

Data and Error Analysis

Ryan D. Martin

February 13, 2018

Contents

1	Introduction	1
1.1	Motivation - “doing good science”	1
1.2	Motivation - the case for using computers	3
2	Python	4
2.1	The basics	4
2.1.1	Variables and the assignment operator	4
2.1.2	Printing to screen	4
2.1.2.1	Formatting numerical output	5
2.1.3	Comments	6
2.1.4	Lists	6
2.1.4.1	Operations on lists	7
2.1.5	Types of variables	8
2.2	Controlling the flow	9
2.2.1	Scope	9
2.2.2	If-statements	9
2.2.3	Loops	10
2.3	Defining custom functions	11
2.4	Importing modules	13
2.5	Reading and writing data files	15
2.6	Modules of use in scientific computing	16
2.6.1	Working with numbers, numpy	17
2.6.1.1	Reading and writing to file	18
2.6.1.2	Calculating properties of an array	19
2.6.1.3	Operations on arrays	19
2.6.2	Making plots with matplotlib and pylab	21
2.6.2.1	Scatter plot and plotting a function	21
2.6.2.2	Plot with error bars	22
2.6.2.3	Plotting histograms	23
2.6.3	Sympy: Symbolic computations in python	27
2.6.4	Curve fitting with scipy optimize	28
3	Determining uncertainties and presenting data	32
3.1	The importance of uncertainties	32

3.1.1	What do we mean by “uncertainty”?	33
3.1.2	Systematic uncertainties	34
3.2	Prescription for determining certain uncertainties	35
3.2.1	Uncertainties from scales - “Half of the smallest division”	35
3.2.2	Repeatable measurements - “Mean and uncertainty on the mean”	36
3.2.3	Counting experiments - The square root rule	38
3.2.4	Measuring a fraction/proportion - The binomial error on a ratio	38
3.3	Overestimating uncertainties is also bad!	38
3.4	Reporting measurements, significant figures	39
3.5	Comparing measured quantities	41
3.6	Summary	42
4	Error Propagation	44
4.1	The Min-Max method	44
4.1.1	Summing and subtracting	44
4.1.2	Multiplication and division	46
4.1.2.1	Uncertainties in functions	49
4.1.2.2	Summary and comments on the Min-Max method	50
4.2	Adding in quadrature	51
4.3	Uncertainty in a function of one variable	52
4.4	The general case: functions of multiple variables	53
4.5	Correlated uncertainties	57
4.6	Averaging numbers with uncertainties	62
4.7	Summary	63
5	Statistics - Presenting Data	65
5.1	Data and statistics	65
5.2	Basic statistical parameters of a set of measurements	66
5.2.1	Mean and variance	66
5.2.2	Error on the mean	67
5.2.3	Mode and median	67
5.2.4	Moments, skewness and kurtosis	68
5.2.5	Covariance and correlation	69
5.3	Statistical parameters for a function of a set of measurements	72
5.3.1	The mean	73
5.3.2	The variance and standard deviation	73
5.3.3	Multiple variables	74
5.4	The histogram	76
5.5	Comparing statistical data to a model	80
5.6	Summary	80
6	Statistics - Distributions	82
6.1	The binomial distribution	82
6.1.1	Statistical properties of the binomial distribution	87
6.2	The Poisson distribution	90

6.2.1	Statistical properties of the Poisson distribution	97
6.2.2	Time between Poisson events	97
6.3	Statistical hypothesis testing	98
6.4	Summary	102
7	Statistics - The Normal Distribution	104
7.1	Properties of the normal distribution	108
7.1.1	Statistical properties	108
7.1.2	Area underneath the normal distribution pdf	108
7.2	Why most measurements are normally distributed	112
7.2.1	The Central Limit Theorem	115
7.3	Combining normally distributed quantities	116
7.3.1	Addition of a number to a normally distributed quantity	116
7.3.2	Multiplication of a number with a normally distributed quantity	117
7.3.3	Sum of normally distributed quantities	118
7.3.4	Function of normally distributed quantities	120
7.4	Determining parameters of the normal distribution from data	121
7.4.1	The principle of maximum likelihood	121
7.4.2	Estimating the mean and standard deviation of a normal distribution from data	122
7.4.3	The error on the mean	125
7.5	Averaging multiple measurements with standard errors	127
7.6	Summary	131
8	Statistics - Modelling Data	134
8.1	Fitting a straight line	134
8.1.1	Linear problems	134
8.1.2	The Method of Least Squares	136
8.1.3	Residuals	141
8.1.4	Correlated parameters	142
8.2	Non-linear fits	145
8.3	Goodness of fit	151
8.4	Summary	153
9	Statistics - Probabilities and the Bayesian approach	156
9.1	Kolmogorov's axioms	156
9.2	Bayes' Theorem	159
9.3	The Bayesian interpretation of probability	162
9.3.1	The probability that a coin is fair	164
9.3.2	Fitting a straight line the Bayesian way	170
10	Statistics - The Monte Carlo Method	175
10.1	Random numbers from a computer	175
10.1.1	Random numbers that follow a given distribution	178
10.2	Monte Carlo simulations	181

10.2.1	Simulating coin tosses	181
10.2.2	Simulating a poutine restaurant	183
10.2.3	Monte Carlo simulation for error propagation	185

List of Python Examples

2.1	Assignment operator	4
2.2	Assignment operator	4
2.3	Printing	5
2.3	Output	5
2.4	Printing multiple items	5
2.4	Output	5
2.5	Formatted printing	5
2.5	Output	5
2.6	Using comments	6
2.6	Output	6
2.7	Defining a list	6
2.8	Indexing a list	6
2.8	Output	7
2.9	Length of a list	7
2.9	Output	7
2.10	Operations on lists	7
2.10	Output	8
2.11	Two dimensional lists	8
2.11	Output	8
2.12	If statements	9
2.12	Output	10
2.13	For in loops	10
2.13	Output	10
2.14	For loop with range	11
2.14	Output	11
2.15	Loop example	11
2.16	Defining a custom function	11
2.17	Using a custom function	12
2.17	Output	12
2.18	Functions with no return values	12
2.18	Output	12
2.19	Functions with multiple return values	12
2.19	Output	13
2.20	Importing the math module	13

2.20	Output	13
2.21	Importing a module as	13
2.21	Output	13
2.22	Import everything from a module	13
2.22	Output	14
2.23	Importing a single function from a module	14
2.23	Output	14
2.24	Defining your own module	14
2.25	Importing your own module	14
2.25	Output	14
2.26	Reading and writing to a file	15
2.27	Reading and writing to a file with formatted data	16
2.28	The numpy module for arrays	17
2.28	Output	18
2.29	Reading and writing numpy arrays from/to a file	18
2.29	Output	19
2.30	Numpy for statistical properties of an array of values.	19
2.30	Output	19
2.31	Numpy operations on arrays	19
2.31	Output	20
2.32	Defining a function to operated on a numpy array	20
2.32	Output	20
2.33	Pylab to make a scatter plot and plot a function	21
2.33	Output	21
2.34	Pylab to make a plot with error bars	22
2.34	Output	22
2.35	Pylab to plot a histogram	23
2.35	Output	23
2.36	Making a histogram with customized bins	24
2.37	Plotting multiple histograms	24
2.37	Output	25
2.38	Making a normalized histogram and making sub figures	26
2.38	Output	26
2.39	Sympy for symbolic computations in python	27
2.40	Using sympy to compute a symbolic derivative	27
2.40	Output	28
2.41	Substituting numerical values into expressions from sympy	28
2.41	Output	28
2.42	Defining a function to use for fitting	28
2.43	Fitting a function to a set of xy data with error bars	29
2.44	Obtaining the errors from the fit	29
2.45	A complete fit including residuals	29
2.45	Output	31
3.1	Mean and std with numpy	37
3.1	Output	37

3.2	Mean and std with QExpy	37
3.2	Output	38
4.1	Min-Max method	49
4.1	Output	50
4.2	Symbolic derivative error propagation with sympy	55
4.2	Output	55
4.3	Numerical evaluation of symbolic values from sympy	56
4.3	Output	56
4.4	Symbolic-based derivative error with sympy	56
4.4	Output	57
4.5	Different error calculations for a set of values	60
4.5	Output	60
5.1	Basic statistical quantities	67
5.1	Output	68
5.2	Calculating covariance and correlation	70
5.2	Output	70
5.3	Making histograms	79
5.3	Output	80
6.1	Evaluating binomial probabilities	85
6.1	Output	85
6.2	Evaluating binomial quantities	88
6.2	Output	89
6.3	Computing probabilities for the Poisson distribution	92
6.3	Output	93
6.4	Computing a 90% confidence interval	96
6.4	Output	96
6.5	Computing binomial probability of null hypothesis	100
6.5	Output	100
6.6	Computing Poisson probability of null hypothesis	101
6.6	Output	102
7.1	Comparison of probabilities from binomial and normal distributions	107
7.1	Output	107
7.2	Comparison of probabilities from Poisson and normal distributions	109
7.2	Output	109
7.3	SF and CDF for normal distribution	112
7.3	Output	112
8.1	Linear fit with least squares	138
8.1	Output	140
8.2	Plotting residuals	141
8.2	Output	142
8.3	Plot chi-squared as a function of the fit parameters	143
8.3	Output	143
8.4	Defining a function for a non-linear fit	147
8.5	Non-linear fit in python	147
8.6	Complete non-linear fit with residuals	148

8.6	Output	150
9.1	Posterior probability for a coin toss	165
9.1	Output	165
9.2	Coverage of the binomial error on a ratio	166
9.2	Output	166
9.3	Posterior probability for a coin toss with different priors	167
9.3	Output	168
9.4	Defining a confidence range from the posterior probability	168
9.4	Output	169
10.1	Park-Miller minimal standard LCG	177
10.1	Output	177
10.2	Converting uniform PRNs to exponential PRNs	179
10.2	Output	179
10.3	Standard distributions of PRNs	180
10.3	Output	180
10.4	Simulating 10 coin tosses	181
10.4	Output	181
10.5	Many simulations of throwing a coin 10 times	181
10.5	Output	182
10.6	Monte Carlo of the poutine restaurant	183
10.6	Output	184
10.7	Standard error propagation	185
10.7	Output	186
10.8	Monte Carlo error propagation	186
10.8	Output	187
10.9	Selecting 68% coverage errors	188
10.9	Output	189

1

Introduction

This book looks at the transition between the physics in textbooks, where everything is exact, to physics in the real world, where nothing can be measured with infinite precision. We thus need a robust framework in which we can apply the laws of physics to the real world, and to test them. We also need a framework that allows us to gather data, understand what the data mean, and then make robust conclusions about what they imply.

1.1 Motivation - “doing good science”

When we “do science”, we are really interested in modelling the real world. At the basis of the Scientific Method is the idea that we can form a hypothesis (a “theory”) and then test if that hypothesis is wrong (by performing an “experiment”).

For example, an experimenter may want to test the hypothesis that the circumference of a circle is $2\pi r$. Suppose that they present the following report of their experiment:

We used a ruler to measure the radius of a circle and obtained 1 cm. We wrapped a string around the circle and determined the length of the string to be 6.4 cm. The theory predicts that the circumference should be $2\pi(1\text{ cm})=6.28318530718\text{ cm}$, which is clearly not equal to 6.4 cm. The theory is thus invalidated.

This trivial example already uncovers a number of aspects that are important in “doing good science”. Based on what is stated above, can we decide whether the theory is invalidated by the experiment? Since we are quite certain that the theory is not invalidated, there are a few questions that we would want the experimenter to address:

- What are the uncertainties in the measurements of the radius and circumference, and how were they determined?
- What is the precision of the ruler?
- Are the digits presented for the predicted value significant?
- How was the radius determined?
- Can the experiment be performed with a much bigger circle?

- What was the diameter of the string?
- How was the ruler calibrated?
- Where was the value of π taken from?

As you can see, we need to be very precise when we do science. Obviously we should be as precise as possible when performing the measurements, but we need to be equally precise in how we report our experimental results. When we write lab reports or scientific publications for our peers, we should always ask ourselves how one could question the validity of our results and then address those concerns.

The subject of this book is “Data and Error Analysis”, and the goal is to establish a strong foundation for analyzing data and presenting them in a way that is scientifically meaningful. We should note that the word “error” in our context is used as a synonym for “uncertainty” rather than “mistake”. The idea of “error analysis” is to understand the uncertainties that are involved in the presentation of a result (and to a much lesser extent the tedious “propagation of errors” that are so feared in undergraduate laboratories).

Example 1-1: How would you present the results of the experiment attempting to test the hypothesis that the circumference of a circle is $2\pi r$?

Here is a possible way to present and discuss the results:

Experimental procedure and uncertainty determination

We used a stainless steel compass (brand: Acme, model: Pro2000) with a 0.5 mm diameter graphite tip to draw ten 1 cm radius circles onto a set of small post-it notes. The radii of the circles were measured using a stainless steel ruler (brand: Acme, model: RRCatcher2000) with 1 mm graduations. The ruler was used to set the stride of the compass, and was then again used to confirm the radius of the drawn circles (by measuring the distance between the hole made by the compass into the paper and the perimeter of the circles).

The average radius of the circles was determined to be 1.0 ± 0.08 cm. We found that the mean measured radius of the 10 circles was 1 cm with a standard deviation of 0.08 cm, which we use as the central value and uncertainty. We believe this uncertainty is representative of the fact that the circles have a width of 0.5 mm (due to the size of the graphite tip on the compass) and that the ruler has 1 mm graduations.

