac{e\left(x-1\right)^{3}}{6} + \frac{e\left(x-1\right)^{4}}{24} + \frac{e\left(x-1\right)^{5}}{120} + O\left(\left(x-1\right)^{6}; x \to 1\right)$$

And we can explicitly define to which order the series expansion should be carried out:

```
print(sp.latex(sp.series(sp.exp(x), x, 1, 5)))
```

$$e + e\left(x-1\right) + \frac{e\left(x-1\right)^{2}}{2} + \frac{e\left(x-1\right)^{3}}{6} + \frac{e\left(x-1\right)^{4}}{24} + O\left(\left(x-1\right)^{5}; x \to 1\right)$$

The series expansion includes the order of the approximation, which is very useful for keeping track of the order of validity when we do calculations with series expansions of different order:

```
s1 = sp.cos(x).series(x, 0, 5)
print(sp.latex(s1))
```

$$1 - \frac{x^{2}}{2} + \frac{x^{4}}{24} + O\left(x^{5}\right)$$

```
s2 = sp.sin(x).series(x, 0, 2)
print(sp.latex(s2))
```

$$x + O\left(x^{2}\right)$$

```
print(sp.latex(sp.expand(s1 * s2)))
```

$$x + O\left(x^{2}\right)$$

If we want to get rid of the order information we can use the removeO method:

```
print(sp.latex(sp.expand(s1.removeO() * s2.removeO())))
```

$$\frac{x^{5}}{24} - \frac{x^{3}}{2} + x$$

But note that this is not the correct expansion $\cos x \sin x$ of to 5th order:

```
(cos(x)*sin(x)).series(x, 0, 6)
```

5.5 Linear Algebra

Matrices are defined using the Matrix class:

```
m11, m12, m21, m22 = sp.symbols("m11, m12, m21, m22")
b1, b2 = sp.symbols("b1, b2")
```

```
A = sp.Matrix([[m11, m12],[m21, m22]])
print(sp.latex(A))
```

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}$$

```
b = sp.Matrix([[b1], [b2]])
print(sp.latex(b))
```

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

With `Matrix` class instances we can do the usual matrix algebra operations:

```
print(sp.latex(A**2))
```

$$\begin{bmatrix} m_{11}^2 + m_{12}m_{21} & m_{11}m_{12} + m_{12}m_{22} \\ m_{11}m_{21} + m_{21}m_{22} & m_{12}m_{21} + m_{22}^2 \end{bmatrix}$$

```
print(sp.latex(A * b))
```

$$\begin{bmatrix} b_1 m_{11} + b_2 m_{12} \\ b_1 m_{21} + b_2 m_{22} \end{bmatrix}$$

And calculate determinants and inverses, and the like:

```
print(sp.latex(A.det()))
```

$$m_{11}m_{22} - m_{12}m_{21}$$

```
print(sp.latex(A.inv()))
```

$$\begin{bmatrix} \frac{m_{22}}{m_{11}m_{22} - m_{12}m_{21}} & -\frac{m_{12}}{m_{11}m_{22} - m_{12}m_{21}} \\ -\frac{m_{21}}{m_{11}m_{22} - m_{12}m_{21}} & \frac{m_{11}}{m_{11}m_{22} - m_{12}m_{21}} \end{bmatrix}$$

5.6 Solving equations

For solving equations and systems of equations we can use the `solve` function:

```
print(sp.latex(sp.solve(x**2 - 1, x)))
```

$$[-1,\ 1]$$

```
print(sp.latex(sp.solve(x**4 - x**2 - 1, x)))
```

$$\left[-i\sqrt{-\frac{1}{2}+\frac{\sqrt{5}}{2}},\ i\sqrt{-\frac{1}{2}+\frac{\sqrt{5}}{2}},\ -\sqrt{\frac{1}{2}+\frac{\sqrt{5}}{2}},\ \sqrt{\frac{1}{2}+\frac{\sqrt{5}}{2}}\right]$$

System of equations:

```
print(sp.latex(sp.solve([x + y - 1, x - y - 1], [x,y])))
```

$$\{x : 1,\ y : 0\}$$

In terms of other symbolic expressions:

```
print(sp.latex(sp.solve([x + y - a, x - y - c], [x,y])))
```

$$\left\{x : \frac{a}{2}+\frac{c}{2},\ y : \frac{a}{2}-\frac{c}{2}\right\}$$

Exercise 5.1: Integration

Please solve the following integration operations:

$$\int_1^4 x \log x^2 \, dx$$

$$\int_0^3 \int_{x^2}^9 x^3 \exp y$$

$$\int_{-1}^1 \int_{-\sqrt{1-y^2}}^{\sqrt{1-y^2}} \int_{24x}^1 1 \, dz \, dy \, dx$$

Chapter 6

Matplotlib

Table of Contents

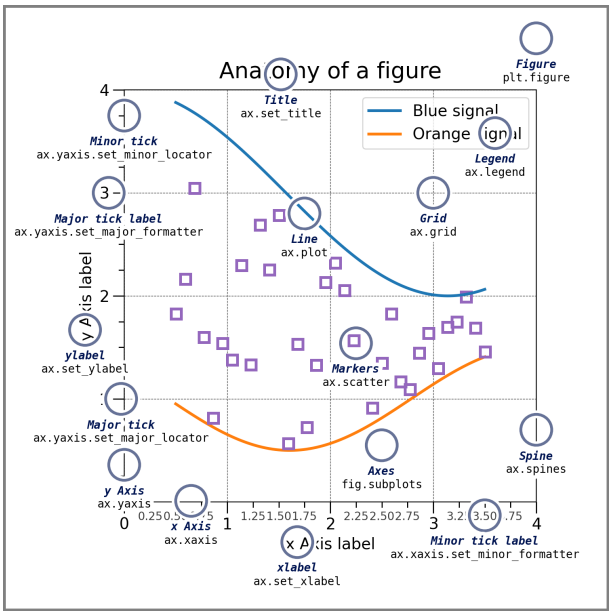
6.1 Introduction 139
6.2 Colormap and contour figures 158

6.1 Introduction

Matplotlib is an excellent 2D and 3D graphics library for generating scientific figures. Some of the many advantages of this library include:

- Easy to get started
- Support for LaTeX formatted labels and text
- Great control of every element in a figure, including figure size and DPI.
- High-quality output in many formats, including PNG, PDF, SVG, EPS, and PGF.
- GUI for interactively exploring figures and support for headless generation of figure files (useful for batch jobs)

One of the key features of matplotlib that I would like to emphasize, and that I think makes matplotlib highly suitable for generating figures for scientific publications is that all aspects of the figure can be controlled programmatically. This is important for reproducibility and convenient when one needs to regenerate the figure with updated data or change its appearance.



More information at the Matplotlib web page: `http://matplotlib.org/`

To get started using Matplotlib in a Python program, either include the symbols from the pylab module (the easy way):

C.R. 1

```python
import matplotlib
import matplotlib.pyplot as plt
```

We also need to import our numpy module to use later on.

C.R. 2

```python
import numpy as np
```

Let's start with a simple figure, but before we begin with generating some visuals, we need to create some data-points.

C.R. 3

```python
x = np.linspace(0, 5, 10)
y = x ** 2
```

Once we have the data of both (`x`, `y`) we can now create some plots.

The main idea with OOP is to have objects that one can apply functions and actions on, and no object or program states should be global.

The real advantage of this approach becomes apparent when more than one figure is created, or when a figure contains more than one subplot.

To use the object-oriented API we store our plot on a reference to the newly created `figure` instance in the `fig` variable, and from it we create a new axis instance axes using the `add_axes` method in the Figure class instance fig:

C.R. 4

```python
plt.style.use('bmh')
plt.figure()
plt.plot(x, y, 'r')
plt.xlabel('x')
plt.ylabel('y')
plt.title('title')
plt.savefig("images/Matplotlib/example-figure.pdf")
plt.close()
```

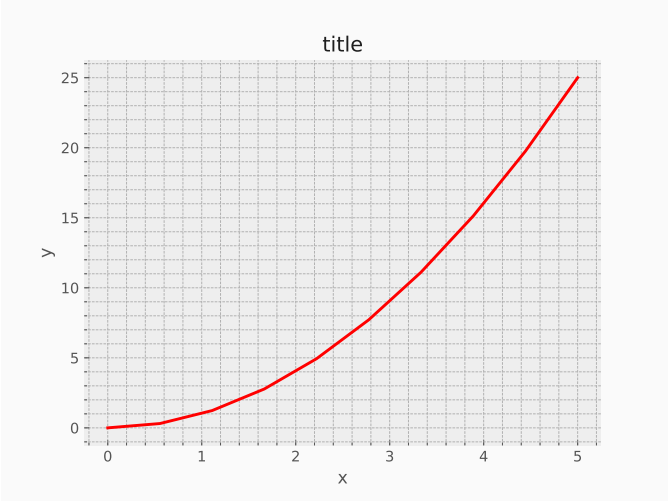


Figure 6.1

As can be seen, we have full control of where the plot axes are placed, and we can easily add more than one axis to the figure:

C.R. 5

```python
fig = plt.figure()

axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # main axes
axes2 = fig.add_axes([0.2, 0.5, 0.4, 0.3]) # inset axes

# main figure
axes1.plot(x, y, 'r')
axes1.set_xlabel('x')
axes1.set_ylabel('y')
axes1.set_title('Primary Plot')

# insert
axes2.plot(y, x, 'g')
axes2.set_xlabel('y')
axes2.set_ylabel('x')
axes2.set_title('Secondary Plot');

```

C.R. 6
python
```
# save figure and close
plt.savefig("images/Matplotlib/figure-in-a-figure.pdf")
plt.close()
```

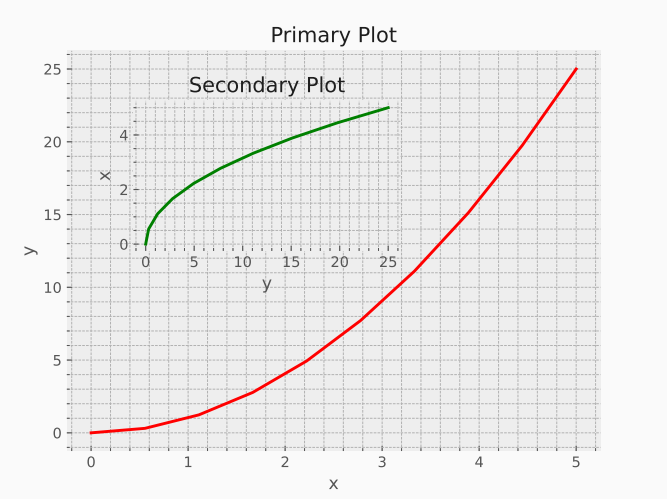


Figure 6.2

If we don't care about being explicit about where our plot axes are placed in the figure canvas, then we can use one of the many axis layout managers in matplotlib.

A good option is **subplots**, which can be used like this:

C.R. 7
python
```
fig, axes = plt.subplots()

axes.plot(x, y, 'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');

# save figure and close
plt.savefig("images/Matplotlib/a-simple-subplot-figure.pdf")
plt.close()
```

And because it is a subplot, we can add anothe subplot into the same frame, such as the following:

C.R. 8
python
```
fig, axes = plt.subplots(nrows=1, ncols=2)

for ax in axes:
    ax.plot(x, y, 'r')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('title')

# save figure and close
plt.savefig("images/Matplotlib/two-subplots-figure.pdf")
```

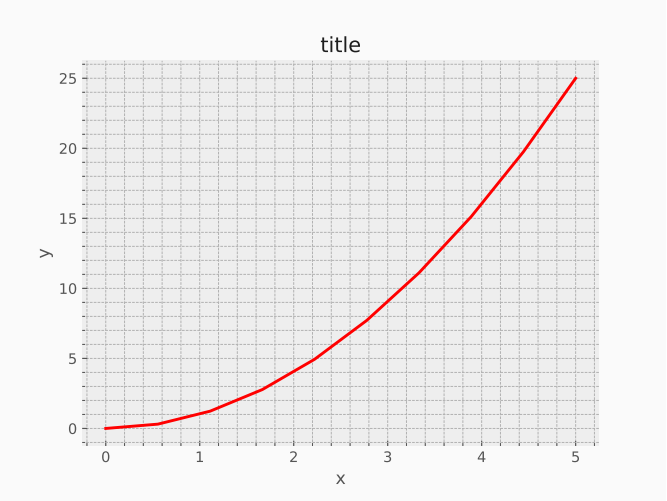


Figure 6.3

C.R. 9

```python
plt.close()
```

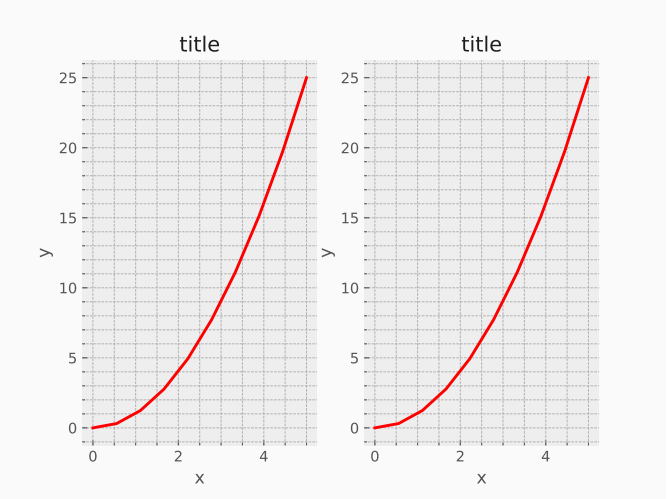


Figure 6.4

That was (relatively) easy, but it isn't so pretty with overlapping figure axes and labels, right?