We then used pieces of string (brand: Acme, model: CatCatcher2000, diameter: 0.8 mm, material: wool) to measure the circumferences of the 10 circles, by wrapping them around the circles, cutting them to length, and then measuring their lengths with the ruler. We obtained a mean value of 6.4 cm with a standard deviation of 0.12 cm (which we use as the central value and uncertainties on the measurement of the circumference) for the 10 circles. The larger variation in the length of the string as compared to that of the radius is likely due to some stretch in the string, the inability to exactly lay it

over the circumference, and the inaccuracy in cutting it to the exact length.

Using the accepted value of π from Wikipedia, the predicted value of the circumference is:

$$2\pi(1.0 \pm 0.08) \text{ cm} = (6.3 \pm 0.5) \text{ cm}$$

We find the predicted value above to be consistent with the measured value of $(6.40 \pm 0.12) \text{ cm}$. The hypothesis that the circumference of a circle is $2\pi r$ is thus verified within the accuracy of this experiment. Unfortunately, we only had small post-it notes, limiting the size of the circles that we could draw to 1 cm radius. We recommend performing this experiment with much larger circles to reduce the relative uncertainties in the measurements of the radius and circumference.

Comment: *Even this more “precise” report on the experiment has holes in it (were the uncertainties in the individual 10 measurements used? What are the 10 values that were measured, were there any outliers? Is using the standard deviation as the uncertainty justified? How was the diameter of the graphite tip and string determined? How was the error in the radius propagated to the uncertainty in the predicted value? Was the ruler calibrated?)*

1.2 Motivation - the case for using computers

As we have seen, an effective way to understand the uncertainty in a measurement is often to perform the measurement multiple times. In realistic scenarios, this can lead to a lot of data to analyze, which can lead to painful calculations if performed by hand. Even calculating the standard deviation of 10 measurements is not particularly pleasant to do with a calculator, even when it has a function to do so. Furthermore, we often collect data in digital form (e.g. a piece of electronic apparatus recording data to files) and it would be an ineffective use of time to then transcribe those data to analyze them by hand when they are already in a form that can be handled by a computer. Finally, the propagation of errors often involves rather painful formulas with derivatives, squares, square roots that can be handled much more easily and with less chances of making mistakes with a simple computer program. Hopefully, we do not need to convince you further that learning a little programming will make your life a lot easier in the long run.

In this course, we will use the python programming language, which is easy to learn and widely accepted. Most data scientists now use python, it is a de-facto requirement in most technology start-up companies and is heavily used throughout physics. The language is also open source and free, and it is extremely easy to find online help due to its wide acceptance. The next chapter gives a small introduction to python, to give you a basis for reproducing the calculations presented in this text.

2

This chapter is meant to serve as an introduction to programming in python and to be used as a reference. Do not worry about memorizing all of the commands, this will come automatically with practice.

2.1 The basics

This section covers some of the basic syntax in python.

2.1.1 Variables and the assignment operator

In a computer program, one uses “variables” and assigns them values. A variable can be thought of as an address inside of the computer’s memory that can store a value. If we want to put a value into the computer’s memory, we first create a variable and then “assign” that variable the desired value. We can then access the value that we stored by telling the program that we are interested in the value that is stored in a particular variable. This is the most basic operation in writing a computer program. In python, the syntax is the following:

Python Code 2.1: Assignment operator

```
a = 2
```

In the above, we have created a variable, **a**, and assigned it the value of 2. The equal sign is called the “**assignment operator**”, and it always “puts the thing on the right **into** the thing on the left”. Notice that python automatically recognizes that we are storing a number into the variable **a**, and python will thus treat **a** like a number. The following code thus has the expected behaviour and the new variable **b** will have the number 4 stored.

Python Code 2.2: Assignment operator

```
a = 2  
b = 2*a
```

2.1.2 Printing to screen

Now, you may not believe that the variable **b** above is actually equal to 4. Luckily, you can ask the computer program to “print” out the value that is stored in a variable, by using the

`print()` function:

Python Code 2.3: Printing

```
a = 2
b = 2*a
print(b)
```

2.3: Output

4

This will print the value of 4 to the output of the program. `print()` is the first example of a function. Functions can take any number of “arguments” and then act on them. In this case, we “passed” the variable `b` to the function `print()` (by putting it into the parenthesis) and then the function took care of printing that to the output. The `print()` function is quite flexible, and can take any number of arguments of different types, separated by commas:

Python Code 2.4: Printing multiple items

```
a = 2
b = 2*a
print("The value of a is",a,"and the value of b is ",b)
```

2.4: Output

The value of a is 2 and the value of b is 4

In this case, we passed 4 different arguments to the `print()` function. The second and fourth arguments, our variables `a` and `b`, are numbers. The first and third arguments are what we call “strings” (strings of characters), which we must include in quotation marks, so that python does not think that these are variables.

2.1.2.1 Formatting numerical output

Python cannot guess how many decimals it makes sense to display when you print a number; often, you want to control this. This can be done using the `format` “method”.

Python Code 2.5: Formatted printing

```
a = 2.
b = 3./2.
c = 2./3.
print("first={:.1f}, second= {:.2f}, third={:.4f}".format(a,b,c))
```

2.5: Output

first=2.0, second= 1.50, third=0.6667

The `format` method works as follows. You first define a “template string” (in the above case, `"a={:.1f}, b= {:.2f}, c={:.4f}"`) with quotation marks. You then add a `.format()` to the end of it. The `format()` method will look inside of the string for occurrences of curly brackets (`{}`). Inside of the curly brackets, you can tell it what format you would like for the number: in our case `{:.nf}` means that we would like to display *n* decimals (*.n*) of a floating point number (*f*). Thus, `{:.4f}` displays 4 decimals. The final component, is that one tells `format()` which numbers we actually want to format that way, by passing those numbers

as arguments to `format()` in the same order in which we want them to appear in the curly braces. What really happens is that `format()` sequentially replace the curly brackets with the “formatted version” of the numbers that it is given (in our case, the variables `a,b,c`).

One particular trick to pay attention to in python, is that it will prefer to think of variables as integers rather than decimal numbers if we do not specify that they are decimal. This can have some unintended consequences, if python mistakenly thinks we want a variable to be a integer instead of a decimal number (a “float”). To avoid any ambiguity, when we assign a value to a variable, it is good practice to add a decimal point to it so that python will not mistakenly treat it as an integer. Thus, it is best to say `a=2.` or `a=2.0`, rather than `a=2` (unless of course, we intend for `a` to be an integer).

2.1.3 Comments

As code becomes more complex, it is important to be able to insert some comments so that we can remember what we did (or so that others can understand what your code is doing). In python, comments are preceded with the number (or pound) sign `'#'`¹, and anything that follows will be ignored by the computer:

Python Code 2.6: Using comments

```
#Here we have a simple program
a = 2 #a is set to 2
b = 2*a #b is twice a
#Now we print out the result:
print("The value of a is",a,"and the value of b is",b)
```

2.6: Output

The value of a is 2 and the value of b is 4

Always comment your code. You cannot over comment! You would be surprised at how quickly you will forget the logic in your code if you have no comments.

2.1.4 Lists

A particularly useful type of variable in python is the “list”. This is analogous to “arrays” in other programming languages. A list is used to hold, well, a list of values. For example, let’s say that we wish to work with a set of measurements, for example radii of circles from example 1-1, and hold them into a single variable, `radii`, we can use a list as follows:

Python Code 2.7: Defining a list

```
radii=[0.9,1.0,0.95,1.1,1.2,1.0,0.8,0.85,1.05,1.0]
```

Python automatically recognizes that the variable `radii` should be a list, since we are trying to assign a set of 10 comma-separated values to it. We can work with the whole list, but we can also get a particular element out of the list, by using the access operator (`'[]'`) and the index of the element:

Python Code 2.8: Indexing a list

```
radii=[0.9,1.0,0.95,1.1,1.2,1.0,0.8,0.85,1.05,1.0]
```

¹This is not called a hashtag!

```
print(radii[2])
```

2.8: Output

```
0.95
```

One thing to note is that the above `print()` function printed out the third element of the list and not the second one. This is because **python indexes the first item in the list with the index 0**. If the list has `n` elements in it, then the last element has the index `n-1`. If you try to ask for an element that is beyond the size of the list, e.g. `radii[10]`, bad things happen. This is because you are really asking for a value stored in the memory of the computer in a location where you have not stored anything.

A useful operation on a list is to ask how many elements are in the list; this is done with the `len()` function:

Python Code 2.9: Length of a list

```
radii=[0.9,1.0,0.95,1.1,1.2,1.0,0.8,0.85,1.05,1.0]
print("We measured",len(radii),"radii")
```

2.9: Output

```
We measured 10 radii
```

2.1.4.1 Operations on lists

The code below illustrates a variety of operations that can be performed on lists, such as inserting and removing values. Try it for yourself and make sure you understand what is going on.

Python Code 2.10: Operations on lists

```
#a list of numbers
array=[1,2,3,4,5,6]
#len(array) is the number of elements in the list
print("The size of the list is",len(array))
#lists start a 0
print("Element 1 of the list is",array[1])
#You can print a list:
print("The list is:", array)
#You can append an element to the end:
array.append(7)
print("The list is now:",array)
#Remove the 5th element
array.pop(5)
print("The list is now:",array)
#Remove the last element
array.pop()
print("The list is now:",array)
#Insert the number 9 at position 3
array.insert(3,9)
print("The list is now:",array)
#Reverse the elements in the list
array.reverse()
print("The list is now:",array)
```



```

#Sort the list
array.sort()
print("The list is now:",array)
#You can mix variable types in the list (even including other lists)
mixedarray=["hello",42,array]
print("The mixed array:",mixedarray)
print("Some elements from it:", mixedarray[0],
      mixedarray[1],mixedarray[2][3])
#You can slice a list to generate a new list
subarray=array[2:4]
print("The sub list:",subarray)

```

2.10: Output

```

The size of the list is 6
Element 1 of the list is 2
The list is: [1, 2, 3, 4, 5, 6]
The list is now: [1, 2, 3, 4, 5, 6, 7]
The list is now: [1, 2, 3, 4, 5, 7]
The list is now: [1, 2, 3, 4, 5]
The list is now: [1, 2, 3, 9, 4, 5]
The list is now: [5, 4, 9, 3, 2, 1]
The list is now: [1, 2, 3, 4, 5, 9]
The mixed array: ['hello ', 42, [1, 2, 3, 4, 5, 9]]
Some elements from it: hello 42 4
The sub list: [3, 4]

```

We should point out that in the second last example, we are accessing a list using two indices. This is because one can nest lists inside of each other. A good analogy is to think of a list as a vector. If each entry in the vector is a vector itself, then we really have a matrix. For example, we can build a 2x3 matrix in the following way:

Python Code 2.11: Two dimensional lists

```

matrix = [[1,2,3],[4,5,6]]
print("The first row:",matrix[0])
print("Element [0][2]:",matrix[0][2])

```

2.11: Output

```

The first row: [1, 2, 3]
Element [0][2]: 3

```

The `matrix` list in this case has 2 elements, and each of those elements is a list of 3 values. `matrix[0]` thus refers to the first set of 3 elements, and `matrix[0][2]` to the third element of that set.

2.1.5 Types of variables

We have already seen a few of the different types of variables that python offers (and it typically can infer from the context what type of variable you intend to use). A few common types of variables are:

- **integers:** These are signed integer numbers.
- **booleans:** These are variables that can be either `True` or `False`.

- **floats:** These are signed decimal numbers; they usually cannot be represented exactly and are subject to round-off errors.
- **strings:** These are “strings” of characters and the value of the variable is always surrounded by quotes (single or double). For example `"someNumber"` is interpreted as a string, whereas `someNumber` is interpreted as a variable (that could be of any type).
- **lists:** These are arrays of variables; each element in the array does not have to be of the same type, and lists can be nested. Elements in a list have a well defined position in the list and can be referenced by using their index (e.g. `array[2]` is the third element of a list since the first element has index 0).
- **dictionaries:** These are similar to lists, but instead of referencing elements by a numbered index, they are referenced by a string (which we call the “key”). Technically, the key does not need to be a string, but it has to be “non-mutable”; a string is just the most common usage case. Each key must be unique. Dictionaries use curly braces when they are defined. For example, `students = {'bob':123,'jane':456}`, and are referenced using square braces: `sn=students['bob']`.
- **tuples:** These are values that are grouped together with a parenthesis and can be indexed in a way similar to a list.
- **objects:** These are more complex data types that can be user defined, for example, one could define a “student” object with a variety of properties (name, student number, grades) and write code that directly manipulates the objects. Strings and lists are actually objects in python, and that is why they have many available functions to manipulate them.

2.2 Controlling the flow

At this point, we can do some basic calculations and pretty much anything that a calculator can do. However, we cannot do anything very complex without controlling the flow of our program. This section introduces a few key ways to control the flow of the program.

2.2.1 Scope

In python, we can define small little units of code that are only executed in certain conditions. Those units of code are **preceded by a line that ends with a colon (':')** and **must be indented to the right** compared to the rest of the code. We say that the part of the code that is indented forms a ‘scope’; whatever happens in the scope, stays in the scope, just like Vegas. Scopes can be nested within each other. Variables that are defined inside of a scope cease to exist outside the scope. Variables that are defined before and outside of a scope, are accessible in that scope.

2.2.2 If-statements

One of the most useful types of scope, is the scope defined by an “if-statement”. Code inside the scope of an if-statement is executed only if the if statement evaluates to the boolean value `True`:

Python Code 2.12: If statements

```
#define 3 variables:
a = 2
b = 3
```

```
c = 4
if b > a: #True
    print("b is bigger than a")
if c < a: #False
    print("c is smaller than a")
if 2*a == c: #True
    print("2a is equal to c")
```

2.12: Output

```
b is bigger than a
2a is equal to c
```

In this case, the code inside the scope of the second if-statement is never executed, because the condition before the colon is False. The greater and lesser than symbols are pretty obvious, but note the use of the double equal sign, `==`, to test if two values are equal. This is because the single equal sign is already taken (it is the assignment operator), so if we want to test if two things are equal, we must use the “equality operator”, `==`.