We can deal with that by using the `fig.tight_layout` method, which automatically adjusts the positions of the axes on the figure canvas so that there is no overlapping content:

C.R. 10

```python
fig, axes = plt.subplots(nrows=1, ncols=2)

for ax in axes:
    ax.plot(x, y, 'r')
    ax.set_xlabel('x')
```

C.R. 11 python
```
    ax.set_ylabel('y')
    ax.set_title('title')

fig.tight_layout()

# save figure and close
plt.savefig("images/Matplotlib/tight-subplots-figure.pdf")
plt.close()
```

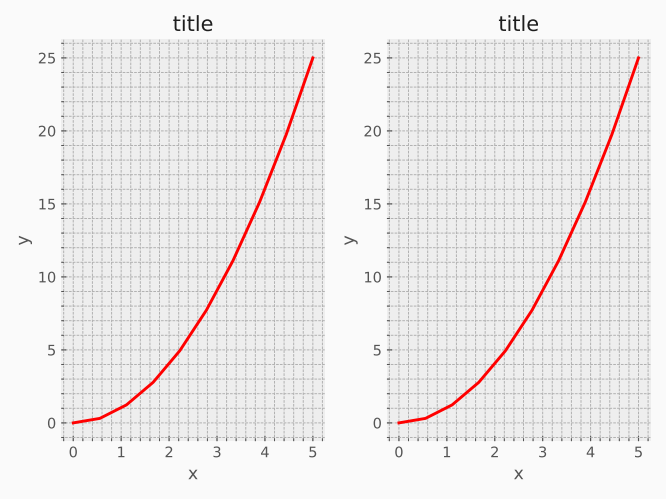


Figure 6.5

6.1.1 Figure size, aspect ratio and DPI

Matplotlib allows the aspect ratio, DPI and figure size to be specified when the Figure object is created, using the figsize and dpi keyword arguments. figsize is a tuple of the width and height of the figure in inches, and dpi is the dots-per-inch (pixel per inch). To create an 800x400 pixel, 100 dots-per-inch figure, we can do:

C.R. 12 python
```
fig = plt.figure(figsize=(8,4), dpi=100)
```

The same arguments can also be passed to layout managers, such as the subplots function:

C.R. 13 python
```
fig, axes = plt.subplots(figsize=(12,3))

axes.plot(x, y, 'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');

# save figure and close
plt.savefig("images/Matplotlib/long-figure.pdf")
```

```
10 plt.close()
```
C.R. 14 python

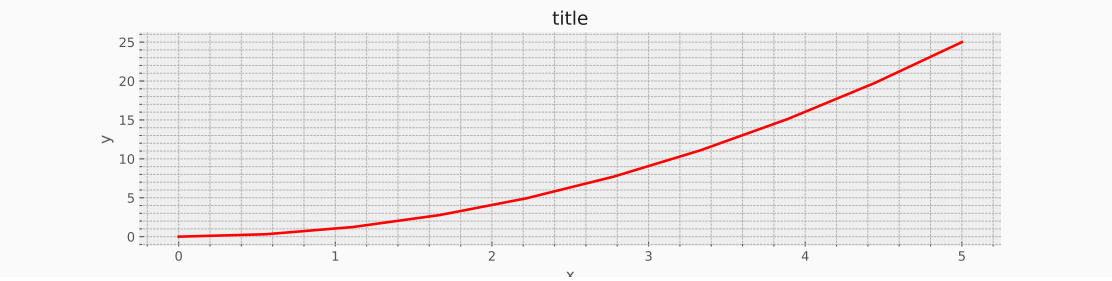


Figure 6.6

Saving Figures

As you have seen, the method `savefig` method is used quite frequently. This method allows you to save any figure generated by matplotlib. As it stands you can generate high-quality output in a number formats, including PNG, JPG, EPS, SVG, PGF and PDF. If you are using any non-vector image formats, you can also set the DPI settings, for example

```
1 fig.savefig("filename.png", dpi=200)
```
C.R. 15 python

6.1.2 Legends, labels and titles

Now that we have covered the basics of how to create a figure canvas and add axes instances to the canvas, let's look at how decorate a figure with titles, axis labels, and legends.

Figure Titles

A title can be added to each axis instance in a figure. To set the title, use the `set_title` method in the axes instance:

```
1 ax.set_title("title");
```
C.R. 16 python

Axis Labels

Similarly, with the methods `set_xlabel` and `set_ylabel`, we can set the labels of the X and Y axes:

```
1 ax.set_xlabel("x")
2 ax.set_ylabel("y")
```
C.R. 17 python

Legends

Legends for curves in a figure can be added in two (2) ways. One method is to use the legend method of the axis object and pass a list/tuple of legend texts for the previously defined curves:

C.R. 18 python
```
ax.legend(["curve1", "curve2", "curve3"])
```

[1] A software used by engineers for calculations

The method described above follows a similar way to MATLAB[1]. It is somewhat prone to errors and un-flexible if curves are added to or removed from the figure (resulting in a wrongly labelled curve).

A better method is to use the `label="label text"` keyword argument when plots or other objects are added to the figure, and then using the legend method without arguments to add the `legend` to the figure:

C.R. 19 python
```
ax.plot(x, x**2, label="curve1")
ax.plot(x, x**3, label="curve2")
ax.legend();
```

The advantage with this method is that if curves are added or removed from the figure, the legend is **automatically updated** accordingly.

The `legend` function takes an optional keyword argument `loc` that can be used to specify where in the figure the legend is to be drawn. The allowed values of `loc` are numerical codes for the various places the legend can be drawn. See `http://matplotlib.org/users/legend_guide.html#legend-location` for details. Some of the most common `loc` values are:

C.R. 20 python
```
ax.legend(loc=0) # let matplotlib decide the optimal location
ax.legend(loc=1) # upper right corner
ax.legend(loc=2) # upper left corner
ax.legend(loc=3) # lower left corner
ax.legend(loc=4) # lower right corner
# .. many more options are available
```

The following figure shows how to use the figure title, axis labels and legends described above:

C.R. 21 python
```
fig, ax = plt.subplots()

ax.plot(x, x**2, label="y = x**2")
ax.plot(x, x**3, label="y = x**3")
ax.legend(loc=2); # upper left corner
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('title')

```

C.R. 22
python
```
# save figure and close
plt.savefig("images/Matplotlib/legend-figure.pdf")
plt.close()
```

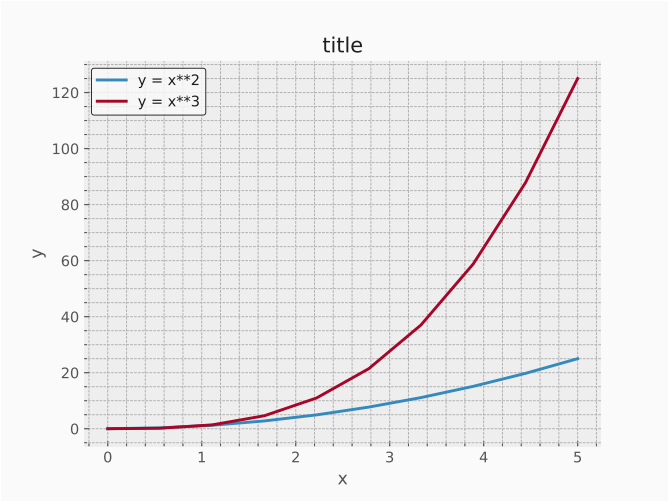


Figure 6.7

6.1.3 Formatting text: LaTeX, fontsize, font family

The figure above is functional, but it does not (yet) satisfy the criteria for a figure used in a publication.

- we need to have LaTeX formatted text,
- we need to be able to adjust the font size to appear right in a publication.

Matplotlib has great support for LaTeX. All we need to do is to use dollar signs encapsulate LaTeX in any text (legend, title, label, etc.).

For example, "$y = x^3$".

But here we can run into a slightly subtle problem with LaTeX code and Python text strings. In LaTeX, we frequently use the backslash in commands, for example `\alpha` to produce the symbol α. But the backslash already has a meaning in Python strings (the escape code character). To avoid Python messing up our latex code, we need to use "raw" text strings. Raw text strings are prepended with an 'r', like `r"\alpha"` or `r'\alpha'` instead of `"\alpha"` or `'\alpha'`:

C.R. 23
python
```
fig, ax = plt.subplots()

```

C.R. 24 python
```python
ax.plot(x, x**2, label=r"$y = \alpha^2$")
ax.plot(x, x**3, label=r"$y = \alpha^3$")
ax.legend(loc=2) # upper left corner
ax.set_xlabel(r'$\alpha$', fontsize=18)
ax.set_ylabel(r'$y$', fontsize=18)
ax.set_title('title')

# save figure and close
plt.savefig("images/Matplotlib/latex-figure.pdf")
plt.close()
```

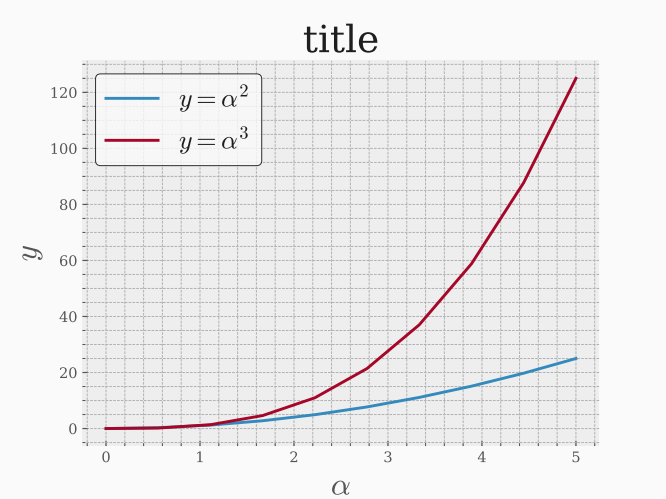


Figure 6.8

We can also change the global font size and font family, which applies to all text elements in a figure (tick labels, axis labels and titles, legends, etc.):

C.R. 25 python
```python
matplotlib.rcParams.update({'font.size': 18, 'font.family': 'serif'})
```

Or, alternatively, we can request that matplotlib uses LaTeX to render the text elements in the figure:

```
matplotlib.rcParams.update({'font.size': 18, 'text.usetex': True})
```

```
fig, ax = plt.subplots()

ax.plot(x, x**2, label=r"$y = \alpha^2$")
ax.plot(x, x**3, label=r"$y = \alpha^3$")
ax.legend(loc=2) # upper left corner
ax.set_xlabel(r'$\alpha$')
ax.set_ylabel(r'$y$')
ax.set_title('title');
```

```
matplotlib.rcParams.update({'font.size': 12, 'font.family': 'sans', 'text.usetex':
↪  False})
```

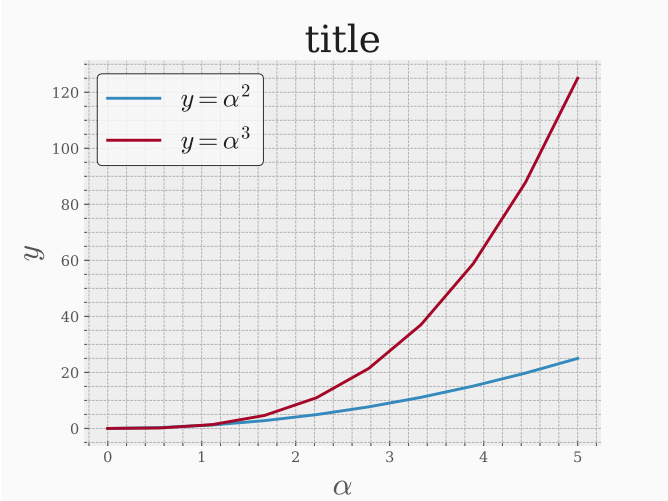


Figure 6.9

6.1.4 Setting colors, linewidths, linetypes

Colours

With matplotlib, we can define the colors of lines and other graphical elements in a number of ways. First of all, we can use the MATLAB-like syntax where 'b' means blue, 'g' means green, etc. The MATLAB API for selecting line styles are also supported: where, for example, 'b.-' means a blue line with dots:

```
# MATLAB style line color and style
ax.plot(x, x**2, 'b.-') # blue line with dots
ax.plot(x, x**3, 'g--') # green dashed line
```

We can also define colors by their names or RGB hex codes and optionally provide an alpha value using the color and alpha keyword arguments:

```
fig, ax = plt.subplots()

ax.plot(x, x+1, color="red", alpha=0.5) # half-transparent red
ax.plot(x, x+2, color="#1155dd")        # RGB hex code for a bluish color
ax.plot(x, x+3, color="#15cc55")        # RGB hex code for a greenish color
```

Line and marker styles

To change the line width, we can use the linewidth or lw keyword argument. The line style can be selected using the linestyle or ls keyword arguments:

```
fig, ax = plt.subplots(figsize=(12,6))
```

```
ax.plot(x, x+1, color="blue", linewidth=0.25)
ax.plot(x, x+2, color="blue", linewidth=0.50)
ax.plot(x, x+3, color="blue", linewidth=1.00)
ax.plot(x, x+4, color="blue", linewidth=2.00)

# possible linestype options '-', '--', '-.', ':', 'steps'
ax.plot(x, x+5, color="red", lw=2, linestyle='-')
ax.plot(x, x+6, color="red", lw=2, ls='-.')
ax.plot(x, x+7, color="red", lw=2, ls=':')

# custom dash
line, = ax.plot(x, x+8, color="black", lw=1.50)
line.set_dashes([5, 10, 15, 10]) # format: line length, space length, ...

# possible marker symbols: marker = '+', 'o', '*', 's', ',', '.', '1', '2', '3', '4',
↪  ...
ax.plot(x, x+ 9, color="green", lw=2, ls='--', marker='+')
ax.plot(x, x+10, color="green", lw=2, ls='--', marker='o')
ax.plot(x, x+11, color="green", lw=2, ls='--', marker='s')
ax.plot(x, x+12, color="green", lw=2, ls='--', marker='1')

# marker size and color
ax.plot(x, x+13, color="purple", lw=1, ls='-', marker='o', markersize=2)
ax.plot(x, x+14, color="purple", lw=1, ls='-', marker='o', markersize=4)
ax.plot(x, x+15, color="purple", lw=1, ls='-', marker='o', markersize=8,
↪  markerfacecolor="red")
ax.plot(x, x+16, color="purple", lw=1, ls='-', marker='s', markersize=8,
        markerfacecolor="yellow", markeredgewidth=2, markeredgecolor="blue");
```

6.1.5 Control over axis appearance

The appearance of the axes is an important aspect of a figure that we often need to modify to make a publication quality graphics. We need to be able to control where the ticks and labels are placed, modify the font size and possibly the labels used on the axes. In this section we will look at controlling those properties in a matplotlib figure.

Plot range

The first thing we might want to configure is the ranges of the axes. We can do this using the set_{ylim} and set_{xlim} methods in the axis object, or axis('tight') for automatically getting "tightly fitted" axes ranges:

```
fig, axes = plt.subplots(1, 3, figsize=(12, 4))
```

```
axes[0].plot(x, x**2, x, x**3)
axes[0].set_title("default axes ranges")

axes[1].plot(x, x**2, x, x**3)
axes[1].axis('tight')
axes[1].set_title("tight axes")

axes[2].plot(x, x**2, x, x**3)
axes[2].set_ylim([0, 60])
axes[2].set_xlim([2, 5])
axes[2].set_title("custom axes range");
```

Logarithmic scale

It is also possible to set a logarithmic scale for one or both axes. This functionality is in fact only one application of a more general transformation system in Matplotlib. Each of the axes' scales are set seperately using $\text{set}_{\text{xscale}}$ and $\text{set}_{\text{yscale}}$ methods which accept one parameter (with the value "log" in this case):

```
fig, axes = plt.subplots(1, 2, figsize=(10,4))

axes[0].plot(x, x**2, x, np.exp(x))
axes[0].set_title("Normal scale")

axes[1].plot(x, x**2, x, np.exp(x))
axes[1].set_yscale("log")
axes[1].set_title("Logarithmic scale (y)");
```

6.1.6 Placement of ticks and custom tick labels

We can explicitly determine where we want the axis ticks with $\text{set}_{\text{xticks}}$ and $\text{set}_{\text{yticks}}$, which both take a list of values for where on the axis the ticks are to be placed. We can also use the $\text{set}_{\text{xticklabels}}$ and $\text{set}_{\text{yticklabels}}$ methods to provide a list of custom text labels for each tick location:

```
fig, ax = plt.subplots(figsize=(10, 4))

ax.plot(x, x**2, x, x**3, lw=2)

ax.set_xticks([1, 2, 3, 4, 5])
ax.set_xticklabels([r'$\alpha$', r'$\beta$', r'$\gamma$', r'$\delta$', r'$\epsilon$'],
↪  fontsize=18)

yticks = [0, 50, 100, 150]
```

```
ax.set_yticks(yticks)
ax.set_yticklabels(["$%.1f$" % y for y in yticks], fontsize=18); # use LaTeX formatted
↪  labels
```

There are a number of more advanced methods for controlling major and minor tick placement in matplotlib figures, such as automatic placement according to different policies. See `http://matplotlib.org/api/ticker_api.html` for details.