2.2.3 Loops

The other very useful type of scope is the “loop”. This is where computers really excel at doing repetitive tasks. The most common type of loop is the “for” loop, which defines a scope that is executed a specified number of times. A common application is to “loop” through all the elements inside of a list. For example:

Python Code 2.13: For in loops

```
radii=[0.9,1.0,0.95,1.1,1.2,1.0,0.8,0.85,1.05,1.0]
for r in radii:
    print("r =",r)
```

2.13: Output

```
r = 0.9
r = 1.0
r = 0.95
r = 1.1
r = 1.2
r = 1.0
r = 0.8
r = 0.85
r = 1.05
r = 1.0
```

Again, the syntax is similar as that for the if-statement. We first have a statement that defines the loop and ends with a colon, followed by a section that is indented to the right (the scope of the for loop). In this case, the `for r in radii` statement means the following: “Create a new variable called `r`, and then execute the code in the scope one time for each value inside of `radii`, assigning that value to `r` each time”. This particular example will print all of the values inside of the list.

A useful function for loops is the `range()` function. In its most basic implementation (when passed only 1 argument, `n`) it returns a list of numbers from 0 to `n-1`. This makes is

particularly easy to loop through a list using the index of the elements:

Python Code 2.14: For loop with range

```
radii=[0.9,1.0,0.95,1.1,1.2,1.0,0.8,0.85,1.05,1.0]
n = len(radii)
for index in range(n):
    #the variable index will have values 0..9
    print("Element",index,"has value:", radii[index])
```

2.14: Output

```
Element 0 has value: 0.9
Element 1 has value: 1.0
Element 2 has value: 0.95
Element 3 has value: 1.1
Element 4 has value: 1.2
Element 5 has value: 1.0
Element 6 has value: 0.8
Element 7 has value: 0.85
Element 8 has value: 1.05
Element 9 has value: 1.0
```

Example 2-1: Use a for loop to create 2 lists that hold the numbers from 1 to 100 and their squares, respectively

Python Code 2.15: Loop example

```
numbers=[] # an empty list that we will fill
squares=[]
for index in range(100):
    x=index+1 # since the first value is 0
    numbers.append(x)
    squares.append(x*x)
print("the numbers are:",numbers)
print("the squares are:",squares)
```

2.3 Defining custom functions

We have already encountered a few functions; `print()` and `range()`. Functions may take “arguments” (they don’t have to), typically would do something (using the arguments if applicable), and may return a value (but they don’t have to). Functions are extremely useful, especially as a way to make your code re-usable and to avoid re-writing the same code over and over.

You must first define a function before you can use it. Defining a function is a way for you to personalize python and give it additional functionality. Let’s write a simple function to return the factorial of a number

Python Code 2.16: Defining a custom function

```
#define the factorial function
```

```
def factorial(n):
    fact=1 # keep track of the factorial
    for i in range(n):
        fact = fact*(i+1) #+1 because the first i is zero
    return fact
```

Functions are defined using the **def** statement, followed by the name of the function (**factorial** in this case) and then, if applicable, the arguments that the function needs in parenthesis. The statement ends with a colon and the next line is indented. This defines the scope of the function. To calculate the factorial, we declared a variable that will hold our result (**fact**). We then needed to do a for loop to calculate the factorial. Finally, outside of the scope of the for loop but still inside the scope of the function, we used the **return** statement to pass the calculated number (**fact**) out of the function. The function can then be used as follows:

Python Code 2.17: Using a custom function

```
print("3 factorial is",factorial(3))
print("10 factorial is",factorial(10))
```

2.17: Output

```
3 factorial is 6
10 factorial is 3628800
```

You have now added the functionality of calculating factorials into python!

Functions need not take an argument or return a value, for example:

Python Code 2.18: Functions with no return values

```
#define a function to print a warning
def printWarning():
    print("Hey, you should be warned here")

#use the function:
printWarning()
```

2.18: Output

```
Hey, you should be warned here
```

Functions can also return multiple values (as a tuple):

Python Code 2.19: Functions with multiple return values

```
#define a function of 1 argument
def squareCube(x):
    #return a tuple
    return x*x, x*x*x

#use the function to assign values to 2 variables
x2 ,x3 =squareCube(5)

print("5 squared is",x2,", 5 cubed is",x3)
```

2.19: Output

```
5 squared is 25 , 5 cubed is 125
```

In reality, python is not actually returning multiple values, but rather it is bundling the two values into a tuple. In practice, you can however pretend that it is in fact returning multiple comma-separated values.

2.4 Importing modules

Fortunately, many have come before us to write a lot of useful functions, and made them available for us to use. Python comes already bundled with a large amount of functions already written for us, and these are available through “modules”. In order to be able to use the functions inside of a module, you must “import” the module. This is typically done at the top of your program. For example, the `math` module has a large number of functions for doing math. To use the square root function for example, you would import the `math` module as follows:

Python Code 2.20: Importing the math module

```
import math
a=2
sqrta = math.sqrt(a)
print("the square root of a is ",sqrta)
```

2.20: Output

```
the square root of a is  1.4142135623730951
```

We may also find it useful to change the name of a package when we import it (e.g. to make it shorter if we have to type it a lot). In this case, we can use the `import ... as` format:

Python Code 2.21: Importing a module as

```
import math as m
a=2
sina = m.sin(a)
print("the sin of a is ",sina)
```

2.21: Output

```
the sin of a is  0.9092974268256817
```

There are other ways to “import” functions from modules, but the above ways are the most explicit (that is, it is clear when you call the function `math.sqrt()` that the function belongs to the `math` module).

For completeness, other equivalent ways to achieve the same functionality are:

Python Code 2.22: Import everything from a module

```
from math import *
a=2
sqrta =sqrt(a) # don't need the math. in this case
```

```
print("the square root of a is ",sqrta)
```

2.22: Output

```
the square root of a is 1.4142135623730951
```

which imports all functions from the `math` module and makes them available directly (without needing the `math.`), or:

Python Code 2.23: Importing a single function from a module

```
from math import sqrt
a=2
sqrta =sqrt(a) # don't need the math. in this case
print("the square root of a is ",sqrta)
```

2.23: Output

```
the square root of a is 1.4142135623730951
```

which imports only the `sqrt` function (so for example, `sin(a)`) would not work.

It is very easy to create your own modules. Simply place the definitions of your functions into a file called 'mymodule.py' (where 'mymodule' can be whatever name you choose), and you can then use them by simply typing `import mymodule` (without the '.py'). For example, you may have a file called 'errorAnalysis.py' with the following content:

Python Code 2.24: Defining your own module

```
def factorial(n):
    fact=1 # keep track of the factorial
    for i in range(n):
        fact = fact*(i+1) #+1 because the first i is zero
    return fact

def squareCube(x):
    #return a tuple
    return x*x, x*x*x
```

and you can then use these functions in a program:

Python Code 2.25: Importing your own module

```
import errorAnalysis

print("8 factorial is:",errorAnalysis.factorial(8))
print("square and cube of 5 are:",errorAnalysis.squareCube(5))
```

2.25: Output

```
8 factorial is: 40320
square and cube of 5 are: (25, 125)
```

There are a lot of modules available in python, some are included directly and many can be downloaded from the internet. Do not be afraid to search the internet for modules that may be of use; chances are that if you have something you need done, someone has already

written a module for it.

2.5 Reading and writing data files

Reading and writing data from and to files is very useful, as opposed to directly “hardcoding” data into a program (as we did above for the radii). This is useful if we have a large number of measurements that all need to be analysed in the same way or if the data themselves were already written into a file by a data acquisition system.

Let’s say that we have a text file with the following data in it (representing measurements of radius and circumference from an experiment like that in example 1-1), and let’s assume the file is named ‘radcirc.dat’:

```
0.9    5.4
1.0    6.2
0.95   6.2
1.1    5.9
1.2    7.9
1.0    6.0
0.8    5.1
0.85   4.8
1.05   6.7
1.0    6.5
```

The data are in 2 columns, one for radius and one for circumference. We now wish to read these values into our program, compute $2\pi r$ from the radius, compare that with the measured circumference, and then output the difference to a new file, called ‘results.dat’. This can be done with the following code:

Python Code 2.26: Reading and writing to a file

```
import math #because we need pi, and it's defined in math

#open the file with the data, need quotation marks on the file name!
dataFile = open('radcirc.dat', 'r') #open in read mode

#open the file to write the output:
resultFile = open('results.dat', 'w') #open in write mode

#loop over each line in the file:
for line in dataFile:
    #line contains both numbers, so we split them up into list
    lineData = line.split()
    #take each value from the list and put them into new variables
    #we need to convert them to numbers (float) because python assumes
    #that they are strings of characters:
    r=float(lineData[0])
    circ=float(lineData[1])

    #the predicted circumference is:
```



```

predicted=2*math.pi*r

#the difference with measured one is:
difference=predicted - circ
print(difference, file=resultFile)

```

The file 'results.dat' now contains a single column with the differences between the predicted and measured circumferences. You can see that we used the `print()` function, with an additional “named argument” to specify that we would like to print to the file, rather than to the screen; in this case, we passed our file variable (`resultFile`) to the argument that is named `file`.

If you look at the file 'results.dat', you will find that the numbers have more decimal places that would really be considered significant. We can thus use the `format()` method to clean it up.

Python Code 2.27: Reading and writing to a file with formatted data

```

import math #because we need pi, and it's defined in math

#open the file with the data, need quotation marks on the file name!
dataFile = open('radcirc.dat', 'r') #open in read mode

#open the file to write the output:
resultFile = open('results.dat', 'w') #open in write mode

#loop over each line in the file:
for line in dataFile:
    #line contains both numbers, so we split them up into list
    lineData = line.split()
    #take each value from the list and put them into new variables
    #we need to convert them to numbers (float) because python assumes
    #that they are strings of characters:
    r=float(lineData[0])
    circ=float(lineData[1])

    #the predicted circumference is:
    predicted=2*math.pi*r

    #the difference with measured one is:
    difference=predicted - circ
    print("{:.2f}".format(difference), file=resultFile)

```

2.6 Modules of use in scientific computing

Data science needs in industry have led to the rapid development of python modules for data analysis. Traditionally, programming languages like python, which are “interpreted” on the fly as the computer tries to run the code, are slower at processing numbers than compilable languages like C++. In order to alleviate this issue somewhat, people have programmed modules that provide “pythonic” (easy to program) interfaces for dealing with numbers that, behind the scenes, are much more efficient than regular python code. These interfaces basically figure out what the user is trying to accomplish and then run the calculations

using C++ code before passing the result back in python. Some of these interfaces are quite complex and actually allow the code to be run on highly efficient graphical processing units (graphics card); these are highly optimized for matrix algebra and anything where parallel computations are efficient.

The other aspect of data analysis that is often needed is a way to visualize data and make graphs, and there are a variety of modules for that as well.

A common module (actually, a set of modules) for scientific work in python is “scipy”. Scipy (a “stack” of modules), comes with most scientific python packages (such as Anaconda, Python(x,y), etc.) and includes modules for plotting (matplotlib), scientific calculations (scipy), symbolic math (sympy), efficiently using arrays of number (numpy), and others. We will give a brief overview of numpy and matplotlib since they are the most useful in our context.

2.6.1 Working with numbers, numpy

In scientific computing, we work predominantly with numbers, and usually a lot of them (think of a list of measurements). We already saw that the python list is a handy way to hold an array of numbers. The list has a lot of useful functions (e.g. dynamically changing the size of the list, searching) that are not very useful in numerical computation, and this added functionality makes lists inefficient for scientific computing. At the core of the numpy module is a type of array that is much more efficient than lists for dealing with numbers. All of the elements in a numpy array must be of the same type. Numpy arrays can also be mult-dimensional (e.g. a matrix or tensor). There are also a variety of functions that can act on numpy arrays, to fill them, evaluate properties about them, and change their shape. Here are a few examples:

Python Code 2.28: The numpy module for arrays

```
# We import numpy (and by convention shorten it to np, since
# we use it a lot)
import numpy as np

# a simple one-dimensional array
a = np.array([1,2,3,4,5,6])

# we can print the array and see its "shape":
print("The array is",a," and has shape:",a.shape, "and the second element is
", a[1])
print("The size of the array is",a.size)

#we can change the shape to be 2D, e.g. 2x3:
a=a.reshape(2,3)
print("The array is :\n",a,"\n and has shape:",a.shape)

#we access an element of a two d array as expected:
print("The 1,2 element is",a[1][2])
print("The size of the array is",a.size)

#we can create an array full of zeroes:
```

```

#we specify the shape of the array with a tuple (hence the extra parenthesis)
z=np.zeros( (4,5) ) # 4 by 5 array of zeros
print("Zeroes: \n", z)

#we can create an array full of ones:
#for a 1D array, no need to use a tuple for the shape
o=np.ones(10) # array of 10 times the number 1
print("Ones:",o)

#an array with numbers in a certain range:
seq1 = np.arange(10,30,5) #numbers from 10 to 30 spaced by 5
print("Sequence 1:", seq1)

#if using floats, better to use linspace
seq2 = np.linspace(0,10,11) #11 numbers from 0 to 10
print("Sequence 2:", seq2)

```

2.28: Output

```

The array is [1 2 3 4 5 6] and has shape: (6,) and the second element is 2
The size of the array is 6
The array is :
[[1 2 3]
 [4 5 6]]
and has shape: (2, 3)
The 1,2 element is 6
The size of the array is 6
Zeroes:
[[ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]]
Ones: [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
Sequence 1: [10 15 20 25]
Sequence 2: [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]

```

Numpy has a lot of more advanced useful features, such as the ability to directly load a file into an array, to write an array into a file, to calculate properties of the array, and to manipulate arrays. Following, are a few advanced examples for reference.

2.6.1.1 Reading and writing to file

Here is a simple way to read and write a numpy array directly to a file. In this example, it uses the default format, which is to separate entries in the file with a space. One can specify different delimiters, such as commas, if preferred.

Python Code 2.29: Reading and writing numpy arrays from/to a file

```

import numpy as np

#make a 4x5 array from random numbers that are normally distributed
a=np.random.normal(7,1,(4,5)) #mean = 7, stdev = 1, shape=(4,5)
print("a:\n",a)

#now save to file (with 2 decimal places)
#for Windows, need the '\r\n', otherwise, '\n' works on Mac and Linux:

```

```
np.savetxt('normaldist.dat',a, fmt='%.2f',newline='\r\n')
```

```
#read from file:
b=np.loadtxt('normaldist.dat')
print("b:\n",b)
```

2.29: Output

```
a:
[[ 7.72906707  7.58554688  6.55403963  8.204429    6.36589465]
 [ 6.32161056  6.46633883  6.94348052  6.16305886  8.22179026]
 [ 7.13446799  6.99732091  5.98000653  7.58423135  6.75268651]
 [ 6.28267233  8.28811545  6.97310529  9.60489909  8.2377974  ]]

b:
[[ 7.73  7.59  6.55  8.2  6.37]
 [ 6.32  6.47  6.94  6.16  8.22]
 [ 7.13  7.   5.98  7.58  6.75]
 [ 6.28  8.29  6.97  9.6  8.24]]
```

2.6.1.2 Calculating properties of an array

Numpy includes some basic functions to calculate simple properties of arrays (such as the mean of the elements or the sum of the elements):

Python Code 2.30: Numpy for statistical properties of an array of values.

```
import numpy as np

#make an array of 15 random numbers that are normally distributed
a=np.random.normal(7,1,15) #mean = 7, stdev = 1, shape = 15
print("a:\n",a)

#the average of those numbers and the standard deviation:
print("The mean is ",a.mean()," the standard deviation is: ",a.std(ddof=1))

#the sum of those numbers:
print("The sum is ",a.sum()," and the average is thus: ",a.sum()/a.size)
```

2.30: Output

```
a:
[ 7.46161595  7.69000179  5.89548669  7.01653231  5.79241436  6.50478794
  7.90049186  7.35861676  7.94790841  7.0088528  7.25540553  6.93505626
  9.12845098  5.97135201  6.9343631 ]
The mean is  7.12008911701  the standard deviation is:  0.882240259864
The sum is  106.801336755  and the average is thus:  7.12008911701
```

2.6.1.3 Operations on arrays

Numpy arrays can be used in computations (e.g. multiplying or summing arrays together) and operated on with numpy functions (such as taking the square root of all elements).

Python Code 2.31: Numpy operations on arrays

```
import numpy as np

#make two arrays of numbers
a=np.linspace(1,5,5)
b=np.linspace(6,10,5)
```

```

print("a: ",a)
print("b: ",b)

#add arrays together, element by element:
sumAB=a+b
print("sum: ",sumAB)

#multiply, element by element:
productAB=a*b
print("product: ",productAB)

#multiply all elements by a scalar:
twoA=2*a
print("2 times a: ",twoA)

#apply a function to elements:
squareA=np.square(a)
print("a squared: ",squareA)

```

2.31: Output

```

a: [ 1.  2.  3.  4.  5.]
b: [ 6.  7.  8.  9. 10.]
sum: [ 7.  9. 11. 13. 15.]
product: [ 6. 14. 24. 36. 50.]
2 times a: [ 2.  4.  6.  8. 10.]
a squared: [ 1.  4.  9. 16. 25.]

```

You can even define a function that is can act on both arrays or on regular numbers. For example, the following will add two numbers “in quadrature”, or if given two arrays as inputs, it will return an array of the values added in quadrature:

Python Code 2.32: Defining a function to operated on a numpy array

```

import numpy as np
#Define a simple function to add two numbers in quadrature:
def addInQuad (e1,e2):
    return np.sqrt(e1*e1+e2*e2)

print("2 and 3 added in quadrature is ",addInQuad(2,3))

#make two arrays of numbers
a=np.linspace(1,5,5)
b=np.linspace(6,10,5)
print("a:",a)
print("b:",b)

#c is an array where each element is the elements of a and b
#added in quadrature
c=addInQuad(a,b)
print("c:",c)

```

2.32: Output

```

2 and 3 added in quadrature is  3.60555127546
a: [ 1.  2.  3.  4.  5.]

```

```
b: [ 6.    7.    8.    9.   10.]
c: [ 6.08276253  7.28010989  8.54400375  9.8488578  11.18033989]
```

2.6.2 Making plots with matplotlib and pylab

Visualizing data is an important way to understand them. The `matplotlib` module is the most popular module for plotting and visualizing data in python. The `pylab` module uses the `matplotlib` module and adds some functionality to give it a Matlab feel (importing `pylab` is actually equivalent to importing both `matplotlib` and `numpy`). When searching for help on the internet for plotting, it may be useful to search for `matplotlib` or `pyplot` rather than `pylab`, as those are the modules that do the actual plotting.

2.6.2.1 Scatter plot and plotting a function

Matplotlib can easily plot the data from numpy arrays. The `plot` function takes as its first two arguments arrays of the same length containing the x and y coordinates of points to plot. The function can take additional optional arguments to specify the looks of the plot (whether to use points or an interpolated line, what color to use, a possible label, etc.).

The following code produces a simple plot of the sine and cosine functions, by building numpy arrays to hold the values and then plotting them (the result is shown in Figure 2.1):

Python Code 2.33: Pylab to make a scatter plot and plot a function

```
import numpy as np
import pylab as pl
import math

#create an array holding values between 0 to 2pi
x=np.linspace(0,2*math.pi,50) #values of x

#use the fact that we can operate element-wise on a whole array, to create
#arrays with value of sin and cos:
y=np.sin(x)
z=np.cos(x)

#plot (x,y) and (x,z)
pl.plot(x,y,label="sin(x)") #default is a blue line
pl.plot(x,z,'ro',label="sin(x)") #plot red circles ('ro') instead

#make it pretty:
pl.xlabel("angle in radians")
pl.ylabel("sin or cos")
pl.title("Trig functions")
pl.legend(loc='upper center')
pl.grid(True)

#show the plot:
pl.show()
```

2.33: Output

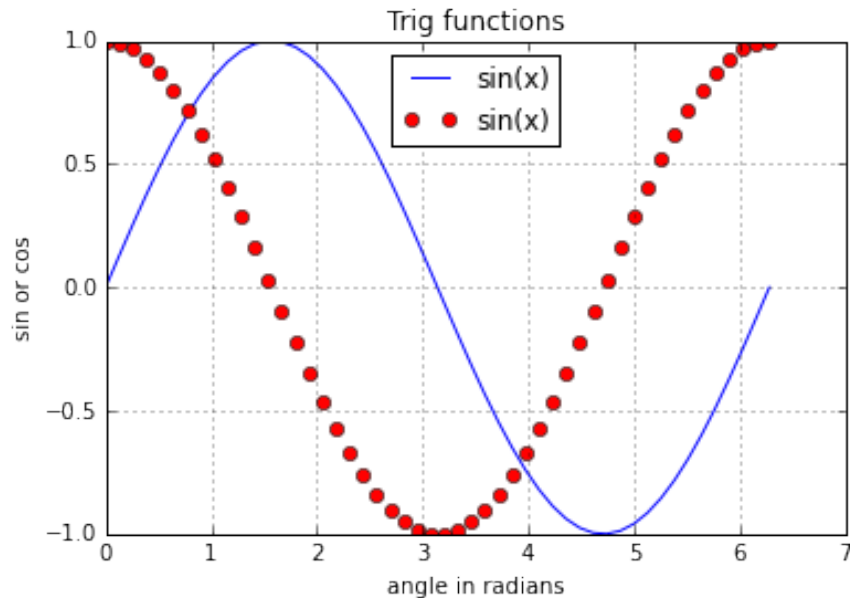


Figure 2.1: Two data sets plotted with a legend.

2.6.2.2 Plot with error bars

The `errorbar()` function allows us to plot a set of data points with error bars. The function expects the first two arguments to be numpy arrays of the x and y coordinates of the data points, and the error bars on those points can be specified using the `xerr` and `yerr` keywords. The error bars can be specified as arrays (thus each point could have different sized error bars), or as a constant number (all points then having the same errorbar). The result is shown in Figure 2.2.

Python Code 2.34: Pylab to make a plot with error bars

```
import numpy as np
import pylab as pl
import math

#plot (x,y)
x=np.linspace(0,2*math.pi,50) # 50 values between 0 and 2pi
y=np.sin(x)

#let's add variable errors in y and fixed errors in x
dy=0.2*np.sqrt(np.abs(y)) #variable errors in y
dx=0.1 #constant error bards in x

pl.errorbar(x,y,yerr=dy,xerr=dx, fmt='o')
pl.title("Plot with error bars")
pl.show()
```

2.34: Output

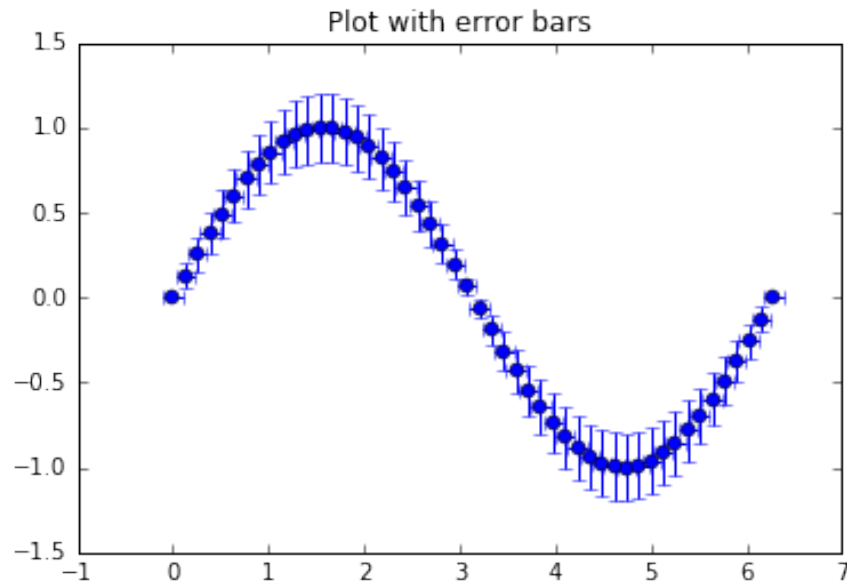


Figure 2.2: Data set with constant x and variable y error bars.

2.6.2.3 Plotting histograms

A particularly useful way to plot data of a statistical nature is the “histogram” (See Chapter 5 for details on histograms). This is done very easily with the `hist()` function. (see Fig. 2.3):

Python Code 2.35: Pylab to plot a histogram

```
import numpy as np
import pylab as pl
import math

#create a set of normally distributed random numbers
x=np.random.normal(7,1,1000)

#histogram the values
n,bins,patches=pl.hist(x,bins=40)

#Make it pretty:
pl.xlabel("value")
pl.ylabel("frequency of value")
pl.title("Normally distributed variable")

#Set the axis range
pl.axis([0,12,0,80])

#show the plot:
pl.show()
```

2.35: Output

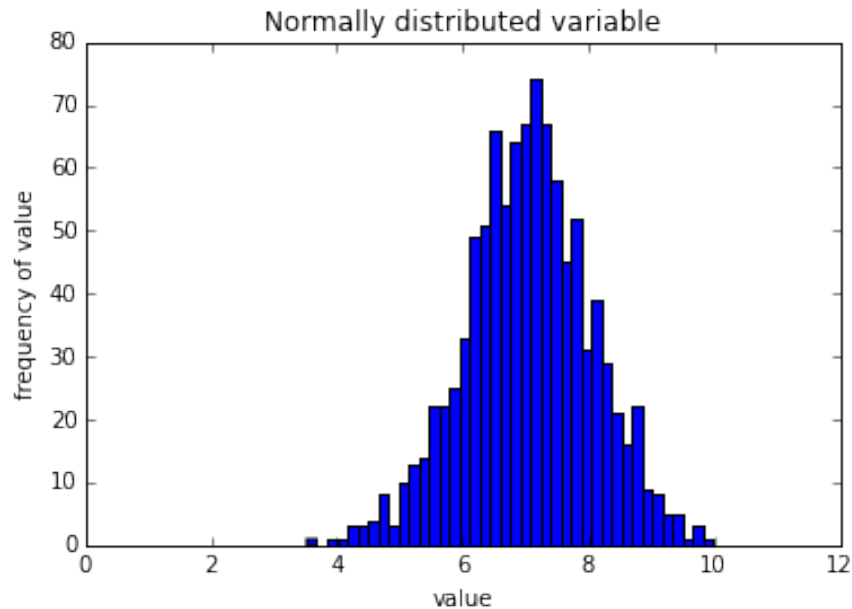


Figure 2.3: Histogram with 40 bins of random values normally distributed with a mean of 7 and standard deviation of 1.