Scientific notation

With large numbers on axes, it is often better use scientific notation:

```
fig, ax = plt.subplots(1, 1)

ax.plot(x, x**2, x, np.exp(x))
ax.set_title("scientific notation")

ax.set_yticks([0, 50, 100, 150])

from matplotlib import ticker
formatter = ticker.ScalarFormatter(useMathText=True)
formatter.set_scientific(True)
formatter.set_powerlimits((-1,1))
ax.yaxis.set_major_formatter(formatter)
```

6.1.7 Axis number and axis label spacing

```
# distance between x and y axis and the numbers on the axes
matplotlib.rcParams['xtick.major.pad'] = 5
matplotlib.rcParams['ytick.major.pad'] = 5

fig, ax = plt.subplots(1, 1)

ax.plot(x, x**2, x, np.exp(x))
ax.set_yticks([0, 50, 100, 150])

ax.set_title("label and axis spacing")

# padding between axis label and axis numbers
ax.xaxis.labelpad = 5
ax.yaxis.labelpad = 5

ax.set_xlabel("x")
ax.set_ylabel("y");
```

```
# restore defaults
matplotlib.rcParams['xtick.major.pad'] = 3
matplotlib.rcParams['ytick.major.pad'] = 3
```

Axis Position Adjustement

Unfortunately, when saving figures the labels are sometimes clipped, and it can be necessary to adjust the positions of axes a little bit. This can be done using $\text{subplots}_{\text{adjust}}$:

```
fig, ax = plt.subplots(1, 1)

ax.plot(x, x**2, x, np.exp(x))
ax.set_yticks([0, 50, 100, 150])

ax.set_title("title")
ax.set_xlabel("x")
ax.set_ylabel("y")

fig.subplots_adjust(left=0.15, right=.9, bottom=0.1, top=0.9);
```

6.1.8 Axis grid

With the grid method in the axis object, we can turn on and off grid lines. We can also customize the appearance of the grid lines using the same keyword arguments as the plot function:

```
fig, axes = plt.subplots(1, 2, figsize=(10,3))

# default grid appearance
axes[0].plot(x, x**2, x, x**3, lw=2)
axes[0].grid(True)

# custom grid appearance
axes[1].plot(x, x**2, x, x**3, lw=2)
axes[1].grid(color='b', alpha=0.5, linestyle='dashed', linewidth=0.5)
```

6.1.9 Axis Spines

We can also change the properties of axis spines:

```
fig, ax = plt.subplots(figsize=(6,2))

ax.spines['bottom'].set_color('blue')
```

```
ax.spines['top'].set_color('blue')

ax.spines['left'].set_color('red')
ax.spines['left'].set_linewidth(2)

# turn off axis spine to the right
ax.spines['right'].set_color("none")
ax.yaxis.tick_left() # only ticks on the left side
```

6.1.10 Twin Axes

Sometimes it is useful to have dual x or y axes in a figure; for example, when plotting curves with different units together. Matplotlib supports this with the twinx and twiny functions:

```
fig, ax1 = plt.subplots()

ax1.plot(x, x**2, lw=2, color="blue")
ax1.set_ylabel(r"area $(m^2)$", fontsize=18, color="blue")
for label in ax1.get_yticklabels():
    label.set_color("blue")

ax2 = ax1.twinx()
ax2.plot(x, x**3, lw=2, color="red")
ax2.set_ylabel(r"volume $(m^3)$", fontsize=18, color="red")
for label in ax2.get_yticklabels():
    label.set_color("red")
```

6.1.11 Axes where x and y is zero

```
fig, ax = plt.subplots()

ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')

ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0)) # set position of x spine to x=0

ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))   # set position of y spine to y=0

xx = np.linspace(-0.75, 1., 100)
ax.plot(xx, xx**3);
```

6.1.12 Other 2D plot styles

In addition to the regular plot method, there are a number of other functions for generating different kind of plots. See the matplotlib plot gallery for a complete list of available plot types: `http://matplotlib.org/gallery.html`. Some of the more useful ones are show below:

```
n = np.array([0,1,2,3,4,5])
```

```
fig, axes = plt.subplots(1, 4, figsize=(12,3))

axes[0].scatter(xx, xx + 0.25*np.random.randn(len(xx)))
axes[0].set_title("scatter")

axes[1].step(n, n**2, lw=2)
axes[1].set_title("step")

axes[2].bar(n, n**2, align="center", width=0.5, alpha=0.5)
axes[2].set_title("bar")

axes[3].fill_between(x, x**2, x**3, color="green", alpha=0.5);
axes[3].set_title("fill_between");
```

```
# polar plot using add_axes and polar projection
fig = plt.figure()
ax = fig.add_axes([0.0, 0.0, .6, .6], polar=True)
t = np.linspace(0, 2 * np.pi, 100)
ax.plot(t, t, color='blue', lw=3);
```

```
# A histogram
n = np.random.randn(100000)
fig, axes = plt.subplots(1, 2, figsize=(12,4))

axes[0].hist(n)
axes[0].set_title("Default histogram")
axes[0].set_xlim((min(n), max(n)))

axes[1].hist(n, cumulative=True, bins=50)
axes[1].set_title("Cumulative detailed histogram")
axes[1].set_xlim((min(n), max(n)));
```

6.1.13 Text annotation

Annotating text in matplotlib figures can be done using the text function. It supports LaTeX formatting just like axis label texts and titles:

```
fig, ax = plt.subplots()
```

```
ax.plot(xx, xx**2, xx, xx**3)

ax.text(0.15, 0.2, r"$y=x^2$", fontsize=20, color="blue")
ax.text(0.65, 0.1, r"$y=x^3$", fontsize=20, color="green");
```

6.1.14 Figures with multiple subplots and insets

Axes can be added to a matplotlib Figure canvas manually using fig.add$_{axes}$ or using a sub-figure layout manager such as subplots, subplot2grid, or gridspec:

subplots

```
fig, ax = plt.subplots(2, 3)
fig.tight_layout()
```

subplot2grid

```
fig = plt.figure()
ax1 = plt.subplot2grid((3,3), (0,0), colspan=3)
ax2 = plt.subplot2grid((3,3), (1,0), colspan=2)
ax3 = plt.subplot2grid((3,3), (1,2), rowspan=2)
ax4 = plt.subplot2grid((3,3), (2,0))
ax5 = plt.subplot2grid((3,3), (2,1))
fig.tight_layout()
```

gridspec

```
import matplotlib.gridspec as gridspec
```

```
fig = plt.figure()

gs = gridspec.GridSpec(2, 3, height_ratios=[2,1], width_ratios=[1,2,1])
for g in gs:
    ax = fig.add_subplot(g)

fig.tight_layout()
```

`add_axes`

Manually adding axes with add_{axes} is useful for adding insets to figures:

```
fig, ax = plt.subplots()

ax.plot(xx, xx**2, xx, xx**3)
fig.tight_layout()

# inset
inset_ax = fig.add_axes([0.2, 0.55, 0.35, 0.35]) # X, Y, width, height

inset_ax.plot(xx, xx**2, xx, xx**3)
inset_ax.set_title('zoom near origin')

# set axis range
inset_ax.set_xlim(-.2, .2)
inset_ax.set_ylim(-.005, .01)

# set axis tick locations
inset_ax.set_yticks([0, 0.005, 0.01])
inset_ax.set_xticks([-0.1,0,.1]);
```

6.2 Colormap and contour figures

Colormaps and contour figures are useful for plotting functions of two (2) variables. In most of these functions we will use a colormap to encode one dimension of the data. There are a number of predefined colormaps. It is relatively straightforward to define custom colormaps.

For a list of pre-defined colormaps, see: `http://www.scipy.org/Cookbook/Matplotlib/Show_colormaps`

C.R. 26 python
```
alpha = 0.7
phi_ext = 2 * np.pi * 0.5

def flux_qubit_potential(phi_m, phi_p):
    return 2 + alpha - 2 * np.cos(phi_p) * np.cos(phi_m) \
        - alpha * np.cos(phi_ext - 2*phi_p)
```

C.R. 27 python
```
phi_m = np.linspace(0, 2*np.pi, 100)
phi_p = np.linspace(0, 2*np.pi, 100)
X,Y = np.meshgrid(phi_p, phi_m)
Z = flux_qubit_potential(X, Y).T
```

pcolor

C.R. 28 python
```
fig, ax = plt.subplots()

p = ax.pcolor(X/(2*np.pi), Y/(2*np.pi), Z, cmap=matplotlib.cm.RdBu, vmin=abs(Z).min(),
 ↪  vmax=abs(Z).max())
cb = fig.colorbar(p, ax=ax)

# save figure and close
plt.savefig("images/Matplotlib/pcolor-plot.pdf")
plt.close()
```

imshow

C.R. 29 python
```
fig, ax = plt.subplots()

im = ax.imshow(Z, cmap=matplotlib.cm.RdBu, vmin=abs(Z).min(), vmax=abs(Z).max(), extent=[0, 1,
 ↪  0, 1])
im.set_interpolation('bilinear')

cb = fig.colorbar(im, ax=ax)

```

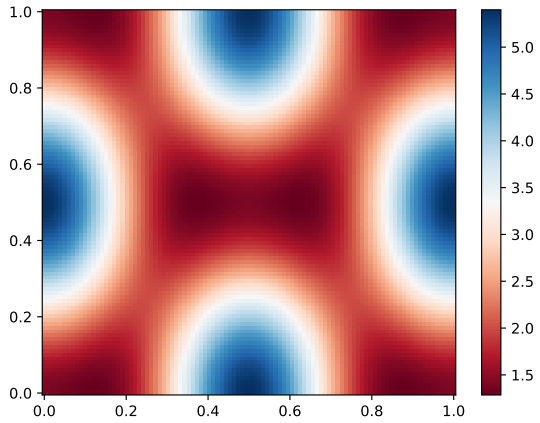


Figure 6.10

C.R. 30

```python
# save figure and close
plt.savefig("images/Matplotlib/imshow-plot.pdf")
plt.close()
```

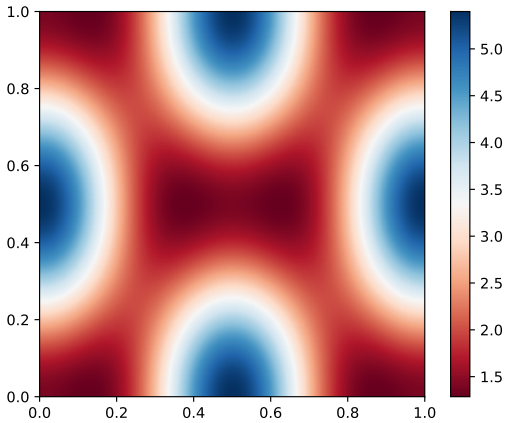


Figure 6.11

contour

C.R. 31

```python
fig, ax = plt.subplots()

cnt = ax.contour(Z, cmap=matplotlib.cm.RdBu, vmin=abs(Z).min(), vmax=abs(Z).max(), extent=[0, 1, 0, 1])

# save figure and close
```

C.R. 32
```python
plt.savefig("images/Matplotlib/contour-plot.pdf")
plt.close()
```

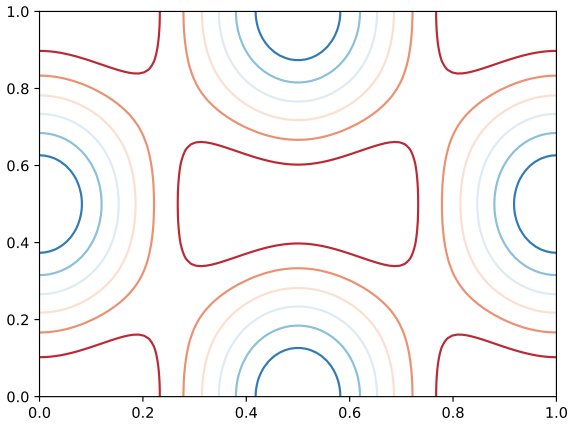


Figure 6.12

6.2.1 3D Figures

To use 3D graphics in matplotlib, we first need to create an instance of the Axes3D class. 3D axes can be added to a matplotlib figure canvas in exactly the same way as 2D axes; or, more conveniently, by passing a `projection='3d'` keyword argument to the `add_axes` or `add_subplot` methods.