The `hist()` function plots the histogram and returns a tuple `(n,bins,patches)`. The first element in the tuple (`n`) is an array containing the number of counts in each bin, the second element (`bins`) is an array with the left edge of each bin, and the third element (`patches`) contains information on drawing the actual histogram (e.g. where to place rectangles) and has no analysis use for us.

The `hist()` function takes as first argument the array of values that it should histogram, and it can take a variety of other arguments to control the display of the histogram. A particularly important argument is the `bins` argument, which controls the binning of the histogram. If this is a single number, then python will automatically figure out the range of values in the array and create that many equal bins (40 bins in the example above). One can also specify `bins` by setting it equal to an array of values which correspond to the left side of the bins that we specify (and the last value in the array in the right edge of the highest bin). For example:

Python Code 2.36: Making a histogram with customized bins

```
#histogram the values
n, bins, patches=pl.hist(x, bins=[6.5,7,7.1,7.2,7.3,7.4,7.5,8])
```

would create a histogram with only a subset of the data (from 6.5 to 8), with two wider bins on either side (from 6.5 to 7 and from 7.5 to 8), and a set of smaller bins between 7 and 7.5.

There are also a variety of options for controlling the look of the histogram, and these can be combined onto a single plot, as below, which produces Figure 2.4.

Python Code 2.37: Plotting multiple histograms

```

import numpy as np
import pylab as pl
import math

#create a set of normally distributed random numbers
x=np.random.normal(7,1,1000)

#histogram the values with different binning and different styles
pl.hist(x, bins=40, alpha=0.5, color='red', histtype='stepfilled', label='40
    regular bins') #alpha makes it translucent...
pl.hist(x, bins=[6.5,7,7.1,7.2,7.3,7.4,7.5,8], alpha=0.5, color='blue', label='7
    uneven bins')
#Make it pretty:
pl.xlabel("value")
pl.ylabel("frequency of value")
pl.title("Normally distributed variable - different binning")
pl.legend()
pl.grid()

#Set the axis range
pl.axis([3.5,10.5,0,250])
#show the plot:
pl.show()

```

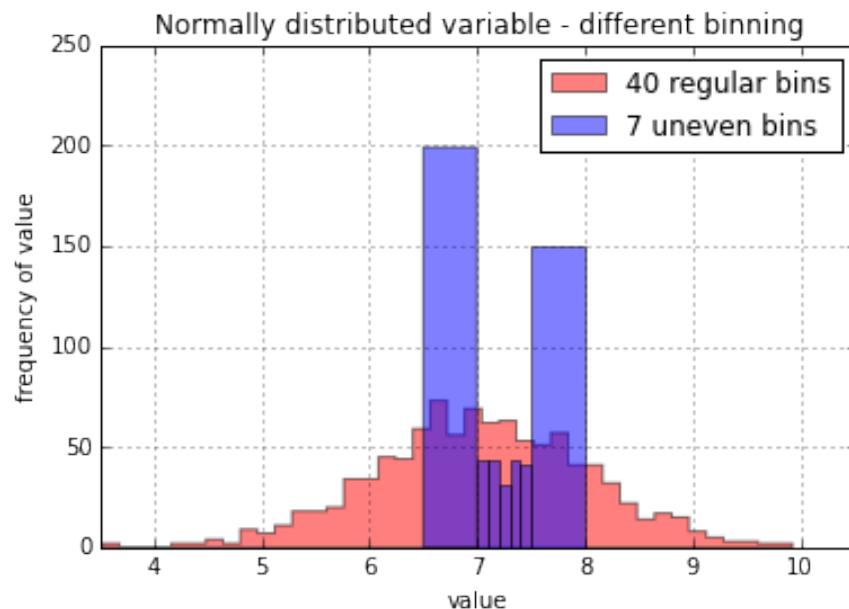
2.37: Output

Figure 2.4: Two different histograms of the same data, with different binning.

We can also normalize the histograms (so that the sum of the bin contents is 1), and we can also plot the “cumulative” histogram (where each bin is the sum of the bin contents that are below it). The cumulative histogram is effectively the integral of a histogram. The code

below illustrates some of the various possibilities for plotting histogram, and also shows how to make “sub-plots” in python (which does not apply only to histograms).

Python Code 2.38: Making a normalized histogram and making sub figures

```
import numpy as np
import pylab as pl
import math

#create a set of normally distributed random numbers
x=np.random.normal(7,1,1000)

#Choose the size of the figure:
pl.figure(figsize=(8,8))

#Create different sub plots:
#Available colours by name are the standard "html colours"
pl.subplot(221)
pl.hist(x, bins=40, color='gray', histtype='step')
pl.title("basic histogram")
pl.xlabel("value")
pl.ylabel("frequency of value")

pl.subplot(222)
pl.hist(x, bins=40, color='burlywood', histtype='stepfilled', normed=True, label='40 bins')
pl.hist(x, bins=10, color='darkslateblue', alpha=0.5, histtype='bar', normed=True, label='10 bins')
pl.title("normalized")
pl.xlabel("value")
pl.ylabel("frequency of value")
pl.legend(loc='upper left', prop={'size':8})

pl.subplot(223)
pl.hist(x, bins=40, color='darkgoldenrod', alpha=0.5, histtype='stepfilled', normed=True, cumulative=True)
pl.title("normalized and cumulative")
pl.xlabel("value")
pl.ylabel("integral of value")

pl.subplot(224)
pl.hist(x, bins=40, color='chocolate', histtype='stepfilled', log=True)
pl.grid()
pl.title("basic — log scale")
pl.xlabel("value")
pl.ylabel("frequency of value")

#show the figure:
pl.tight_layout() #space out the sub plots
pl.show()
```

2.38: Output

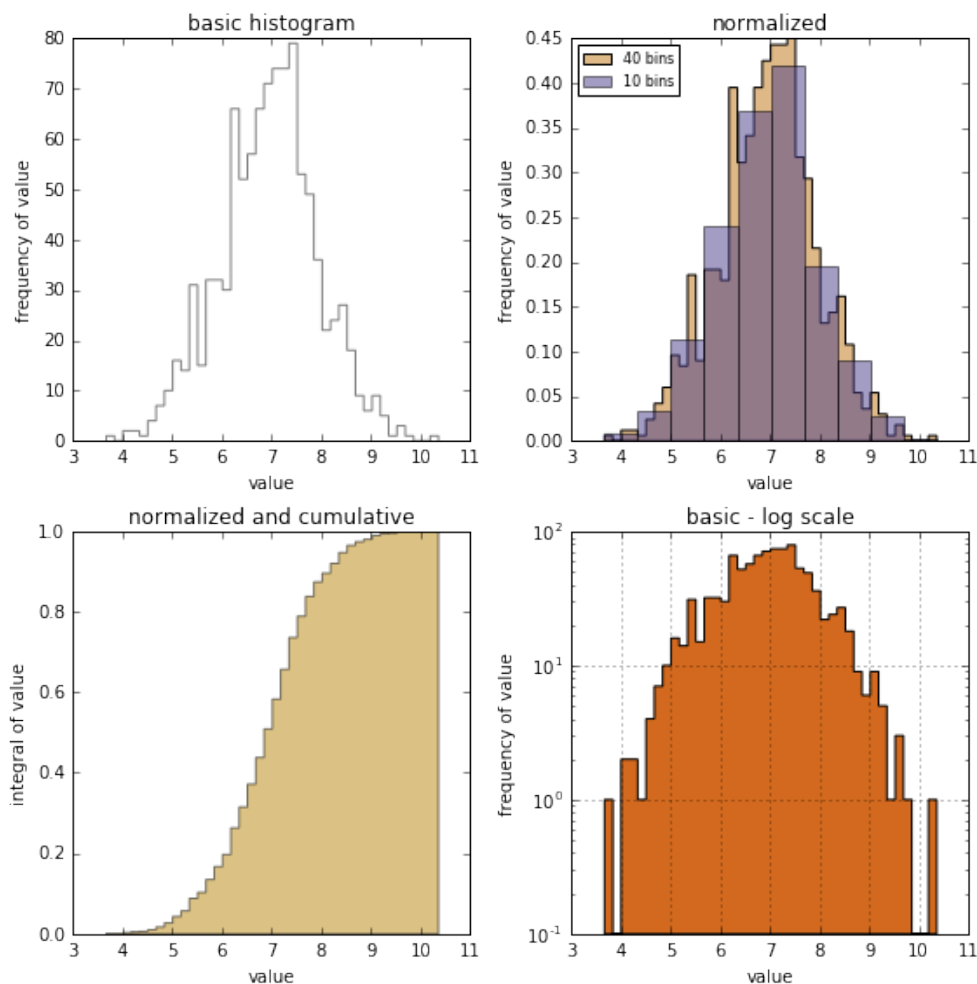


Figure 2.5: Various ways to display a histogram, also showing how to use subplots.

2.6.3 Sympy: Symbolic computations in python

In later chapters, we will find that we often need to take derivatives of quantities. As in other popular software packages, it is also possible to evaluate derivatives symbolically in python. This is done using the **sympy** (“symbolic python”) package.

In order to use symbolic computation, one must first import the sympy package, and then define variable that python will represent as symbols. In this example, we define x and y as symbols, as well as a symbolic function $f(x, y) = \sqrt{x^2 + y^2}$, that depends on x and y :

Python Code 2.39: Sympy for symbolic computations in python

```
import sympy as sym
x, y = sym.symbols('x y') #declare variables x and y to be used as symbols
f = sym.sqrt(x**2+y**2) #a function of x and y
```

The derivatives of f with respect to x and y can be found easily, and manipulated as a symbolic expressions themselves (which we sum below):

Python Code 2.40: Using sympy to compute a symbolic derivative

```
import sympy as sym
sym.init_printing()

x,y = sym.symbols('x y') #declare variables x and y to be used as symbols
#When we use a function, such as sqrt, we should use the version from the
    sympy package:
f = sym.sqrt(x**2+y**2) #a function of x and y
#take the partial derivatives of f
dfdxdx = sym.diff(f,x)
dfdy = sym.diff(f,y)
print("df/dx=",dfdxdx,"df/dy=",dfdy)
#add them together
dSum = dfdxdx+dfdy
print("df/dx+df/dy = ",dSum)
```

2.40: Output

```
df/dx= x/sqrt(x**2 + y**2) df/dy= y/sqrt(x**2 + y**2)
df/dx+df/dy = x/sqrt(x**2 + y**2) + y/sqrt(x**2 + y**2)
```

Finally, we can also substitute in numerical values using the `subs()` command on an expression and passing it an array of tuples. We can numerically evaluate an expression using `N()`:

Python Code 2.41: Substituting numerical values into expressions from sympy

```
#substitute some numbers for x and y into our dSum expression
nValue1 = dSum.subs([(x,2),(y,3)])
nValue2 = dSum.subs([(x,4),(y,5)])
print("value 1: {:.2f}, value 2: {:.2f}".format(sym.N(nValue1),sym.N(nValue2)))
```

2.41: Output

```
value 1: 1.39, value 2: 1.41
```

2.6.4 Curve fitting with scipy optimize

It is often the case that given a set of data points, (x_i, y_i) , we want to “fit” a model function, $y = f(x, \vec{\beta})$, to the data. The goal is to determine the set of parameters $\vec{\beta} = (\beta_1, \beta_2, \dots)$ that best fit the data. For example, a set of data may be described by a second order polynomial, $f(x) = a + bx + cx^2$, and we may wish to use the data to determine the parameters, a , b , and c that best fit the data. This can be done using the `scipy.optimize` module. The first thing to do is to define a python function to represent our model function:

Python Code 2.42: Defining a function to use for fitting

```
import numpy as np
def model(x,*pars):
    return pars[0]+pars[1]*x+pars[2]*x*x
```

where you should note that the dependent variable, x , is the first argument of the model function, whereas we passed the parameters as a list, `*pars`, after the dependent variable.

Let us assume that our data is stored into three one-dimensional numpy arrays, `xdata`,

`ydata`, and `ydataErr`, holding the x and y coordinates of the points, and the uncertainties on the y values, respectively. The algorithm that varies the model parameters can have some difficulties in finding good fit values for the fit parameters, especially if the fit function is complicated. It is thus often necessary to give the fitter an initial guess for the parameters. The fitter is run with the following code:

Python Code 2.43: Fitting a function to a set of xy data with error bars

```
from scipy.optimize import curve_fit
#Guesses for the parameters
pars_guess = [5,50,-3]
pars_fit , pcov = curve_fit(model, xdata , ydata ,sigma=ydataErr ,p0=pars_guess)
```

where we imported only the `curve_fit()` function from the `scipy.optimize` module. The `curve_fit()` function returns a tuple, `pars_fit,pcov`, where `pars_fit` are the best fit values for the parameters that we want and `pcov` is the covariance matrix of the parameters. The square root of the diagonal elements of the covariance matrix correspond to the errors on the parameters from the fit, which we can easily get:

Python Code 2.44: Obtaining the errors from the fit

```
#Get the errors out of the covariance matrix
pars_err = np.sqrt(np.diag(pcov))
```

Below is a full example that creates “fake data”, fits those data to the model, and then makes a plot showing the fitted function, the data, and the residuals from the fit (see Chapter 8, the residuals are the difference between the data and the model evaluated with the best fit parameters).

Python Code 2.45: A complete fit including residuals

```
import numpy as np
import pylab as pl
import matplotlib.gridspec as gridspec # for unequal plot boxes
from math import *
from scipy.optimize import curve_fit #for fitting

#Define our model function , that depends on x and the parameters that we want
to determine
def model(x,*pars):
    return pars[0]+pars[1]*x+pars[2]*x*x

#####
#Create the fake data#
#####
#True values for the parameters , that we use to generate the data:
pars_true=[5,50,-3]

#generate x-values of the data:
ndata = 50 #number of data points
xdata = np.linspace(0,10,ndata)

#generate y coordinates that are normally distributed about the model
```

```

    evaluated at xdata
#with a sigma of yerr
yerr=5*np.ones(xdata.size)#5 #the uncertainty on y values
ydata = np.random.normal(model(xdata,*pars_true),yerr)

#Guesses for the parameters (use the true values here)
pars_guess = pars_true

#####
#Run the fit:#
#####
pars_fit , pcov = curve_fit(model, xdata, ydata ,sigma=yerr ,p0=pars_guess)

#Get the errors out of the covariance matrix (sqrt of the diagonal elements)
pars_err = np.sqrt(np.diag(pcov))

#Create some text with the fit results to put into our plot
resultTxt = '''Fitted parameters for:
y=a+bx+cx^2:\n'''
parNames = ["a","b","c"]
for i in range(pars_fit.size):
    resultTxt = resultTxt+"{:s}: {:.2f} +/- {:.2f}\n".format(parNames[i] ,
    pars_fit[i] , pars_err[i])

#####
#Plot the data with error bars and the result of the fit#
#Also include a subplot with the residuals #
#####

#Generage a curve from the model and the fitted parameters
yfit = model(xdata,*pars_fit)
#Generate a set of residuals for the fit
yres = ydata-yfit

#Set the size of the figure
pl.figure(figsize=(8,8))
#Divide the figure into 2 rows, with 1 row 3 times higher for the data
gs = gridspec.GridSpec(2, 1, height_ratios=[3, 1])
pl.subplot(gs[0])
#Plot the data with error bars
pl.errorbar(xdata,ydata,yerr=yerr,fmt='o',label='data',color='black')
#Plot the fit line
pl.plot(xdata,yfit,'r',label='fit',lw=3)
#Set the axes range to be pretty:
pl.axis([xdata.min()-1,xdata.max()+1,ydata.min()-yerr[0]-1,1.1*ydata.max()])
pl.legend(loc='upper left')
#Placement of the textbox may not be ideal:
pl.text(xdata.mean(),ydata.min(),resultTxt,fontsize=14)
pl.title("Fit of the data")
pl.xlabel('x')
pl.ylabel('y')
#Use the bottom row of the figure for the residuals:
pl.subplot(gs[1])
pl.errorbar(xdata,yres,yerr=yerr,fmt='o',color='black')#residuals

```

```

pl.ylabel('residuals')
pl.xlabel('x')
pl.axis([xdata.min()-1,xdata.max()+1,-5*yerr.max(),5*yerr.max()])
pl.tight_layout()
pl.show()

```

The result is shown in Figure 2.6.

2.45: Output

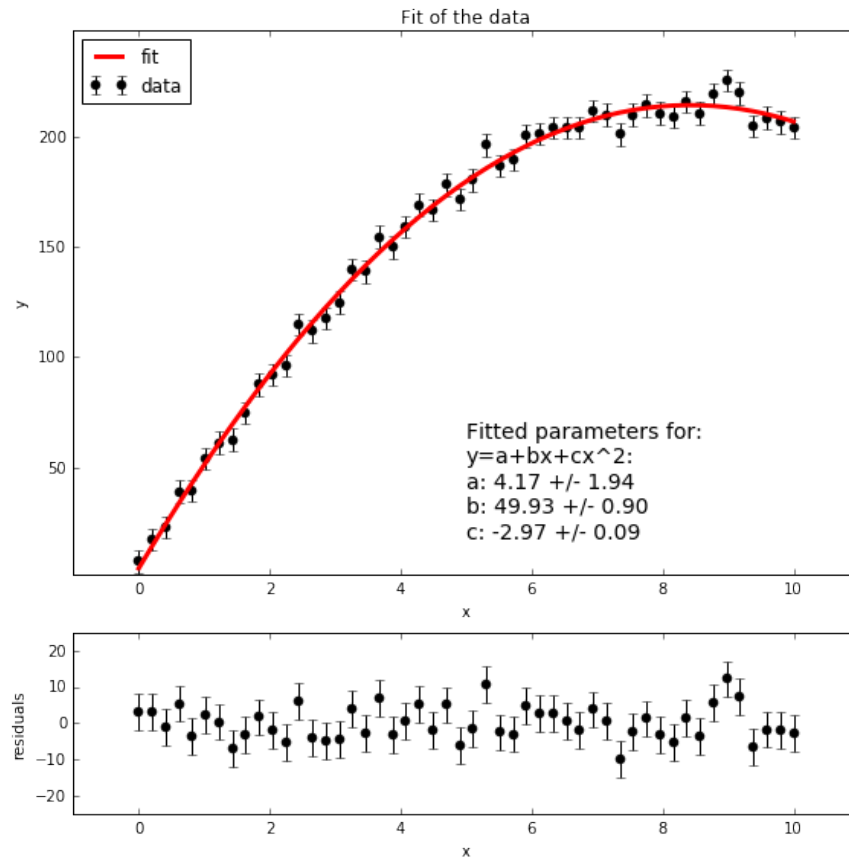


Figure 2.6: Example of a fit to a second order polynomial and the residual plot.

3

Determining uncertainties and presenting data

In this chapter we will look at how to determine uncertainties on measurements, how to present those uncertainties, what they mean, and how to determine if two values agree. We will give some basic guidelines that are of use for undergraduate laboratories, and justify most of those guidelines in later chapters.

3.1 The importance of uncertainties

The use of uncertainties is fundamental in applying the Scientific Method, where one tries to invalidate a hypothesis by using data. Generally, we can think of the need to compare two numbers that cannot be determined to infinite precision (e.g. a measurement and a theoretical prediction). We thus need a way to scientifically determine whether two quantities are “consistent” with each other, when they are not exactly equal. We also need to understand how to present a number; if our “value determined from experiment” involves an operation that leads to many digits (e.g. multiplying by an irrational number), we will have to decide how many digits are really “significant” before performing a comparison.

You may find that you can measure a number to several decimals with an accurate ruler, but then are unable to reproduce that measurement to the same accuracy. Even if the ruler can measure something to a certain number of decimals, the uncertainty in the measurement may not be representative of that precision. For example, consider using a micrometer (which can measure distances of order 10 cm to sub-millimetre accuracy) to measure your height; you will inevitably introduce errors when you try to stack the micrometers atop each other to reach your height. Intuitively, the uncertainty in your height should be larger than the precision of the micrometer, even though that is the only instrument that was used.

The need to assign uncertainties to numbers also arises when we try to build things. If you are cutting pieces of wood to build furniture, you will need them to fit together. You will find that you cannot cut them to be exactly the length that you determined in your drawings (even when you take into account the width of the saw blade, use a good square, work very carefully, etc.). You will find that you need to determine the “tolerance” in all of your required dimensions for the parts to fit. If you go to a professional machine shop to

have parts manufactured, the machinists will inevitably ask you what tolerance you have on the pieces; it is usually not their job to assemble your parts for you. This issue becomes one that is quite important in designing parts; it may be relatively easy to design the function of a part compared to determining how much tolerance is required for the parts to fit. Although the issue of tolerance is related to uncertainties, in this book, we will focus more on the aspect of comparing two numbers, but do keep in mind that the concepts usually apply to tolerances when designing parts.

3.1.1 What do we mean by “uncertainty”?

You have likely already encountered uncertainties in your labs, but you may not really have thought them through. A typical experiment could be to build a simple balance, where two unequal weights are placed on either side of a lever arm. We may want to determine where to place the weights to equilibrate the balance. We will need to measure the masses of the weights, and if the digital scale displays numbers down to a gram, we would likely quote the weight of one of the masses as, say, $m_1 = (595.0 \pm 0.5) \text{ g}$. We should ask ourselves what we really mean by $(595.0 \pm 0.5) \text{ g}$? Do we mean that the mass of the weight is *guaranteed* to be between 594.5 g and 595.5 g? Does it mean we would bet our lives on it? What if we later discovered that the scale reads all weights to be 5 g too light, can we guard ourselves against that (since we are apparently willing to bet our lives on our measurements)?

The basic answer to these questions is that we cannot just quote the measured value to be $m_1 = (595.0 \pm 0.5) \text{ g}$! It is *on us*, the scientists, to explain what we mean by $m_1 = (595.0 \pm 0.5) \text{ g}$. If we mean that we just assigned an uncertainty, without much thought, of 0.5 g because that is half of the smallest digit on the scale, then we have to say that. If we mean that we would bet our lives that the true value is between 594.5 g and 595.5 g, then we should say that. If we mean that we are 90% confident that the true mass lies between 594.5 g and 595.5 g, then we should say that (but then, what do we mean by 90% confident?). If we are not confident that the scale reads “true” (e.g. because we did not repeat our measurement with a different scale, or do not know if the scale was calibrated), then we should mention that as well. Again, we see that doing good science is about being precise. Furthermore, doing good experimental physics, is all about *understanding* uncertainties.

You may have also seen results from opinion polls that try to survey the general population to predict election results. These polls can be quite important in how politicians plan their campaigns (read: where to spend money), and the polls may be quite expensive to conduct, so they typically are motivated to get scientific data. You may, for example, see the following reported in a news article: “Candidate A is leading, favoured by 52% of voters polled, whereas Candidate B has fallen behind and is favoured by only 48% of voters”. At the end of the article, there will typically be a statement of the form: “The poll was conducted by BMIP by calling 1000 registered voters and conducting a 10 question survey by phone. The margin of error on the polling numbers are 4%”.

We can ask ourselves what they mean by the “margin of error”. It certainly makes sense that they cannot get an exact number for the whole electorate by surveying only 1000 people, as there will necessarily be statistical fluctuations in the sample that they surveyed. We

can ask ourselves if the difference between the two candidates really is “significant”. If the results really are $(52 \pm 4)\%$ and $(48 \pm 4)\%$, there is substantial overlap between the possible results and a statistical fluctuation cannot be excluded. Do they mean that for Candidate A, the result is definitely between 48% and 56% with an equal probability over that whole range? How did they determine the 4% uncertainty? Why did they ask 10 questions when they only needed one?

Although we will not spend any time on designing opinion polls, we will spend a substantial amount of time trying to understand the uncertainties in statistical quantities. As you can see, a strong understanding of statistical uncertainties is not only useful in physics, but carries out to a wide range of applications in real life, where we are often confronted with dubious conclusions from badly analysed data.

All this being said, when we quote a number as $A \pm \sigma_A$, we imply that A is quite likely to be in the range $A - \sigma_A$ to $A + \sigma_A$ (and we typically specify what we mean by “quite likely”). Generally, we also imply that it is **not** more likely for A to be closer to one of the end points of the range than the other. We always quote the uncertainty, σ_A , as a positive number. We call the quoted value of A the “best estimate” or “central value”, and σ_A as the “uncertainty” or “error”.

3.1.2 Systematic uncertainties

Systematic uncertainties are different from what we would call “statistical” or “random” uncertainties. By **random uncertainty**, we mean that a repeated measurement will give variable results that are **equally likely to be higher or lower than the true value**. With a **systematic error**, we mean that **repeated measurements will give systematically higher or lower values than the true value** (hence the name). The advantage of a random error is that if we make many measurements, we can expect that the average of those measurements will tend to the true value. There are no advantages to systematic errors! They are often difficult to detect and the design of most experiments is often driven by the need to reduce systematic uncertainties, ideally to the level that they are negligible.

For example, imagine that several laboratory groups are measuring weights for an experiment. They all have their own scales, and as good experimentalists, they borrow each other’s scales to check their measurements and make sure that their own scale is calibrated. However, all of the scales were calibrated by the TA before the lab, and the TA used the wrong “standard weight” when calibrating all of the scales. This is an example of a systematic uncertainty in all of the mass measurements in the lab that is very difficult to catch. Or maybe the scales are more accurate for weights that are smaller than 100 g, and the various lab groups have used weight well below and well above 100 g; some groups may be affected by the systematic uncertainty, while others not.

Generally, it is not possible to arrive at a prescription of how to determine the systematic uncertainty in a measurement. It is however very important to think very carefully about all possible sources of systematic uncertainties. Ideally, this is done before designing an experiment so that the design of the experiment minimizes the associated systematics. One

can then typically estimate the magnitude of the systematic uncertainties and conduct the experiment with the knowledge that the most precise result that they can obtain will have an uncertainty at least as big as the systematic uncertainty.

Experiments that are currently searching for dark matter are a good example of how the minimization of systematic uncertainties are critical in the design of the experiment. These experiments require building large sensitive detectors to look for small flashes of light resulting from dark matter particles interacting in the detector. The problem is that particles emitted from naturally occurring radioactive decays in the materials used to construct the experiments will also produce flashes of light. Therefore, those experiments can only be sensitive to dark matter if the rate of the flashes of light from dark matter is substantially higher than the rate from “backgrounds”. Much of the effort in designing these experiments is thus spent in minimizing the backgrounds (e.g. by material selection) so that the sensitivity to dark matter is as high as possible, and not “systematically” limited by a high background rate. The experiments are then conducted until the sensitivity to dark matter becomes limited by the background rate.

3.2 Prescription for determining certain uncertainties

In certain situations, one can use fairly well-prescribed methods to specify the uncertainty in a given measurement. We cover those particular cases in this section.

3.2.1 Uncertainties from scales - “Half of the smallest division”

We first consider the uncertainty in measurements made with a device that has some sort of scale on it (a ruler, a digital scale, a needle on a dial, etc.). We assume that the measuring device is well calibrated, but in a real application, we should estimate whether we also need to include a systematic uncertainty from the calibration.