C.R. 33
```python
from mpl_toolkits.mplot3d.axes3d import Axes3D
```

Surface Plots

C.R. 34
```python
fig = plt.figure(figsize=(14,6))

# `ax` is a 3D-aware axis instance because of the projection='3d' keyword argument to
↪ add_subplot
ax = fig.add_subplot(1, 2, 1, projection='3d')

p = ax.plot_surface(X, Y, Z, rstride=4, cstride=4, linewidth=0)

# surface_plot with color grading and color bar
ax = fig.add_subplot(1, 2, 2, projection='3d')
p = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=matplotlib.cm.coolwarm, linewidth=0,
↪ antialiased=False)
cb = fig.colorbar(p, shrink=0.5)
```

C.R. 35

```python

# save figure and close
plt.savefig("images/Matplotlib/3d-surface-plot.pdf")
plt.close()
```

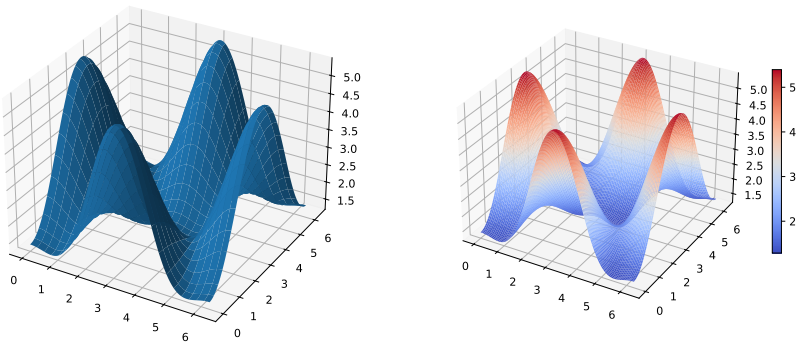


Figure 6.13

Wire-Frame Plots

C.R. 36

```python
fig = plt.figure(figsize=(8,6))

ax = fig.add_subplot(1, 1, 1, projection='3d')

p = ax.plot_wireframe(X, Y, Z, rstride=4, cstride=4)

# save figure and close
plt.savefig("images/Matplotlib/3d-wire-plot.pdf")
plt.close()
```

Contour Plots with Projections

C.R. 37

```python
fig = plt.figure(figsize=(8,6))

ax = fig.add_subplot(1,1,1, projection='3d')

ax.plot_surface(X, Y, Z, rstride=4, cstride=4, alpha=0.25)
cset = ax.contour(X, Y, Z, zdir='z', offset=-np.pi, cmap=matplotlib.cm.coolwarm)
cset = ax.contour(X, Y, Z, zdir='x', offset=-np.pi, cmap=matplotlib.cm.coolwarm)
cset = ax.contour(X, Y, Z, zdir='y', offset=3*np.pi, cmap=matplotlib.cm.coolwarm)

ax.set_xlim3d(-np.pi, 2*np.pi);
```

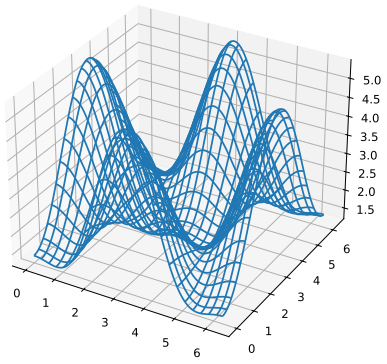


Figure 6.14

C.R. 38

```python
ax.set_ylim3d(0, 3*np.pi);
ax.set_zlim3d(-np.pi, 2*np.pi);

# save figure and close
plt.savefig("images/Matplotlib/contour-plot-with-projections.pdf")
plt.close()
```

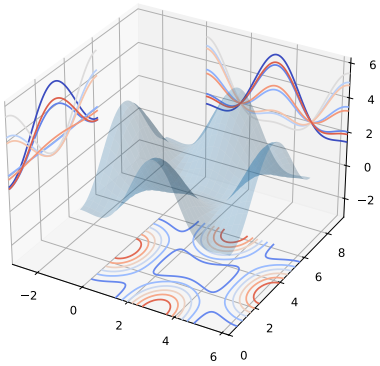


Figure 6.15

Change the view angle

We can change 3D plot perspective using the method `view_init`, which takes two (2) arguments:

- elevation angle,

- azimuth angle.

> Both these values are in degrees and not radians.

C.R. 39

```python
fig = plt.figure(figsize=(12,6))

ax = fig.add_subplot(1,2,1, projection='3d')
ax.plot_surface(X, Y, Z, rstride=4, cstride=4, alpha=0.25)
ax.view_init(30, 45)

ax = fig.add_subplot(1,2,2, projection='3d')
ax.plot_surface(X, Y, Z, rstride=4, cstride=4, alpha=0.25)
ax.view_init(70, 30)

fig.tight_layout()


# save figure and close
plt.savefig("images/Matplotlib/view-angle.pdf")
plt.close()
```

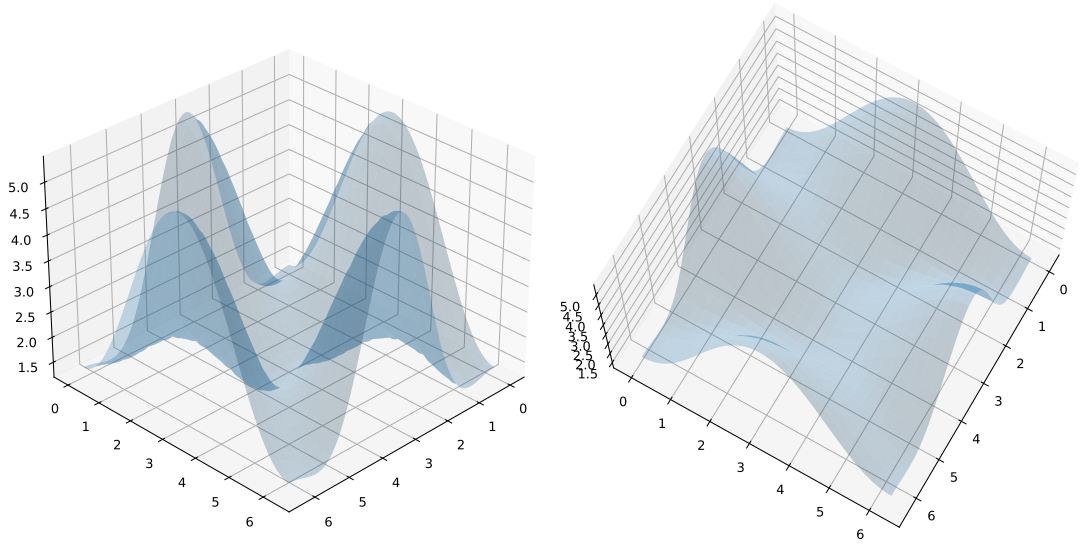


Figure 6.16

Chapter 7

pandas

Table of Contents

7.1 Introduction 165
7.2 Series 167
7.3 DataFrames 171
7.4 Pandas for Panel Data 177
7.5 Application: Long-Run Growth 180

7.1 Introduction

The pandas DataFrame is a structure that contains two-dimensional data and its corresponding labels. DataFrames are widely used in data science, machine learning, scientific computing, and many other data-intensive fields.

DataFrames are similar to SQL [1] tables or the spreadsheets that you work with in Microsoft Excel or OpenOffice Calc. In many cases, DataFrames are faster, easier to use, and more powerful than tables or spreadsheets as they're an integral part of the Python and NumPy ecosystems.

[1] a domain-specific language used to manage data, especially in a relational database management system (RDBMS)

Things that can be done with pandas include:

- Defines fundamental structures for working with data,
- Access methods that facilitate operations,
- Reading in data,
- Adjusting indices,
- Working with dates and time series,
- Sorting, grouping, re-ordering and general data manipulation,
- Sealing with missing values, etc., etc.

More sophisticated statistical functionality is left to other packages, such as statsmodels and sklearn, which are built on top of pandas.

Figure 7.1: A spreadsheet is a computer application for computation, organization, analysis and storage of data in tabular form and plays a vital role in many organisation be it engineering or economics.

Throughout the lecture, it will be assumed that the following imports have taken place and have a reasonable grasp on these packages.

C.R. 1
python
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import requests
```

> Requests is a simple yet powerful python library to work with HTTP requests.

There are two (2) important data types defined by pandas:

Series Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the index.

DataFrame a two-dimensional, tabular data structure with rows and columns. It is similar to a spreadsheet or a table in a relational database. The DataFrame has three main components: the data, which is stored in rows and columns; the rows, which are labeled by an index; and the columns, which are labeled and contain the actual data.

Let's start working with series.

7.2 Series

Series, as mentioned previously, is a one-dimensional array-like object that can hold data of **any type** (integer, float, string, etc.). It is labelled, meaning each element has a unique identifier called an index.

> Think of a Series as a column in a spreadsheet or a single column of a database table.

Series are a fundamental data structure in Pandas and are employed for data manipulation and analysis tasks. They can be created from lists, arrays, dictionaries, and existing Series objects. Series are also a building block for the more complex Pandas DataFrame, which is a two-dimensional table-like structure consisting of multiple Series objects, which we will be beginning shortly.

We begin by creating a series of four (4) random observations:

C.R. 2

```python
s = pd.Series(np.random.randn(4), name='daily returns')
print(s)
```

In our results we can see the new indices, our random values generated by the numpy random function. In addition, information on the series name (`daily returns`) and its type (`float64`) is also given.

```text
0   -0.775075
1    2.113545
2    0.256437
3    0.093216
Name: daily returns, dtype: float64
```

To simplify our understanding, we can imagine the indices `0`, `1`, `2`, `3` as indexing four (4) listed companies, and the values being daily returns on their shares.

Exercise 7.1: Series - An Introduction

Write a Pandas program to create and display a one-dimensional array-like object containing an array of data.

Solution

C.R. 3

```python
import pandas as pd
ds = pd.Series([2, 4, 6, 8, 10])
print(ds)
```

```text
0     2
1     4
2     6
3     8
4    10
dtype: int64
```

Exercise 7.2: Sorting a Series

Write a Pandas program to sort a given Series.

text

```
#+begin_example
Original data series:
0       100
1       200
2    python
3    300.12
4       400
dtype: object
0       100
1       200
3    300.12
4       400
```

Solution

```
2    python
```

C.R. 4

python

C.R. 5

```
import pandas as pd
ds = pd.Series([2, 4, 6, 8, 10])
```

C.R. 6

python

```
print(ds)
```

text

```
Original data series:
0       100
1       200
2    python
3    300.12
4       400
dtype: object
0       100
1       200
3    300.12
4       400
```

```
2    python
```

As with any respectable spreadsheet software, pandas has a wide variety of operations thanks to being built on top of numpy. Let's do a simple multiplication of 100 of each element in the series we have defined previously:

C.R. 7

```
print(s * 100)
```

python

As can be seen the operation is easily implemented to each individual value in the series.

text

```
0    -77.507464
1    211.354529
2     25.643659
3      9.321635
Name: daily returns, dtype: float64
```

We can take the absolute value of all values using `np.abs()`:

Absolute Value

The absolute value of a real number x, denoted $\boldsymbol{p}x$, is the non-negative value of x without regard to its sign. Namely, $\boldsymbol{p}x = x$ if x is a positive number, and $\boldsymbol{p}x = -x$ if x is negative.

C.R. 8
```python
print(np.abs(s))
```

```text
0    0.775075
1    2.113545
2    0.256437
3    0.093216
Name: daily returns, dtype: float64
```

As a spreadsheet software, Series provide more than numpy arrays. Not only do they have some additional methods but also functions describing statistical information. For example here we use `describe()` to view some basic statistical details like percentile, mean, std, etc. of numeric values.

Some Statistical Informations

A **mean** is a numeric quantity representing the "center" of a collection of numbers and is intermediate to the extreme values of the set of numbers.

The **standard deviation** is a measure of the amount of variation of the values of a variable about its mean.

C.R. 9
```python
print(s.describe())
```

```text
count    4.000000
mean     0.422031
std      1.215157
min     -0.775075
25%     -0.123856
50%      0.174826
75%      0.720714
max      2.113545
Name: daily returns, dtype: float64
```

In an pandas series, indices are more flexible than it looks at first glance. Here we can insert new values to its indices. For example let's make this look a bit more like a stock information:

C.R. 10
```python
s.index = ['AMZN', 'AAPL', 'MSFT', 'GOOG']
print(s)
```

```text
AMZN   -0.775075
AAPL    2.113545
MSFT    0.256437
```

```
GOOG    0.093216
Name: daily returns, dtype: float64
```

[2] with the restriction that the items in the dictionary all have the same type, which for our example, floats.

It is not a hard strecth to think of Series as a fast, efficient Python dictionaries [2]. In fact, you can use much of the same syntax as Python dictionaries. So for example, we can access a value by writing its indices:

C.R. 11
```python
print(s['AMZN'])
```

```text
-0.7750746403810563
```

Or we can change the definition of a value by accessing it via its key value.

C.R. 12
```python
s['AMZN'] = 0
print(s)
```

```text
AMZN    0.000000
AAPL    2.113545
MSFT    0.256437
GOOG    0.093216
Name: daily returns, dtype: float64
```

Or we can check if a certain key is in a Series.

C.R. 13
```python
print('AAPL' in s)
```

```text
True
```

7.3 DataFrames

A DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three (3) principal components:

- data,
- rows,
- and columns,

> In essence, a DataFrame in pandas is analogous to a Excel spreadsheet.

Therefore, it is a powerful tool for representing and analysing data that are naturally organized into rows and columns, often with **descriptive indexes** for individual rows and individual columns.

With all that information covered, let's look at an example that reads data from the CSV file `pandas/data/test/pwt.csv`, which is taken from the Penn World Tables.

> A Comma-separated values (CSV) is a text file format that uses commas to separate values, and newlines to separate records.

The dataset contains the following indicators:

Variable Name	Description
POP	Population (in thousands)
XRAT	Exchange Rate to US Dollar
tcgdp	Total PPP Converted GDP (in million international dollar)
cc	Consumption Share of PPP Converted GDP Per Capita (%)
cg	Government Consumption Share of PPP Converted GDP Per Capita (%)

We'll read this in from a URL using the pandas function `.read_csv()`.

C.R. 14

```python
df = pd.read_csv("https://raw.githubusercontent.com/QuantEcon/" + \
                 "lecture-python-programming/master/source/" + \
                 "_static/lecture_specific/pandas/data/test_pwt.csv")
print(type(df))
```

and as usual we can check the data of the dataFrame and see it is a different class compared to build in classes.

```text
<class 'pandas.core.frame.DataFrame'>
```

Here's the content of `test_pwt.csv`, which we can easily see by printing the dataframe.

C.R. 15
```python
print(df)
```

```text
        country country isocode  year          POP       XRAT         tcgdp         cc
        ↪  cg
0     Argentina             ARG  2000    37335.653   0.999500  2.950722e+05  75.716805
↪  5.578804
1     Australia             AUS  2000    19053.186   1.724830  5.418047e+05  67.759026
↪  6.720098
2         India             IND  2000  1006300.297  44.941600  1.728144e+06  64.575551
↪  14.072206
3        Israel             ISR  2000     6114.570   4.077330  1.292539e+05  64.436451
↪  10.266688
4        Malawi             MWI  2000    11801.505  59.543808  5.026222e+03  74.707624
↪  11.658954
5  South Africa             ZAF  2000    45064.098   6.939830  2.272424e+05  72.718710
↪  5.726546
6  United States             USA  2000   282171.957   1.000000  9.898700e+06  72.347054
↪  6.032454
7       Uruguay             URY  2000     3219.793  12.099592  2.525596e+04  78.978740
↪  5.108068
```

Exercise 7.3: DataFrame - An Introduction

Write a Pandas program to create a dataframe from a dictionary and display it.
Sample data:

```
{'X':[78,85,96,80,86], 'Y':[84,94,89,83,86],'Z':[86,97,96,72,83]}
```

7.3.1 Selecting Data by Position

In practice, one thing that we do all the time is to find, select and work with a subset of the data of our interests. We can select particular rows using standard Python array slicing notation. We can think of these as working on an Excel sheet with ranges similar to A2:B5.

C.R. 16
```python
print(df[2:5])
```

Here we can see the result showing a range of the dataframe.

```text
  country country isocode  year          POP       XRAT         tcgdp         cc         cg
2   India             IND  2000  1006300.297  44.941600  1.728144e+06  64.575551  14.072206
3  Israel             ISR  2000     6114.570   4.077330  1.292539e+05  64.436451  10.266688
4  Malawi             MWI  2000    11801.505  59.543808  5.026222e+03  74.707624  11.658954
```

To select columns, we can pass a list containing the names of the desired columns represented as strings. For now we will look at the two (2) columns.

C.R. 17
```python
print(df[['country', 'tcgdp']])
```

```text
         country         tcgdp
0      Argentina  2.950722e+05
1      Australia  5.418047e+05
2          India  1.728144e+06
3         Israel  1.292539e+05
4         Malawi  5.026222e+03
5   South Africa  2.272424e+05
6  United States  9.898700e+06
7        Uruguay  2.525596e+04
```

To select both rows and columns using integers, the `iloc` attribute should be used with the format `.iloc[rows, columns]`, which you can see below.

C.R. 18
```python
print(df.iloc[2:5, 0:4])
```

```text
  country country isocode  year          POP
2   India             IND  2000  1006300.297
3  Israel             ISR  2000     6114.570
4  Malawi             MWI  2000    11801.505
```

To select rows and columns using a mixture of integers and labels, the `loc` attribute can be used in a similar way:

C.R. 19
```python
print(df.loc[df.index[2:5], ['country', 'tcgdp']])
```

```text
  country         tcgdp
2   India  1.728144e+06
3  Israel  1.292539e+05
4  Malawi  5.026222e+03
```

7.3.2 Select Data by Conditions

Instead of indexing rows and columns using integers and names, we can also obtain a sub-dataframe of our interests that satisfies certain (potentially complicated) conditions.

The most straightforward way is with the `[]` operator, where we can put any logical expressions to filter out our dataframe.

C.R. 20
```python
print(df[df.POP >= 20000])
```

```
         country country isocode  year          POP      XRAT         tcgdp         cc      text
          ↪  cg
0      Argentina             ARG  2000    37335.653   0.99950  2.950722e+05  75.716805
 ↪  5.578804
2          India             IND  2000  1006300.297  44.94160  1.728144e+06  64.575551
 ↪  14.072206
5   South Africa             ZAF  2000    45064.098   6.93983  2.272424e+05  72.718710
 ↪  5.726546
6  United States             USA  2000   282171.957   1.00000  9.898700e+06  72.347054
 ↪  6.032454
```

To understand what is going on here, notice `df.POP >= 20000` returns a series of boolean values.

C.R. 21

```python
print(df.POP >= 20000)
```

```
0     True                                                                      text
1    False
2     True
3    False
4    False
5     True
6     True
7    False
Name: POP, dtype: bool
```

In this case, `df[]` takes a series of boolean values and only returns rows with the `True` values. Let's have one more example,

C.R. 22

```python
print(df[(df.country.isin(['Argentina', 'India', 'South Africa'])) & (df.POP > 40000)])
```

```
        country country isocode  year          POP      XRAT         tcgdp         cc      text
         ↪  cg
2         India             IND  2000  1006300.297  44.94160  1.728144e+06  64.575551
 ↪  14.072206
5  South Africa             ZAF  2000    45064.098   6.93983  2.272424e+05  72.718710
 ↪  5.726546
```

However, there is another way of doing the same thing, which can be slightly faster for large dataframes, with more natural syntax using the method `.query()`

C.R. 23

```python
print(df.query("POP >= 20000"))
```

```
         country country isocode  year          POP      XRAT         tcgdp         cc      text
          ↪  cg
0      Argentina             ARG  2000    37335.653   0.99950  2.950722e+05  75.716805
 ↪  5.578804
```

```
3  2          India             IND  2000  1006300.297  44.94160  1.728144e+06  64.575551
   ↪  14.072206
4  5   South Africa             ZAF  2000    45064.098   6.93983  2.272424e+05  72.718710
   ↪  5.726546
5  6  United States             USA  2000   282171.957   1.00000  9.898700e+06  72.347054
   ↪  6.032454
```

We can put more complicated conditions inside the query code.

C.R. 24
```python
print(df.query("country in ['Argentina', 'India', 'South Africa'] and POP > 40000"))
```

```
1        country country isocode  year          POP      XRAT         tcgdp         cc      text
         ↪  cg
2          India             IND  2000  1006300.297  44.94160  1.728144e+06  64.575551
   ↪  14.072206
3  5  South Africa             ZAF  2000    45064.098   6.93983  2.272424e+05  72.718710
   ↪  5.726546
```

We can also allow arithmetic operations between different columns and combine different conditional operations.

C.R. 25
```python
print(df[(df.cc + df.cg >= 80) & (df.POP <= 20000)])
```

```
1     country country isocode  year        POP       XRAT         tcgdp         cc         cgtext
2  4   Malawi             MWI  2000  11801.505  59.543808   5026.221784  74.707624  11.658954
3  7  Uruguay             URY  2000   3219.793  12.099592  25255.961693  78.978740   5.108068
```

C.R. 26
```python
print(df.query("cc + cg >= 80 & POP <= 20000"))
```

```
1     country country isocode  year        POP       XRAT         tcgdp         cc         cgtext
2  4   Malawi             MWI  2000  11801.505  59.543808   5026.221784  74.707624  11.658954
3  7  Uruguay             URY  2000   3219.793  12.099592  25255.961693  78.978740   5.108068
```

For example, we can use the conditioning to select the country with the largest household consumption: gdp share cc.

C.R. 27
```python
print(df.loc[df.cc == max(df.cc)])
```

```
1     country country isocode  year        POP       XRAT         tcgdp        cc        cg   text
2  7  Uruguay             URY  2000  3219.793  12.099592  25255.961693  78.97874  5.108068
```

When we only want to look at certain columns of a selected sub-dataframe, we can use the above conditions with the `.loc[]` command:

1. The first argument takes the condition,

2. The second argument takes a list of columns we want to return.

C.R. 28

```python
print(df.loc[(df.cc + df.cg >= 80) & (df.POP <= 20000), ['country', 'year', 'POP']])
```

```text
   country  year        POP
4   Malawi  2000  11801.505
7  Uruguay  2000   3219.793
```

7.4 Pandas for Panel Data

Econometricians often need to work with more complex data sets, such as panels. Common tasks include:

- Importing data, cleaning it and reshaping it across several axes.
- Selecting a time series or cross-section from a panel.
- Grouping and summarizing data.

pandas contains powerful and easy-to-use tools for solving exactly these kinds of problems. In what follows, we will use a panel data set of real minimum wages from the OECD to create:

- summary statistics over multiple dimensions of our data
- a time series of the average minimum wage of countries in the dataset
- kernel density estimates of wages by continent

We will begin by reading in our long format panel data from a `.csv` file and reshaping the resulting DataFrame with `pivot_table` to build a `MultiIndex`. Additional detail will be added to our DataFrame using pandas' `merge` function, and data will be summarized with the `groupby` function.

C.R. 29 python
```
import matplotlib.pyplot as plt
```

7.4.1 Slicing and Reshaping Data

We will read in a dataset from the OECD of real minimum wages in 32 countries and assign it to `realwage`. The dataset can be accessed with the following link:

C.R. 30 python
```
url1 = "https://raw.githubusercontent.com/" + \
    "QuantEcon/lecture-python/master/source/" + \
    "_static/lecture_specific/pandas_panel/realwage.csv"
```

C.R. 31 python
```
import pandas as pd

# Display 6 columns for viewing purposes
pd.set_option('display.max_columns', 6)

# Reduce decimal points to 2
pd.options.display.float_format = '{:,.2f}'.format

realwage = pd.read_csv(url1)
```

Let's have a look at what we've got to work with which its result can be seen in

C.R. 32
```python
print(realwage.head())  # Show first 5 rows
```

The data is currently in long format, which is difficult to analyze when there are several dimensions to the data. To simplify, we will use `pivot_table` to create a wide format panel, with a MultiIndex to handle higher dimensional data. `pivot_table` arguments should specify the data (values), the index, and the columns we want in our resulting dataframe.

By passing a list in columns, we can create a MultiIndex in our column axis:

C.R. 33
```python
realwage = realwage.pivot_table(values='value',
                                index='Time',
                                columns=['Country', 'Series', 'Pay period'])
print(realwage.head())
```

```text
Country                                    Australia         \
Series     In 2015 constant prices at 2015 USD PPPs
Pay period                                    Annual Hourly
Time
2006-01-01                                20,410.65  10.33
2007-01-01                                21,087.57  10.67
2008-01-01                                20,718.24  10.48
2009-01-01                                20,984.77  10.62
2010-01-01                                20,879.33  10.57
Country                                                        ...  \
Series     In 2015 constant prices at 2015 USD exchange rates  ...
Pay period                                               Annual  ...
Time                                                             ...
2006-01-01                                           23,826.64  ...
2007-01-01                                           24,616.84  ...
2008-01-01                                           24,185.70  ...
2009-01-01                                           24,496.84  ...
2010-01-01                                           24,373.76  ...
Country                                 United States  \
Series     In 2015 constant prices at 2015 USD PPPs
Pay period                                    Hourly
Time
2006-01-01                                      6.05
2007-01-01                                      6.24
2008-01-01                                      6.78
2009-01-01                                      7.58
2010-01-01                                      7.88
Country
Series     In 2015 constant prices at 2015 USD exchange rates
Pay period                                               Annual Hourly
Time
2006-01-01                                           12,594.40   6.05
2007-01-01                                           12,974.40   6.24
2008-01-01                                           14,097.56   6.78
```

```
2009-01-01                                    15,756.42   7.58
2010-01-01                                    16,391.31   7.88
[5 rows x 128 columns]
```

7.5 Application: Long-Run Growth

In this section we will use pandas and matplotlib to download, organize, and visualise historical data on economic growth. In addition to learning how to deploy these tools more generally, we'll use them to describe facts about economic growth experiences across many countries over several centuries.

Such growth facts are interesting for a variety of reasons:

- They explain growth facts is a principal purpose of both *development economics* and *economic history*.
- They are important inputs into historians' studies of geopolitical forces and dynamics.

Therefore, Adam Tooze's account of the geopolitical precedents and antecedents of World War I begins by describing how the Gross Domestic Products (GDP) of European Great Powers had evolved during the 70 years preceding 1914 [3]. Using the very same data Tooze used to construct his figure (with a slightly longer timeline), here is our version of his Chapter 1 figure.

[3] Adam Tooze. The deluge: the great war, america and the remaking of the global order, 1916-1931. 2014.

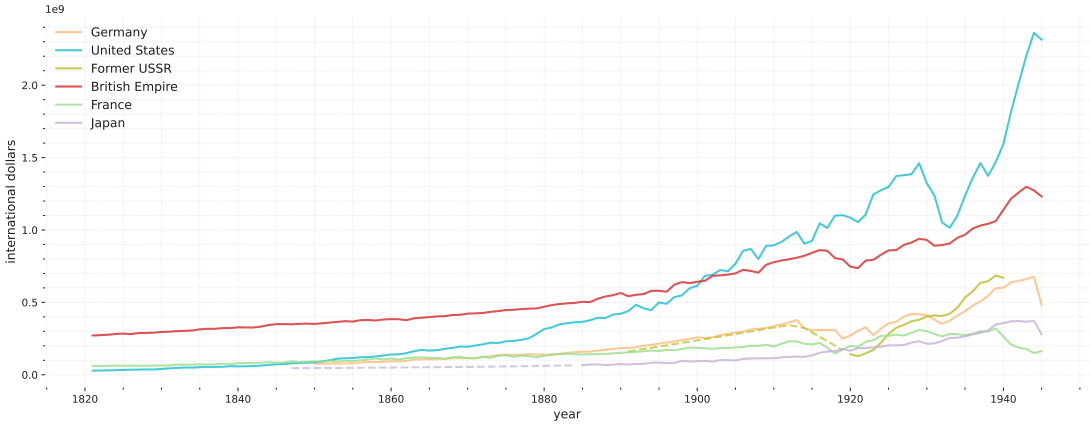


Figure 7.2: The evolution of the GDP of many nations with respect to year.

Chapter 1 of Tooze's work used his graph (**Fig.** 7.2) to show how US GDP started the 19th century way behind the GDP of the British Empire. By the end of the 19th century, US GDP had caught up with GDP of the British Empire, and how during the first half of the 20th century, US GDP surpassed that of the British Empire.

For Adam Tooze, this evolution of GPD was a key geopolitical underpinning for the American century. Looking at this graph and how it set the geopolitical stage for the American 20th century naturally tempts one to want a counterpart to his graph for 2014 or later.

> Reasoning by analogy, this graph perhaps set the stage for an XXX (21st) century, where you are free to fill in your guess for country XXX.

As we gather data to construct those two (2) graphs, we'll also study growth experiences for a

number of countries for time horizons extending as far back as possible. These graphs will portray how the Industrial Revolution[4] began in Britain in the late 18th century, then migrated to one country after another.

[4] Industrial Revolution, in modern history, the process of change from an agrarian and handicraft economy to one dominated by industry and machine manufacturing.

In a nutshell, this exercise records growth trajectories of various countries over long time periods. While some countries have experienced long-term rapid growth across that has lasted a hundred years, others have not. As populations differ across countries and vary within a country over time, it will be interesting to describe both total GDP and GDP per capita as it evolves within a country.

First let's import the necessary packages to explore what the data says about long-run growth:

C.R. 34

```python
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np
from collections import namedtuple
```

> A project initiated by *Angus Maddison* has collected many historical time series related to economic growth, some dating back to the 1st century. The data can be downloaded from the Maddison Historical Statistics by clicking on the Latest Maddison Project Release.

For our work, we are going to read the data from a QuantEcon GitHub repository. QuantEcon, is an online open source education platform for people who are interested in learning more about data science from the perspective of quantitative economics. Our objective in this section is to produce a convenient DataFrame instance containing per capita GDP for different countries.

Here we read the Maddison data into a pandas DataFrame and print it out too see what is in it:

C.R. 35

```python
data_url = "https://github.com/QuantEcon/lecture-python-intro/raw/main/lectures/datasets/mpd2020.xlsx"
data = pd.read_excel(data_url,
                     sheet_name='Full data')
print(data.head())
```

```text
  countrycode      country  year    gdppc      pop
0         AFG  Afghanistan  1820      NaN 3,280.00
1         AFG  Afghanistan  1870      NaN 4,207.00
2         AFG  Afghanistan  1913      NaN 5,730.00
3         AFG  Afghanistan  1950 1,156.00 8,150.00
4         AFG  Afghanistan  1951 1,170.00 8,284.00
```

We can see that this dataset contains GDP per capita (gdppc) and population (pop) for many countries and years. Let's look at how many and which countries are available in this dataset:

C.R. 36