If we have a ruler that has mm graduations on it, we should expect that we can measure a distance to within 1 mm. In a typical scenario, we will use the ruler and find that the true value is between two lines on the ruler (Figure 3.1). In this case, we would typically quote the best estimate of the measurement to correspond to the line on the ruler **that is closest to the distance being measured, and an uncertainty that is half of the smallest division on the ruler**. Figure 3.1 shows an example of measuring something with a ruler. Using our prescription, we would quote the grey object to have length of (2.80 ± 0.05) cm. That is, we are reasonably confident that the true length is between 2.75 cm and 2.85 cm, which is a range of 1 mm, corresponding to the precision of the ruler.

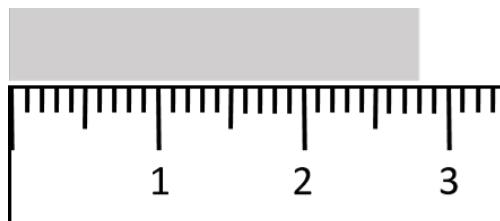


Figure 3.1: Measuring with a ruler.

This general rule (“half of the smallest division”) can be used for all measurements devices where numbers are read off a scale, including digital scales. For example, a digital voltmeter may read 12.1 V (0.1 V being the smallest displayed digit) and we would then quote the measurement as (12.10 ± 0.05) V. Sometimes, it is clear that we can provide a measurement with a higher accuracy than half of the smallest division (if the divisions are far apart), but we should only do so cautiously. Even using half of the smallest division may significantly underestimate the uncertainty (for example, we may not be able to get the ruler right up against what we are trying to measure, or what we are trying to measure has a finite width that makes it unclear from which point to measure, or the needle on the dial is fluctuating in position, or the digits on the voltmeter are fluctuating). Often, we can get the best idea of whether our uncertainty is reasonable by repeating the measurement, and we should always do so if we can.

With this type of uncertainty, we are usually almost 100% confident that the true value lies in the quoted range. However, since it is very difficult to guarantee that there was no systematic effect (e.g. parallax, non-calibrated instrument, clearly defined what we need to measure), it is not a good idea to treat even these types of measurements as giving us absolute confidence in the quoted uncertainty range. It is generally safer to treat even this type of uncertainty with a confidence around the typical 68%, unless repeated measurements and control of systematic effects give us reason to be more confident.

3.2.2 Repeatable measurements - “Mean and uncertainty on the mean”

When we have the opportunity to repeat measurements, we should always do so. If we have several measurements of a single quantity, then how do we define the central value and the uncertainty? For example, we may be using a stop watch to measure the period of a pendulum, and obtained several values, say $T = \{2.1 \text{ s}, 2.2 \text{ s}, 2.1 \text{ s}, 2.0 \text{ s}, 2.3 \text{ s}, 2.1 \text{ s}\}$, so what should we quote as our measured value? It is likely that the true value is somewhere between 2.0 s and 2.3 s, so we could quote the result as (2.15 ± 0.15) s (given by the central value of the range with an uncertainty that spans the whole range). In later chapters, we will motivate a different prescription, but we take it here as given:

When we have a set of N independent measurements, $\{x_1, x_2, x_3, \dots, x_n\}$ of a quantity, x , the quoted result should be given as $\bar{x} \pm \frac{\sigma_x}{\sqrt{N}}$, where

$$\begin{aligned}\bar{x} &\equiv \frac{1}{N} \sum x_i \\ \sigma_x &\equiv \sqrt{\frac{1}{N-1} \sum (x_i - \bar{x})^2} \\ \sigma_{\bar{x}} &\equiv \frac{\sigma_x}{\sqrt{N}}\end{aligned}\tag{3.1}$$

\bar{x} is the algebraic average (or “mean”) of the values, and σ_x is called the “standard deviation” of the measurements (and requires the mean to be known). The standard deviation is a measure of the average distance between the individual measurements and the mean. It is

also representative of the uncertainty on a single measurement. A large standard deviation means that the measurements are spread out over a larger range about the mean.

$\sigma_{\bar{x}}$ is called the “error on the mean”, and corresponds to the uncertainty in determining the mean of the measurements, which is smaller than the standard deviation (since it is the standard deviation divided by \sqrt{N}). It makes sense for the error on the mean to be smaller than that of the individual measurements because it combines the information from multiple measurements. Again, here we assumed that there is no systematic uncertainty that skewed all of the measurements and that the measurements are independent of each other.

The uncertainty that we obtain using the error on the mean does not guarantee that the true value is within the quoted range. In fact, it is carefully designed (under most circumstances) so that there is a 68% chance that the true value lies within the specified range. We will explain why we use 68% in chapter 7.

Example 3-1: What is the central value and uncertainty on T , if the following measurements were performed: $T=\{2.1\text{ s}, 2.2\text{ s}, 2.1\text{ s}, 2.0\text{ s}, 2.3\text{ s}, 2.1\text{ s}\}$?

This is done easily in python

Python Code 3.1: Mean and std with numpy

```
import numpy as np
from math import *

#load the measurements into a numpy array
Ti = np.array([2.1, 2.2, 2.1, 2.0, 2.3, 2.1])

#numpy already knows how to find the mean and standard deviation:
Tavg = Ti.mean()
Tstd = Ti.std(ddof=1) # this results in the N-1 in the denominator
                        instead of N
Terr = Tstd/sqrt(Ti.size)
print("Mean = {:.2f}, Standard deviation = {:.2f}, Error on the mean = {:.2f}").format(Tavg, Tstd, Terr)
print("T = ", "{:.2f} +/- {:.2f} s".format(Tavg, Terr))
```

3.1: Output

```
Mean = 2.13, Standard deviation = 0.10, Error on the mean = 0.04
T = 2.13 +/- 0.04 s
```

Or, using QExpy:

Python Code 3.2: Mean and std with QExpy

```
import qexpy as q
T = q.Measurement([2.1, 2.2, 2.1, 2.0, 2.3, 2.1])
q.set_sigfigs(2)
print("T = ", "{:.2f} +/- {:.2f} s".format(T.mean, T.error_on_mean))
```

3.2: Output

$$T = 2.13 \pm 0.04 \text{ s}$$

3.2.3 Counting experiments - The square root rule

In some experiments, the quantity that we measure is something that we can count. For example, we may want to figure out the average rate of car accidents per month in our town. We may, for example, count that in January, there were 8 accidents. That is a definite number without an obvious uncertainty. However, we would likely be wrong if we claimed that every month, there will be exactly 8 accidents. So how do we translate our measurement of 8 counts into a more scientific statement, such as “*In our town, we determined the average rate of accidents per month to be $8 \pm X$* ”?

It turns out that the answer is rather simple: **if the events occur at random times but with a well defined average rate, the uncertainty on the number of counts, N , is the square root of the number of counts, \sqrt{N} .** In our example, we would conclude that the rate of accidents per month is 8 ± 2.8 . The square root uncertainty is also constructed such that the true value has approximately a 68% chance of occurring in the quoted range. The value is close to 68% when N is large (bigger than about 75), and reduces as N becomes smaller. We will examine this in more detail in Chapter 6, when we consider the Poisson distribution.

3.2.4 Measuring a fraction/proportion - The binomial error on a ratio

In some cases, one needs to measure the fraction of a number of outcomes. For example, one may be interested to know which fraction, p , of a number of particle collisions result in the creation of a Higgs Boson. Or what fraction of people wear red socks, or what fraction of camelids are vicuñas. In all these cases, the fraction is defined as the ratio of the number of outcomes of interest, k , to the total number of outcomes, N :

$$p = \frac{k}{N} \quad (3.2)$$

In such a situation, it is important to note that k and N are not independent; that is, if there are more people in the room (N), there will also be more people wearing red socks (k), but their ratio, p , is assumed to be constant. When this is the case, the uncertainty in p is given by:

$$\sigma_p = \sqrt{\frac{p(1-p)}{N}} \quad (3.3)$$

This will be justified in Chapter 6, when we consider the binomial distribution.

3.3 Overestimating uncertainties is also bad!

With all of these caveats in determining the uncertainties on measurements, we may be tempted to just round up all of our uncertainties to be on the safe side. That way, we will have a good chance of concluding in our lab report that the results agree with the laws of

physics! In principle, for an undergraduate laboratory, there will not be much consequences in doing this (the labs are not really expected to revolutionize our understanding of physics). Hopefully, there is no need to convince you that this is not a very scientific attitude, and you certainly should not think like this! Similarly, it should be obvious that if we really are testing the laws of physics (with the hope of discovering something new), then we really want to have the most precise result possible, with the smallest, but justifiable, uncertainties.

If we understand our uncertainties and have defined them precisely, then we would only be tempted to inflate them after the fact, when we see that our results disagrees with the hypothesis that we are testing (or do not lead to the expected conclusion). If our uncertainties are well defined and our result does not agree with the hypothesis, then we must have missed a systematic effect. In fact, we can tell from the disagreement between our result and the hypothesis how big that systematic effect must be. This is in fact an opportunity to think through the experiment and try to identify where the systematic error arose (or find the mistake in our calculations).

This approach of “discovering a systematic” uncertainty can apply in undergraduate laboratories, where we are highly confident that we will not find an inconsistency with the laws of physics that we are testing (and thus that we can reasonably expect a certain result). However, in “real” experimental physics, one does not have a hypothesis that can be taken as “definitely correct” (or rather, “tested to a much higher degree of accuracy by someone else”) and this is why systematic uncertainties are particularly dangerous, since there is no guaranteed way to find them. This is also why, in general, we want to make sure that our uncertainties are not overestimated, so that others can verify our results as accurately as possible. Please become a good scientist and do not overestimate your uncertainties!

3.4 Reporting measurements, significant figures

In the above example (Example 3-1), we asked python to format the output to 2 decimal places. By default, python would have output the result as $T = (2.1333333333 \pm 0.0461880215352) \text{ s}$. In this case, it would not make sense to present that many decimal places on the central value, when the uncertainty is much bigger than most of them. Similarly, it does not make sense to claim that we have determined the uncertainty that precisely either. We should quote results and uncertainties showing only “significant” digits, and in a way that is the most legible (for example by using scientific notation).

Again, we do not have a perfect rule for determining the significance of digits, but in general, **the uncertainty should have 1 (and rarely more) “significant figures”**. By significant figure, we mean digits after leading zeros or before trailing zeros. For example, in the number 00019.21000, only 19.21 are “significant”, and we say that the measurement is determined to four significant figures. Once the number of significant figures is determined on the uncertainty, one can quote the central value with the same number of figures. When we quote a number, we should ask ourselves if we can justify all of the decimals, or if they are instead arbitrary (maybe from a calculation or from a random fluctuation in the apparatus); if we do not think that we can reproduce a measurement to all of the significant figures that

we quoted, then we should consider them as non-significant and not present all of them.

Usually, we are interested in evaluating the precision of a result; that is, how big the uncertainty is relative to the central value. The “relative” uncertainty (also called the “fractional uncertainty”) is usually a good measure of the precision, and should generally be quoted with a measurement. **The relative uncertainty is quoted as a percent and is given by the uncertainty divided by the central value.**

In order to help our audience appreciate the precision of our measurement even without giving the relative uncertainty, we should present the number using scientific notation and factoring out the units in a way that it is easy to estimate the relative uncertainty. For example, a measurement of $m=12000\text{ g} \pm 23\text{ g}$, could be presented as $m=(120.00 \pm 0.23) \times 10^2\text{ g}$; that way, it is quite obvious that the relative uncertainty is of order 0.2%.

Generally, if the first significant digit in the uncertainty is small (say, less than 5) and the second digit is large (say, more than 5), then it may make sense to include a second digit since that digit results in an “appreciable change” in the uncertainty. For example if an uncertainty is determined to be 0.15, then it might make sense to quote it as 0.15 instead of 0.2, since the additional decimal changes the uncertainty by 25% (an arguably appreciable change). Conversely, if the uncertainty is determined to be 0.92, then it might as well be quoted as 0.9, since the 0.02 results in a small relative change in the uncertainty. In addition to considering the relative change in the uncertainty itself when deciding whether to include an additional digit, we can consider the change in the relative uncertainty of the measurement. For example, if a measurement is determined to be 10001.12 ± 0.15 (a relative uncertainty of $\frac{0.15}{10001.12} = 0.001\%$), then quoting it as 10001.0 ± 0.2 probably does not make a significant difference in our conclusion. So what is an “appreciable relative change”? That is of course arbitrary, but for high precision measurements anything above a 1% change may be considered appreciable, while in undergraduate laboratories, that threshold could be closer to 5%.

Example 3-2: Present the following results using appropriate significant figures and give the relative uncertainties on the results: 2.1124 ± 0.156 , 100000 ± 3900 , $0.000084341387 \pm 0.000000322331$, 0.000091 ± 0.1

- $2.1124 \pm 0.156 \rightarrow 2.11 \pm 0.16$ (8% relative) - Include a second digit in the uncertainty since it is an appreciable change in both the uncertainty and the relative uncertainty
- $100000 \pm 3900 \rightarrow (100 \pm 4) \times 10^3$ (4% relative) - Change to scientific notation to easily see the size of the relative uncertainty, only use 1 digit on the uncertainty since changing between 3.9% and 4% relative uncertainty is not significant.
- $0.000084341387 \pm 0.000000322331 \rightarrow (84.3 \pm 0.3) \times 10^{-6}$ (0.4% relative) - Scientific notation to make it more legible, only show 1 digit for the uncertainty since the

relative uncertainty is already very small.

- $0.000091 \pm 0.1 \rightarrow 0.0 \pm 0.1$ (*relative not defined*) - The uncertainty being so much bigger than the value, we cannot distinguish it from zero, so we might as well quote it as zero. With a zero central value, we cannot determine the relative uncertainty.