```python
countries = data.country.unique()
print(len(countries))
```

```
169
```
text

We can now explore some of the 169 countries[5] that are available for us to work on. Let's loop over each country to understand which years are available for each country

[5] As with everything even the number of countries is not clearly defined as some sources say it is 196 and some 195.

C.R. 37
```python
country_years = []
for country in countries:
    cy_data = data[data.country == country]['year']
    ymin, ymax = cy_data.min(), cy_data.max()
    country_years.append((country, ymin, ymax))
country_years = pd.DataFrame(country_years,
                    columns=['country', 'min_year', 'max_year']).set_index('country')
print(country_years.head())
```

```
                     min_year  max_year
country
Afghanistan              1820      2018
Angola                   1950      2018
Albania                     1      2018
United Arab Emirates     1950      2018
Argentina                1800      2018
```
text

Let's now reshape the original data into some convenient variables to enable quicker access to countries' time series data. We can build a useful mapping between country codes and country names in this dataset:

C.R. 38
```python
code_to_name = data[
    ['countrycode',
     'country']].drop_duplicates().reset_index(drop=True).set_index(['countrycode'])
```

Now we can focus on GDP per capita (gdppc) and generate a wide data format.

C.R. 39
```python
gdp_pc = data.set_index(['countrycode', 'year'])['gdppc']
gdp_pc = gdp_pc.unstack('countrycode')
```

C.R. 40
```python
print(gdp_pc.tail())
```

```
countrycode         AFG         AGO          ALB          ARE          ARG          ARM          AUS
text
↪ AUT          AZE  ...          USA          UZB          VEN          VNM          YEM          YUG
↪ ZAF         ZMB         ZWE
year
↪ ...
2014          2022.0000  8673.0000   9808.0000  72601.0000  19183.0000   9735.0000  47867.0000
↪ 41338.0000  17439.0000  ...  51664.0000   9085.0000  20317.0000  5455.0000   4054.0000
↪ 14627.0000  12242.0000  3478.0000  1594.0000
```

```
2015         1928.0000  8689.0000  10032.0000  74746.0000  19502.0000  10042.0000  48357.0000
↪ 41294.0000  17460.0000  ...  52591.0000   9720.0000  18802.0000  5763.0000  2844.0000
↪ 14971.0000  12246.0000  3478.0000  1560.0000
2016         1929.0000  8453.0000  10342.0000  75876.0000  18875.0000  10080.0000  48845.0000
↪ 41445.0000  16645.0000  ...  53015.0000  10381.0000  15219.0000  6062.0000  2506.0000
↪ 15416.0000  12139.0000  3479.0000  1534.0000
2017         2014.7453  8146.4354  10702.1201  76643.4984  19200.9061  10859.3783  49265.6135
↪ 42177.3706  16522.3072  ...  54007.7698  10743.8666  12879.1350  6422.0865  2321.9239
↪ 15960.8432  12189.3579  3497.5818  1582.3662
2018         1934.5550  7771.4418  11104.1665  76397.8181  18556.3831  11454.4251  49830.7993
↪ 42988.0709  16628.0553  ...  55334.7394  11220.3702  10709.9506  6814.1423  2284.8899
↪ 16558.3123  12165.7948  3534.0337  1611.4052
[5 rows x 169 columns]
```

We create a variable `color_mapping` to store a map between country codes and colors for consistency as we will be using it in just a little bit later.

C.R. 41

```python
country_names = data['countrycode']

# Generate a colormap with the number of colors matching the number of countries
colors = cm.tab20(np.linspace(0, 0.95, len(country_names)))

# Create a dictionary to map each country to its corresponding color
color_mapping = {country: color for
                 country, color in zip(country_names, colors)}
```

7.5.1 GDP per Capita

In this section we examine GDP per capita over the long run for several different countries.

United Kingdom

First we examine UK GDP growth. To plot the data we can easily access plotting method directly from the dataframe.

C.R. 42

```python
fig, ax = plt.subplots(figsize = (15,6))
country = 'GBR'
gdp_pc[country].plot(
        ax=ax,
        ylabel='international dollars',
        xlabel='year',
        color=color_mapping[country]
    );
```

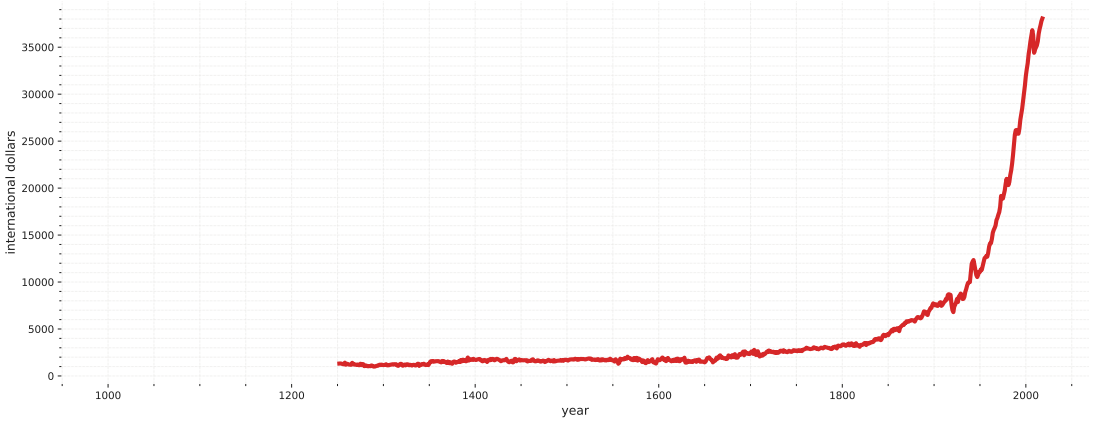


Figure 7.3: GDP per Capita of United Kingdom.

> International dollars are a hypothetical unit of currency that has the same purchasing power parity that the U.S. Dollar has in the United States at a given point in time. They are also known as Geary-Khamis dollars (GK Dollars).

We can see that the data is non-continuous for longer periods in the early 250 years of this millennium, so we could choose to interpolate to get a continuous line plot. Here we use dashed lines to indicate interpolated trends:

C.R. 43 python

```
fig, ax = plt.subplots(figsize = (15,6))
country = 'GBR'
ax.plot(gdp_pc[country].interpolate(),
        linestyle='--',
        lw=2,
        color=color_mapping[country])

ax.plot(gdp_pc[country],
        lw=2,
        color=color_mapping[country])
ax.set_ylabel('international dollars')
ax.set_xlabel('year')
```

Comparing the US, UK, and China

In this section we will compare GDP growth for the US, UK and China. As a first step we create a function to generate plots for a list of countries.

C.R. 44 python

```
def draw_interp_plots(series,        # pandas series
                      country,       # list of country codes
                      ylabel,        # label for y-axis
                      xlabel,        # label for x-axis
```

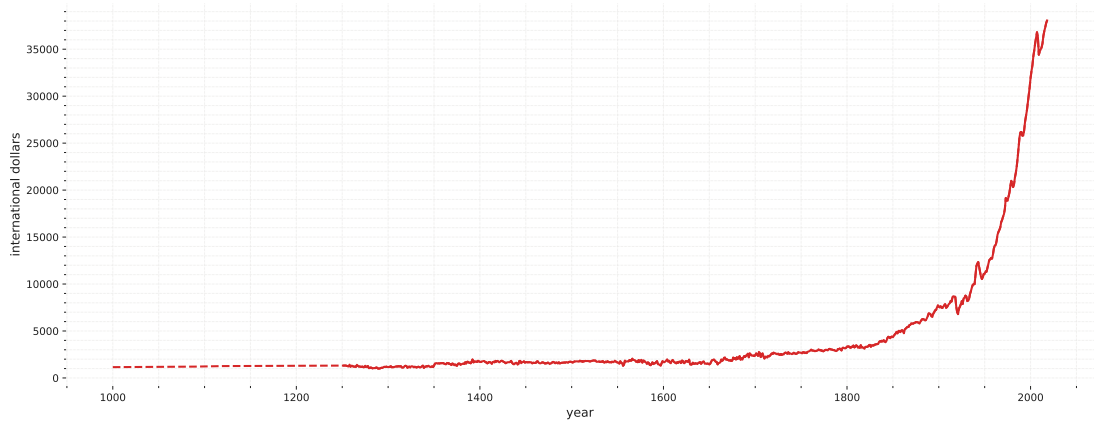


Figure 7.4: GDP per Capita of United Kingdom with continuation.

C.R. 45

python

```
                    color_mapping, # code-color mapping
                    code_to_name,  # code-name mapping
                    lw,            # line width
                    logscale,      # log scale for y-axis
                    ax             # matplolib axis
                   ):

    for c in country:
        # Get the interpolated data
        df_interpolated = series[c].interpolate(limit_area='inside')
        interpolated_data = df_interpolated[series[c].isnull()]

        # Plot the interpolated data with dashed lines
        ax.plot(interpolated_data,
                linestyle='--',
                lw=lw,
                alpha=0.7,
                color=color_mapping[c])

        # Plot the non-interpolated data with solid lines
        ax.plot(series[c],
                lw=lw,
                color=color_mapping[c],
                alpha=0.8,
                label=code_to_name.loc[c]['country'])

        if logscale:
            ax.set_yscale('log')

    # Draw the legend outside the plot
    ax.legend(loc='upper left', frameon=False)
    ax.set_ylabel(ylabel)
    ax.set_xlabel(xlabel)
```

As you can see from this chart, economic growth started in earnest in the 18th century and continued for the next two hundred years.How does this compare with other countries' growth trajectories?

Let's look at the United States (USA), United Kingdom (GBR), and China (CHN)

C.R. 46 python

```python
# Define the namedtuple for the events
Event = namedtuple('Event', ['year_range', 'y_text', 'text', 'color', 'ymax'])

fig, ax = plt.subplots(figsize = (15,6))

country = ['CHN', 'GBR', 'USA']
draw_interp_plots(gdp_pc[country].loc[1500:],
                  country,
                  'international dollars','year',
                  color_mapping, code_to_name, 2, False, ax)

# Define the parameters for the events and the text
ylim = ax.get_ylim()[1]
b_params = {'color':'grey', 'alpha': 0.2}
t_params = {'fontsize': 9,
            'va':'center', 'ha':'center'}

# Create a list of events to annotate
events = [
    Event((1650, 1652), ylim + ylim*0.04,
          'the Navigation Act\n(1651)',
          color_mapping['GBR'], 1),
    Event((1655, 1684), ylim + ylim*0.13,
          'Closed-door Policy\n(1655-1684)',
          color_mapping['CHN'], 1.1),
    Event((1848, 1850), ylim + ylim*0.22,
          'the Repeal of Navigation Act\n(1849)',
          color_mapping['GBR'], 1.18),
    Event((1765, 1791), ylim + ylim*0.04,
          'American Revolution\n(1765-1791)',
          color_mapping['USA'], 1),
    Event((1760, 1840), ylim + ylim*0.13,
          'Industrial Revolution\n(1760-1840)',
          'grey', 1.1),
    Event((1929, 1939), ylim + ylim*0.04,
          'the Great Depression\n(19291939)',
          'grey', 1),
    Event((1978, 1979), ylim + ylim*0.13,
          'Reform and Opening-up\n(1978-1979)',
          color_mapping['CHN'], 1.1)
]

def draw_events(events, ax):
    # Iterate over events and add annotations and vertical lines
    for event in events:
        event_mid = sum(event.year_range)/2
        ax.text(event_mid,
                event.y_text, event.text,
                color=event.color, **t_params)
        ax.axvspan(*event.year_range, color=event.color, alpha=0.2)
        ax.axvline(event_mid, ymin=1, ymax=event.ymax, color=event.color,
```

```python
                    clip_on=False, alpha=0.15)

# Draw events
draw_events(events, ax)
```

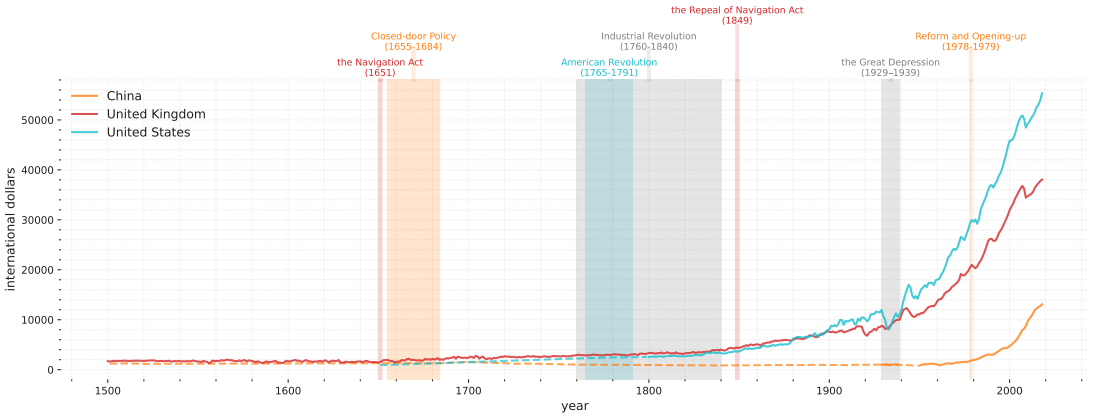


Figure 7.5: GDP per capita of UK, US, China with important economic event imposed.

The preceding graph of per capita GDP strikingly reveals how the spread of the Industrial Revolution has over time gradually lifted the living standards of substantial groups of people.

- Most of the growth happened in the past 150 years after the Industrial Revolution.
- Per capita GDP in the US and UK rose and diverged from that of China from 1820 to 1940.
- The gap has closed rapidly after 1950 and especially after the late 1970s.
- These outcomes reflect complicated combinations of technological and economic-policy factors that students of economic growth try to understand and quantify.

Focusing on China

It is fascinating to see China's GDP per capita levels from 1500 through to the 1970s. Notice the long period of declining GDP per capital levels from the 1700s until the early 20th century. Thus, the graph indicates the following points:

- A long economic downturn and stagnation after the Closed-door Policy by the Qing government.
- China's very different experience than the UK's after the onset of the industrial revolution in the UK.
- How the Self-Strengthening Movement seemed mostly to help China to grow.
- How stunning have been the growth achievements of modern Chinese economic policies by

the PRC that culminated with its late 1970s reform and liberalization.

C.R. 48 python

```python
fig, ax = plt.subplots(figsize = (15,6))

country = ['CHN']
draw_interp_plots(gdp_pc[country].loc[1600:2000],
                  country,
                  'international dollars','year',
                  color_mapping, code_to_name, 2, True, ax)

ylim = ax.get_ylim()[1]

events = [
Event((1655, 1684), ylim + ylim*0.06,
      'Closed-door Policy\n(1655-1684)',
      'tab:orange', 1),
Event((1760, 1840), ylim + ylim*0.06,
      'Industrial Revolution\n(1760-1840)',
      'grey', 1),
Event((1839, 1842), ylim + ylim*0.2,
      'First Opium War\n(18391842)',
      'tab:red', 1.07),
Event((1861, 1895), ylim + ylim*0.4,
      'Self-Strengthening Movement\n(18611895)',
      'tab:blue', 1.14),
Event((1939, 1945), ylim + ylim*0.06,
      'WW 2\n(1939-1945)',
      'tab:red', 1),
Event((1948, 1950), ylim + ylim*0.23,
      'Founding of PRC\n(1949)',
      color_mapping['CHN'], 1.08),
Event((1958, 1962), ylim + ylim*0.5,
      'Great Leap Forward\n(1958-1962)',
      'tab:orange', 1.18),
Event((1978, 1979), ylim + ylim*0.7,
      'Reform and Opening-up\n(1978-1979)',
      'tab:blue', 1.24)
]

# Draw events
draw_events(events, ax)
```

Focusing on the US and UK

Now we look at the United States (USA) and United Kingdom (GBR) in more detail. In the following graph, please watch for:

- Impact of trade policy (Navigation Act).
- productivity changes brought by the Industrial Revolution.

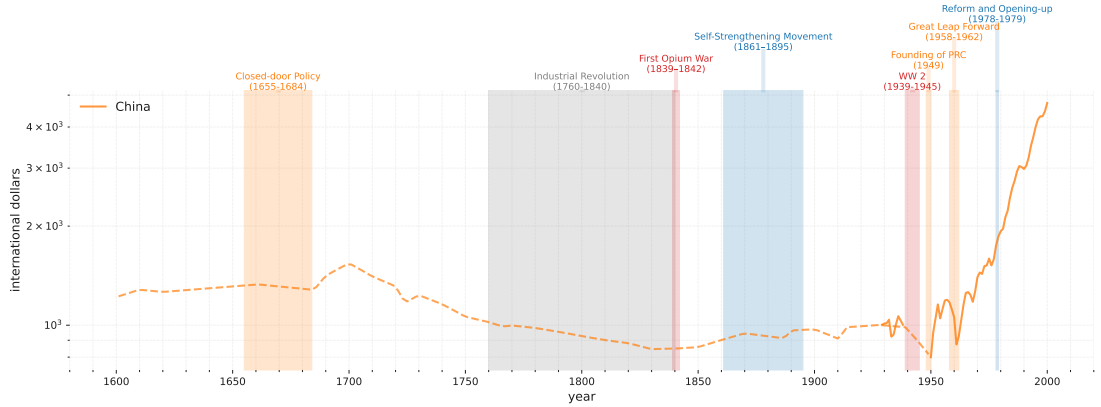


Figure 7.6: GDP per capita of China with important economic event imposed.

- how the US gradually approaches and then surpasses the UK, setting the stage for the "American Century".
- the often unanticipated consequences of wars.
- interruptions and scars left by business cycle recessions and depressions.

C.R. 49

python

```
fig, ax = plt.subplots(figsize = (15,6))

country = ['GBR', 'USA']
draw_interp_plots(gdp_pc[country].loc[1500:2000],
                  country,
                  'international dollars','year',
                  color_mapping, code_to_name, 2, True, ax)

ylim = ax.get_ylim()[1]

# Create a list of data points
events = [
    Event((1651, 1651), ylim + ylim*0.15,
          'Navigation Act (UK)\n(1651)',
          'tab:orange', 1),
    Event((1765, 1791), ylim + ylim*0.15,
          'American Revolution\n(1765-1791)',
          color_mapping['USA'], 1),
    Event((1760, 1840), ylim + ylim*0.6,
          'Industrial Revolution\n(1760-1840)',
          'grey', 1.08),
    Event((1848, 1850), ylim + ylim*1.1,
          'Repeal of Navigation Act (UK)\n(1849)',
          'tab:blue', 1.14),
    Event((1861, 1865), ylim + ylim*1.8,
          'American Civil War\n(1861-1865)',
          color_mapping['USA'], 1.21),
    Event((1914, 1918), ylim + ylim*0.15,
```

C.R. 50

```python
            'WW 1\n(1914-1918)',
            'tab:red', 1),
    Event((1929, 1939), ylim + ylim*0.6,
            'the Great Depression\n(19291939)',
            'grey', 1.08),
    Event((1939, 1945), ylim + ylim*1.1,
            'WW 2\n(1939-1945)',
            'tab:red', 1.14)
]

# Draw events
draw_events(events, ax)
```

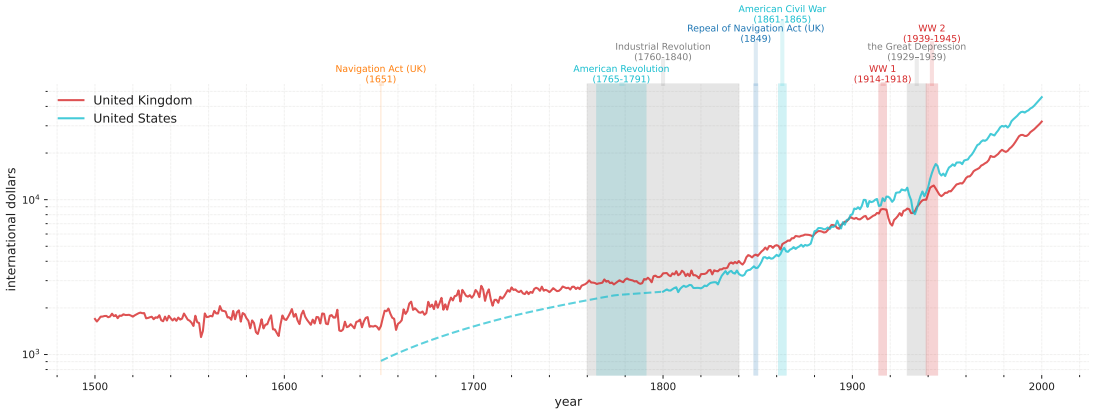


Figure 7.7: GDP per capita of US and UK with important economic event imposed.

7.5.2 GDP Growth

Now we'll construct some graphs of interest to geopolitical historians like Adam Tooze. We'll focus on total Gross Domestic Product (GDP) (as a proxy for "national geopolitical-military power") rather than focusing on GDP per capita (as a proxy for living standards).

C.R. 51

```python
data = pd.read_excel(data_url, sheet_name='Full data')
data.set_index(['countrycode', 'year'], inplace=True)
data['gdp'] = data['gdppc'] * data['pop']
gdp = data['gdp'].unstack('countrycode')
```

Early industrialization (1820 to 1940)

We first visualize the trend of China, the Former Soviet Union, Japan, the UK and the US. The most notable trend is the rise of the US, surpassing the UK in the 1860s and China in the 1880s. The growth continued until the large dip in the 1930s when the Great Depression hit. Meanwhile,

Russia experienced significant setbacks during World War I and recovered significantly after the February Revolution.

C.R. 52
python
```
fig, ax = plt.subplots(figsize = (15,6))
country = ['CHN', 'SUN', 'JPN', 'GBR', 'USA']
start_year, end_year = (1820, 1945)
draw_interp_plots(gdp[country].loc[start_year:end_year],
                  country,
                  'international dollars', 'year',
                  color_mapping, code_to_name, 2, False, ax)
```

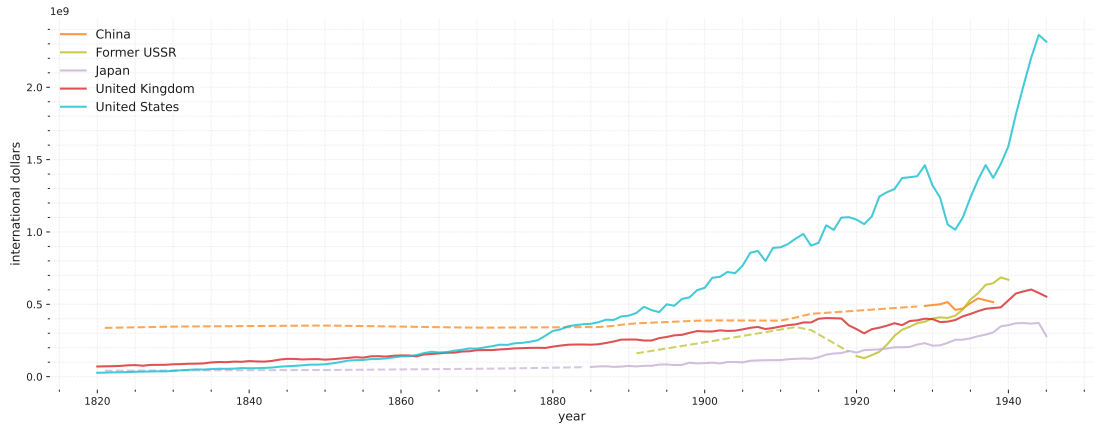


Figure 7.8: GDP in the early industrialization era.

Constructing a plot similar to Tooze's In this section we describe how we have constructed a version of the striking figure we discussed at the start of this lecture. Let's first define a collection of countries that consist of the British Empire (BEM) so we can replicate that series in Tooze's chart.

C.R. 53
python
```
BEM = ['GBR', 'IND', 'AUS', 'NZL', 'CAN', 'ZAF']
# Interpolate incomplete time-series
gdp['BEM'] = gdp[BEM].loc[start_year-1:end_year].interpolate(method='index').sum(axis=1)
```

Now let's assemble our series and get ready to plot them.

C.R. 54
python
```
# Define colour mapping and name for BEM
color_mapping['BEM'] = color_mapping['GBR']  # Set the color to be the same as Great Britain
# Add British Empire to code_to_name
bem = pd.DataFrame(["British Empire"], index=["BEM"], columns=['country'])
bem.index.name = 'countrycode'
code_to_name = pd.concat([code_to_name, bem])
```

C.R. 55
python
```
fig, ax = plt.subplots(figsize = (15,6))
country = ['DEU', 'USA', 'SUN', 'BEM', 'FRA', 'JPN']
start_year, end_year = (1821, 1945)
```

C.R. 56
python
```
draw_interp_plots(gdp[country].loc[start_year:end_year],
                  country,
                  'international dollars', 'year',
                  color_mapping, code_to_name, 2, False, ax)
```

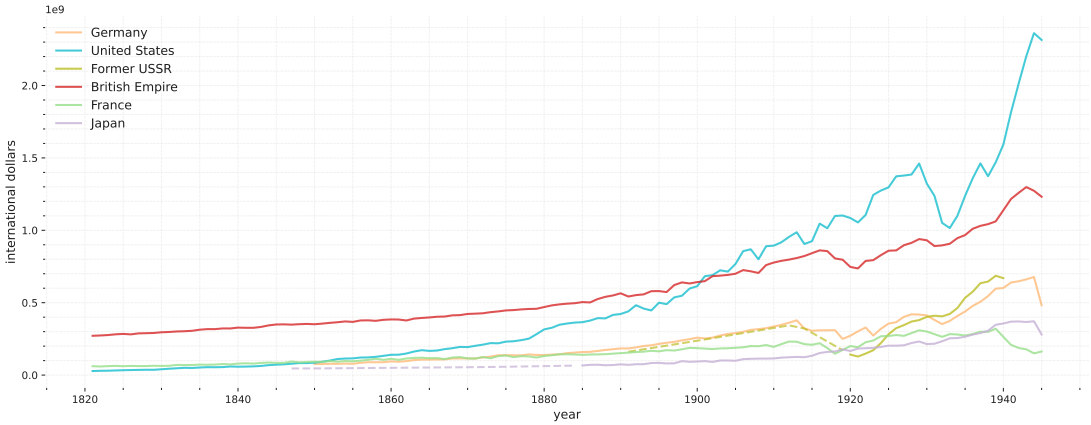


Figure 7.9

At the start of this lecture, we noted how US GDP came from "nowhere" at the start of the 19th century to rival and then overtake the GDP of the British Empire by the end of the 19th century, setting the geopolitical stage for the "American (twentieth) century". Let's move forward in time and start roughly where Tooze's graph stopped after World War II. In the spirit of Tooze's chapter 1 analysis, doing this will provide some information about geopolitical realities today.

The modern era (1950 to 2020)

The following graph displays how quickly China has grown, especially since the late 1970s.

C.R. 57
python
```
fig, ax = plt.subplots(figsize = (15,6))
country = ['CHN', 'SUN', 'JPN', 'GBR', 'USA']
start_year, end_year = (1950, 2020)
draw_interp_plots(gdp[country].loc[start_year:end_year],
                  country,
                  'international dollars', 'year',
                  color_mapping, code_to_name, 2, False, ax)
```

It is tempting to compare this graph with figure Fig. 2.6 that showed the US overtaking the UK near the start of the "American Century".

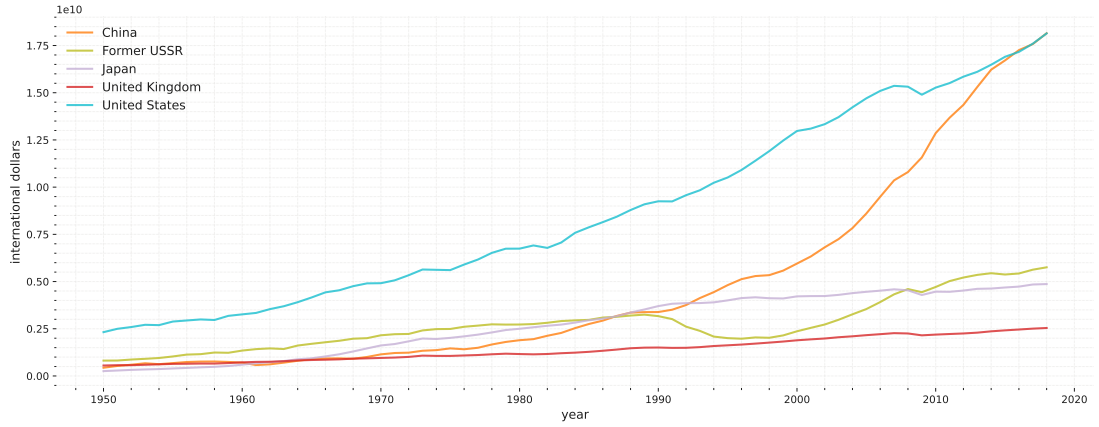


Figure 7.10: GDP in the modern era.

7.5.3 Regional Analysis

We often want to study the historical experiences of countries outside the club of "World Powers". The Maddison Historical Statistics dataset also includes regional aggregations

C.R. 58

```python
data = pd.read_excel(data_url,
                     sheet_name='Regional data',
                     header=(0,1,2),
                     index_col=0)
data.columns = data.columns.droplevel(level=2)
```

We can save the raw data in a more convenient format to build a single table of regional GDP per capita.

C.R. 59

```python
regionalgdp_pc = data['gdppc_2011'].copy()
regionalgdp_pc.index = pd.to_datetime(regionalgdp_pc.index, format='%Y')
```

Let's interpolate based on time to fill in any gaps in the dataset for the purpose of plotting.

C.R. 60

```python
regionalgdp_pc.interpolate(method='time', inplace=True)
```

Looking more closely, let's compare the time series for Western Offshoots and Sub-Saharan Africa with a number of different regions around the world. Again we see the divergence of the West from the rest of the world after the Industrial Revolution and the convergence of the world after the 1950s.

C.R. 61

```python
fig, ax = plt.subplots(figsize = (15,6))
regionalgdp_pc.plot(ax=ax, xlabel='year',
                    lw=2,
                    ylabel='international dollars')
```

C.R. 62

```python
ax.set_yscale('log')
plt.legend(loc='lower center',
           ncol=3, bbox_to_anchor=[0.5, -0.5])
```

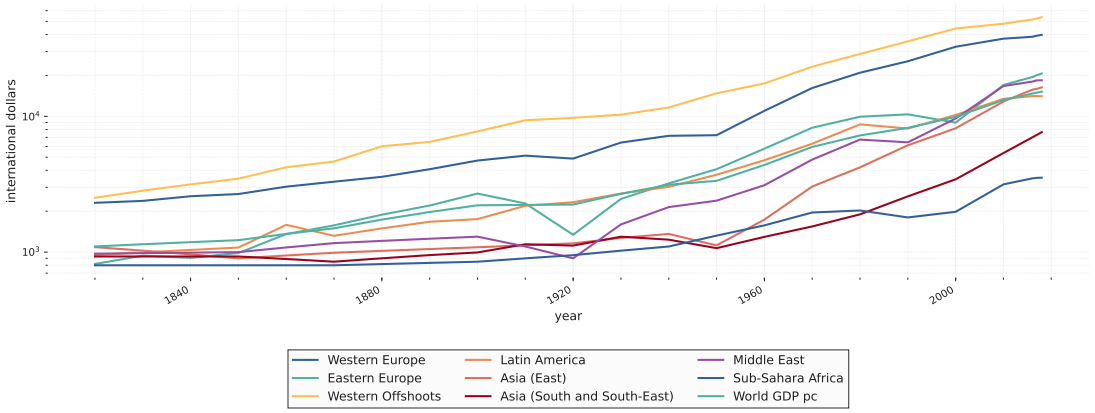


Figure 7.11: Regional GDP per capita.

Part III

Appendix

Appendix A

Tables

Table of Contents

A.1 Introduction 197
A.2 Special Functions 198
A.3 Student-t Distribution 200
A.4 Chi-Square Distribution 202

A.1 Introduction

The following are tables used in solving the exercises present in this book.

A.2 Special Functions

A.2.1 Bessel Function

Table A.1: Particular values of the Bessel function

x	$J_0(x)$	$J_1(x)$	x	$J_0(x)$	$J_1(x)$	x	$J_0(x)$	$J_1(x)$
0.0	1.0	0.0	3.0	-0.2601	0.3391	6.0	0.1506	-0.2767
0.1	0.9975	0.0499	3.1	-0.2921	0.3009	6.1	0.1773	-0.2559
0.2	0.99	0.0995	3.2	-0.3202	0.2613	6.2	0.2017	-0.2329
0.3	0.9776	0.1483	3.3	-0.3443	0.2207	6.3	0.2238	-0.2081
0.4	0.9604	0.196	3.4	-0.3643	0.1792	6.4	0.2433	-0.1816
0.5	0.9385	0.2423	3.5	-0.3801	0.1374	6.5	0.2601	-0.1538
0.6	0.912	0.2867	3.6	-0.3918	0.0955	6.6	0.274	-0.125
0.7	0.8812	0.329	3.7	-0.3992	0.0538	6.7	0.2851	-0.0953
0.8	0.8463	0.3688	3.8	-0.4026	0.0128	6.8	0.2931	-0.0652
0.9	0.8075	0.4059	3.9	-0.4018	-0.0272	6.9	0.2981	-0.0349
1.0	0.7652	0.4401	4.0	-0.3971	-0.066	7.0	0.3001	-0.0047
1.1	0.7196	0.4709	4.1	-0.3887	-0.1033	7.1	0.2991	0.0252
1.2	0.6711	0.4983	4.2	-0.3766	-0.1386	7.2	0.2951	0.0543
1.3	0.6201	0.522	4.3	-0.361	-0.1719	7.3	0.2882	0.0826
1.4	0.5669	0.5419	4.4	-0.3423	-0.2028	7.4	0.2786	0.1096
1.5	0.5118	0.5579	4.5	-0.3205	-0.2311	7.5	0.2663	0.1352
1.6	0.4554	0.5699	4.6	-0.2961	-0.2566	7.6	0.2516	0.1592
1.7	0.398	0.5778	4.7	-0.2693	-0.2791	7.7	0.2346	0.1813
1.8	0.34	0.5815	4.8	-0.2404	-0.2985	7.8	0.2154	0.2014
1.9	0.2818	0.5812	4.9	-0.2097	-0.3147	7.9	0.1944	0.2192
2.0	0.2239	0.5767	5.0	-0.1776	-0.3276	8.0	0.1717	0.2346
2.1	0.1666	0.5683	5.1	-0.1443	-0.3371	8.1	0.1475	0.2476
2.2	0.1104	0.556	5.2	-0.1103	-0.3432	8.2	0.1222	0.258
2.3	0.0555	0.5399	5.3	-0.0758	-0.346	8.3	0.096	0.2657
2.4	0.0025	0.5202	5.4	-0.0412	-0.3453	8.4	0.0692	0.2708
2.5	-0.0484	0.4971	5.5	-0.0068	-0.3414	8.5	0.0419	0.2731
2.6	-0.0968	0.4708	5.6	0.027	-0.3343	8.6	0.0146	0.2728

Continued on next page

Table A.1: Particular values of the Bessel function (Continued)

x	$J_0(x)$	$J_1(x)$	x	$J_0(x)$	$J_1(x)$	x	$J_0(x)$	$J_1(x)$
2.7	-0.1424	0.4416	5.7	0.0599	-0.3241	8.7	-0.0125	0.2697
2.8	-0.185	0.4097	5.8	0.0917	-0.311	8.8	-0.0392	0.2641
2.9	-0.2243	0.3754	5.9	0.122	-0.2951	8.9	-0.0653	0.2559

A.2.2 Error Function, Sine and Cosine Integrals

Table A.2: Particular values of Error function, sine and cosine integrals.

x	erf(x)	Si(x)	ci(x)	x	erf(x)	Si(x)	ci(x)
0.0	0.0	0.0	-∞	2.0	0.9953	1.6054	0.423
0.2	0.2227	0.1996	-1.0422	2.2	0.9981	1.6876	0.3751
0.4	0.4284	0.3965	-0.3788	2.4	0.9993	1.7525	0.3173
0.6	0.6039	0.5881	-0.0223	2.6	0.9998	1.8004	0.2533
0.8	0.7421	0.7721	0.1983	2.8	0.9999	1.8321	0.1865
1.0	0.8427	0.9461	0.3374	3.0	1.0	1.8487	0.1196
1.2	0.9103	1.108	0.4205	3.2	1.0	1.8514	0.0553
1.4	0.9523	1.2562	0.462	3.4	1.0	1.8419	-0.0045
1.6	0.9763	1.3892	0.4717	3.6	1.0	1.8219	-0.058
1.8	0.9891	1.5058	0.4568	3.8	1.0	1.7934	-0.1038
2.0	0.9953	1.6054	0.423	4.0	1.0	1.7582	-0.141

A.3 Student-t Distribution

Table A.3: Values of z for given values of the distribution function F(z) with $m = 1 - 10$.

	Degrees of Freedom									
$F(z)$	1	2	3	4	5	6	7	8	9	10
0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.6	0.32	0.29	0.28	0.27	0.27	0.26	0.26	0.26	0.26	0.26
0.7	0.73	0.62	0.58	0.57	0.56	0.55	0.55	0.55	0.54	0.54
0.8	1.38	1.06	0.98	0.94	0.92	0.91	0.9	0.89	0.88	0.88
0.9	3.08	1.89	1.64	1.53	1.48	1.44	1.41	1.4	1.38	1.37
0.95	6.31	2.92	2.35	2.13	2.02	1.94	1.89	1.86	1.83	1.81
0.975	12.71	4.3	3.18	2.78	2.57	2.45	2.36	2.31	2.26	2.23
0.99	31.82	6.96	4.54	3.75	3.36	3.14	3.0	2.9	2.82	2.76
0.995	63.66	9.92	5.84	4.6	4.03	3.71	3.5	3.36	3.25	3.17
0.995	63.66	9.92	5.84	4.6	4.03	3.71	3.5	3.36	3.25	3.17
0.999	318.31	22.33	10.21	7.17	5.89	5.21	4.79	4.5	4.3	4.14

Table A.4: Values of z for given values of the distribution function F(z) with $m = 11 - 20$.

	Degrees of Freedom									
$F(z)$	11	12	13	14	15	16	17	18	19	20
0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.6	0.26	0.26	0.26	0.26	0.26	0.26	0.26	0.26	0.26	0.26
0.7	0.54	0.54	0.54	0.54	0.54	0.54	0.53	0.53	0.53	0.53
0.8	0.88	0.87	0.87	0.87	0.87	0.86	0.86	0.86	0.86	0.86
0.9	1.36	1.36	1.35	1.35	1.34	1.34	1.33	1.33	1.33	1.33
0.95	1.8	1.78	1.77	1.76	1.75	1.75	1.74	1.73	1.73	1.72
0.975	2.2	2.18	2.16	2.14	2.13	2.12	2.11	2.1	2.09	2.09
0.99	2.72	2.68	2.65	2.62	2.6	2.58	2.57	2.55	2.54	2.53
0.995	3.11	3.05	3.01	2.98	2.95	2.92	2.9	2.88	2.86	2.85
0.995	3.11	3.05	3.01	2.98	2.95	2.92	2.9	2.88	2.86	2.85
0.999	4.02	3.93	3.85	3.79	3.73	3.69	3.65	3.61	3.58	3.55

Table A.5: Values of z for given values of the distribution function F(z) with $m = 21 - 30$.

	Degrees of Freedom (m)									
$F(z)$	21	22	23	24	25	26	27	28	29	30
0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.6	0.26	0.26	0.26	0.26	0.26	0.26	0.26	0.26	0.26	0.26
0.7	0.53	0.53	0.53	0.53	0.53	0.53	0.53	0.53	0.53	0.53
0.8	0.86	0.86	0.86	0.86	0.86	0.86	0.86	0.85	0.85	0.85
0.9	1.32	1.32	1.32	1.32	1.32	1.31	1.31	1.31	1.31	1.31
0.95	1.72	1.72	1.71	1.71	1.71	1.71	1.7	1.7	1.7	1.7
0.975	2.08	2.07	2.07	2.06	2.06	2.06	2.05	2.05	2.05	2.04
0.99	2.52	2.51	2.5	2.49	2.49	2.48	2.47	2.47	2.46	2.46
0.995	2.83	2.82	2.81	2.8	2.79	2.78	2.77	2.76	2.76	2.75
0.995	2.83	2.82	2.81	2.8	2.79	2.78	2.77	2.76	2.76	2.75
0.999	3.53	3.5	3.48	3.47	3.45	3.43	3.42	3.41	3.4	3.39

A.4 Chi-Square Distribution

Table A.6: Values of z for given values of the distribution function F(z) with $m = 1 - 10$.

	Degrees of Freedom (m)									
$F(z)$	1	2	3	4	5	6	7	8	9	0
0.005	0.0	0.01	0.07	0.21	0.41	0.68	0.99	1.34	1.73	2.16
0.01	0.0	0.02	0.11	0.3	0.55	0.87	1.24	1.65	2.09	2.56
0.025	0.0	0.05	0.22	0.48	0.83	1.24	1.69	2.18	2.7	3.25
0.05	0.0	0.1	0.35	0.71	1.15	1.64	2.17	2.73	3.33	3.94
0.95	3.84	5.99	7.81	9.49	11.07	12.59	14.07	15.51	16.92	18.31
0.975	5.02	7.38	9.35	11.14	12.83	14.45	16.01	17.53	19.02	20.48
0.99	6.63	9.21	11.34	13.28	15.09	16.81	18.48	20.09	21.67	23.21
0.995	7.88	10.6	12.84	14.86	16.75	18.55	20.28	21.95	23.59	25.19

Table A.7: Values of z for given values of the distribution function F(z) with $m = 11 - 20$.

	Degrees of Freedom (m)									
$F(z)$	11	12	13	14	15	16	17	18	19	20
0.005	2.6	3.07	3.57	4.07	4.6	5.14	5.7	6.26	6.84	7.43
0.01	3.05	3.57	4.11	4.66	5.23	5.81	6.41	7.01	7.63	8.26
0.025	3.82	4.4	5.01	5.63	6.26	6.91	7.56	8.23	8.91	9.59
0.05	4.57	5.23	5.89	6.57	7.26	7.96	8.67	9.39	10.12	10.85
0.95	19.68	21.03	22.36	23.68	25.0	26.3	27.59	28.87	30.14	31.41
0.975	21.92	23.34	24.74	26.12	27.49	28.85	30.19	31.53	32.85	34.17
0.99	24.72	26.22	27.69	29.14	30.58	32.0	33.41	34.81	36.19	37.57
0.995	26.76	28.3	29.82	31.32	32.8	34.27	35.72	37.16	38.58	40.0

Table A.8: Values of z for given values of the distribution function F(z) with $m = 21 - 30$.

	Degrees of Freedom (m)									
$F(z)$	21	22	23	24	25	26	27	28	29	30
0.005	8.03	8.64	9.26	9.89	10.52	11.16	11.81	12.46	13.12	13.79
0.01	8.9	9.54	10.2	10.86	11.52	12.2	12.88	13.56	14.26	14.95

Continued on next page

Table A.8: Values of z for given values of the distribution function F(z) with $m = 21 - 30$. (Continued)

	Degrees of Freedom (m)									
$F(z)$	21	22	23	24	25	26	27	28	29	30
0.025	10.28	10.98	11.69	12.4	13.12	13.84	14.57	15.31	16.05	16.79
0.05	11.59	12.34	13.09	13.85	14.61	15.38	16.15	16.93	17.71	18.49
0.95	32.67	33.92	35.17	36.42	37.65	38.89	40.11	41.34	42.56	43.77
0.975	35.48	36.78	38.08	39.36	40.65	41.92	43.19	44.46	45.72	46.98
0.99	38.93	40.29	41.64	42.98	44.31	45.64	46.96	48.28	49.59	50.89
0.995	41.4	42.8	44.18	45.56	46.93	48.29	49.64	50.99	52.34	53.67

Bibliography

[1] David C Hoaglin, Frederick Mosteller, and John W Tukey. *Understanding robust and exploratory data analysis*. Vol. 76. John Wiley & Sons, 2000.