3.5 Comparing measured quantities

Now that we have some grasp on uncertainties, or at least an understanding that uncertainties are not easy to define accurately, how do we come to conclusions based on measured values? How do we conclude if two numbers with uncertainties agree? Again, we have to remember that we are scientists and that we work within the Scientific Method. We really need to think in terms of “does a measurement exclude a hypothesis?”. When comparing two measurements, we must then ask ourselves if one measurement is “incompatible” with the other one.

Suppose that two teams have each measured the mass of a dark matter particle. Team 1 measured the mass to be $m_1 = (10.4 \pm 1)$ GeV and Team 2 measured the mass to be $m_2 = (12.5 \pm 1)$ GeV. Do the results agree, or should we conclude that there may in fact exist two different dark matter particles? Well, it depends! What is represented by the uncertainties? If both teams are betting their lives that the mass is definitely included within the range that they quote, then the measurements disagree (since the highest compatible value with Team 1 is 11.4 and the lowest value compatible with Team 2 is 11.5). In general however, results are reported with some “degree of confidence” that the true value lies within the specified range. Often, that degree of confidence is around 68% (which we will motivate in chapter 7). If the two teams are both quoting their result with a 68% confidence that the true value lies in their quoted range, there is some room for the value to be outside of the respective ranges and for the two measurements to agree (and indeed, we would conclude that the two measurements, at this level of precision, are consistent with each other).

We can also use the relative difference between the measurements and compare that to the relative uncertainty on the measurements to get an idea of the agreement in terms of the precision. Team 1 has a relative uncertainty on their measurement of $\frac{1.0}{10.4} = 9.6\% \sim 10\%$, and Team 2 has a relative uncertainty in their measurement of $\frac{1.0}{12.5} = 8\%$; the two measurements thus have similar precision, around 10%. The “relative difference” between the central value of the two measurements is $\frac{12.5-10.4}{12.5} = 16.8\% \sim 17\%$. In this analysis, given that the relative difference between the measurements (17%) is of the same order as the quadrature sum of the individual relative uncertainties ($\sqrt{(10\%)^2 + (8\%)^2} \approx 13\%$), one would not conclude that the measurements are incompatible, even if the ranges covered by the uncertainties do not quite overlap.

As a scientific community, we would want to understand if both teams have correctly represented their uncertainties, we would want to see the experiments repeated to be more precise, and if the difference subsists, we would first assume that there is a systematic uncertainty in at least one of the experiments that is shifting their value one way. Only after

all of those issues have been addressed, and if the difference between the two measurements was many times the uncertainty between them, we would consider accepting that there may be two different dark matter particles.

Again, we find that we have to be very precise when comparing results, as the conclusions that we draw are dependent on how the uncertainties are determined. Even worse, the standard that we apply to make a conclusion may well depend on how important that conclusion is. Let's say that instead of measuring the mass of a dark matter particle, Team 1 simply claims that they have finally discovered dark matter (which would certainly earn them a Nobel Prize). In their results, they report that they expected 5 ± 2 background events, and instead observed 11 events. They claim that the excess of 6 events over the expected background of 5 is a clear indication of dark matter events, as it is larger than the expected background by 3 times the uncertainty on the background rate. Indeed, a measurement of 11 is not very consistent with 5 ± 2 , so the hypothesis that they observed an unusually large fluctuation in the background is difficult to support. We also assume that the community has thoroughly analysed their background estimate and agrees that 5 ± 2 is reasonable. As we will see in chapter 7, there is approximately a 0.3% chance that the background actually did fluctuate by 3 times its uncertainty. Since we have to significantly change the laws of physics if the measurement is correct, we may not be satisfied with a 0.3% chance that the conclusion of dark matter existing is wrong. And indeed, in particle physics, the standard is often that a measurement must disagree with the null hypothesis by 5 times its uncertainty in order to invalidate the null hypothesis (in this case, the null hypothesis is that Team 1 only observed background events).

3.6 Summary

Hopefully this chapter will have convinced you of the following points, which you should typically think about when discussing your results:

- There is no perfect rule to determine the size of uncertainties in a measurement; the goal is to make sure that the uncertainties are justified.
- Overestimating uncertainties is just as bad as underestimating them. If you think your uncertainties are too small, understand why and look for systematic effects.
- If possible, you should perform independent measurements to gauge whether your uncertainties are well estimated.
- It is absolutely critical to present results precisely (explain how uncertainties were determined), otherwise it is impossible to use the results.
- Systematic errors are the plague of experimental physics and the design of experiments should focus on minimizing them.
- Random errors can be minimized by repeating measurements (but some measurements may not be easy to repeat).
- When you quote a value with uncertainty, $x \pm \sigma_x$, you should mean that it is equally likely for the true value to be on either side of x .
- Comparing the relative difference between values to the relative uncertainties on the values (or their sum in quadrature) gives a good idea of whether the values agree.

- The ultimate comparison between numbers with uncertainties involves some degree of subjectivity and depends on: what the uncertainties are defined to be, whether systematic errors have been considered, and what standard needs to be met for a hypothesis to be rejected.

With the caveat that you need to consider systematic uncertainties and that this may underestimate your uncertainty, it is usually safe to use half of the smallest division from an instrument as the uncertainty on a direct measurement from the instrument.

We saw that for a set of N independent measurements, $\{x_1, x_2, x_3, \dots, x_n\}$ of a quantity, x , the quoted result should be given as $\bar{x} \pm \frac{\sigma_{\bar{x}}}{\sqrt{N}}$, where

$$\begin{aligned}\bar{x} &\equiv \frac{1}{N} \sum x_i \\ \sigma_x &\equiv \sqrt{\frac{1}{N-1} \sum (x_i - \bar{x})^2} \\ \sigma_{\bar{x}} &\equiv \frac{\sigma_x}{\sqrt{N}}\end{aligned}\tag{3.4}$$

σ_x is called the standard deviation of the measurements and is representative of the uncertainty on a single measurement. \bar{x} is the mean of the measurements, and $\frac{\sigma_x}{\sqrt{N}}$ is the uncertainty on the mean. If the measurements are independent and there are no systematic effects, there is a 68% chance that the true value lies within the specified range.

In a counting experiment, where events occur at random times but with a well defined average rate, the uncertainty on the number that you counted, N , is simply the square root, \sqrt{N} . This uncertainty is constructed so that it is the standard deviation that you would get if you measured N many times. If N is large (say bigger than about 80), the range given by $N \pm \sqrt{N}$ will have the true value in it 68% of the time, if N is small, that percentage will be somewhat reduced.

4

Error Propagation

In this chapter, we show how to propagate the uncertainties from “direct” measurements to uncertainties in quantities calculated from those measurements. We will justify many of the rules that we present here in the following chapters.

4.1 The Min-Max method

We will look at the “Min-Max” method as an introduction to propagating uncertainties, although it is only typically used in first year introductory physics labs. The Min-Max method is based on considering the maximal change in a calculated quantity that can result from the variation of the measured quantities.

4.1.1 Summing and subtracting

First, we consider the case of summing (or subtracting) two numbers with uncertainties. For example, we may be placing two masses (m_1, m_2) on a balance and need to know their sum, $M = m_1 + m_2$. If each individual mass, m_i , has an uncertainty of σ_{m_i} , then the absolute maximal value that the sum of the masses could have is:

$$M^{max} \equiv (m_1 + \sigma_{m_1}) + (m_2 + \sigma_{m_2}) \quad (4.1)$$

which assumes that in each case, the “true” value of the masses was at the maximum end of the range defined by our uncertainty. Similarly, if the “true” value of the masses was at the minimum end of the range, the minimum value that the sum could be is:

$$M^{min} \equiv (m_1 - \sigma_{m_1}) + (m_2 - \sigma_{m_2}) \quad (4.2)$$

Now that we know the range over which M must lie, we can define a value of M with uncertainty, σ_M , to correspond to the center of the range with an uncertainty that allows the range to be covered:

$$\begin{aligned} M &\equiv \frac{1}{2}(M^{max} + M^{min}) \\ \sigma_M &\equiv \frac{1}{2}(M^{max} - M^{min}) \end{aligned} \quad (4.3)$$

Substituting the expressions for M^{max} and M^{min} , we have:

$$\begin{aligned}
 M &= \frac{1}{2} ([(m_1 + \sigma_{m_1}) + (m_2 + \sigma_{m_2})] + [(m_1 - \sigma_{m_1}) + (m_2 - \sigma_{m_2})]) \\
 &= m_1 + m_2 \\
 \sigma_M &= \frac{1}{2} ([(m_1 + \sigma_{m_1}) + (m_2 + \sigma_{m_2})] - [(m_1 - \sigma_{m_1}) + (m_2 - \sigma_{m_2})]) \\
 &= \sigma_{m_1} + \sigma_{m_2}
 \end{aligned} \tag{4.4}$$

The central value of the calculated quantity, M , is thus given by the sum of m_1 and m_2 , and the uncertainty, σ_M , is given by summing the uncertainties σ_{m_1} and σ_{m_2} .

It is easy to show that if there are more than two measurements, say N measurements of quantities x_i , each with uncertainty σ_{x_i} , the best estimate of the sum, X , and its uncertainty, σ_X , are given by:

$$\begin{aligned}
 X &= \sum_{i=1}^{i=N} x_i \\
 \sigma_X &= \sum_{i=1}^{i=N} \sigma_{x_i}
 \end{aligned} \tag{4.5}$$

Example 4-1: Using the Min-Max method, show that the best estimate and uncertainty on the difference of two measurements, x_1 and x_2 , with uncertainty σ_{x_1} and σ_{x_2} , are given by $X = x_1 - x_2$ and $\sigma_X = \sigma_{x_1} + \sigma_{x_2}$, respectively.

The maximum value of the difference is given when x_1 has the biggest value in its range, while x_2 has the smallest values in its range:

$$X^{max} = (x_1 + \sigma_{x_1}) - (x_2 - \sigma_{x_2})$$

and the minimum value that the difference can have is given by:

$$X^{min} = (x_1 - \sigma_{x_1}) - (x_2 + \sigma_{x_2})$$

The central value and uncertainties are then:

$$\begin{aligned}
 X &= \frac{1}{2}(X^{max} + X^{min}) \\
 &= \frac{1}{2}([(x_1 + \sigma_{x_1}) - (x_2 - \sigma_{x_2})] + [(x_1 - \sigma_{x_1}) - (x_2 + \sigma_{x_2})]) \\
 &= x_1 - x_2 \\
 \sigma_X &= \frac{1}{2}(X^{max} - X^{min}) \\
 &= \frac{1}{2}([(x_1 + \sigma_{x_1}) - (x_2 - \sigma_{x_2})] - [(x_1 - \sigma_{x_1}) - (x_2 + \sigma_{x_2})]) \\
 &= \sigma_{x_1} + \sigma_{x_2}
 \end{aligned}$$

as required.

We see that for addition and subtraction, the uncertainties on the individual measurements are always summed, whereas the central value is obtained by summing (or subtracting) the individual measurements as if they had no uncertainty. It should be clear that with this method, the final uncertainty is always “conservative”, that is, it is never going to be too small. In fact, it is almost always too big. In our example with the sum of two masses, if our measurements of m_1 and m_2 are truly independent and the errors on the individual measurements are truly random errors, it is very unlikely that the sum is actually at one of the extreme ranges given by the uncertainty. On average, it is much more likely for the error in the measurement of one mass to somewhat offset the error in the measurement of the other (unless there is a systematic bias). Our uncertainty estimated this way is almost always an overestimate of the true uncertainty in the sum, and should rather be considered as an upper limit on the uncertainty.

4.1.2 Multiplication and division

Suppose that we measured the two sides of a rectangle, l_1 and l_2 with uncertainties σ_{l_1} and σ_{l_2} , and want to know the area of the rectangle, $A = l_1 l_2$, and its uncertainty, σ_A . Multiplication (and division) are best handled by using the relative uncertainties, $\frac{\sigma_{l_1}}{l_1}$ and $\frac{\sigma_{l_2}}{l_2}$. We can write the biggest and smallest possible values for, say, l_1 , in terms of the relative uncertainty as:

$$\begin{aligned}
 l_1^{max} &= l_1 \left(1 + \frac{\sigma_{l_1}}{l_1}\right) \\
 l_1^{min} &= l_1 \left(1 - \frac{\sigma_{l_1}}{l_1}\right)
 \end{aligned} \tag{4.6}$$

The biggest possible value of the area of the rectangle is thus:

$$\begin{aligned}
 A^{max} &= l_1^{max} l_2^{max} \\
 &= l_1 \left(1 + \frac{\sigma_{l_1}}{l_1}\right) l_2 \left(1 + \frac{\sigma_{l_2}}{l_2}\right) \\
 &= l_1 l_2 \left(1 + \frac{\sigma_{l_1}}{l_1} + \frac{\sigma_{l_2}}{l_2} + \frac{\sigma_{l_1}}{l_1} \frac{\sigma_{l_2}}{l_2}\right) \\
 &\approx l_1 l_2 \left(1 + \left[\frac{\sigma_{l_1}}{l_1} + \frac{\sigma_{l_2}}{l_2}\right]\right)
 \end{aligned} \tag{4.7}$$

where in the last line, we made the approximation that if each relative uncertainty is small, then their product $(\frac{\sigma_{l_1}}{l_1} \frac{\sigma_{l_2}}{l_2})$ is really small and negligible. This is reasonable, for example if each relative uncertainty is of order 10% (0.1), then their product (0.01) is certainly negligible when compared to their sum (0.2). And often, one would have relative uncertainties well below 10%.

The minimum value of the area is found similarly:

$$\begin{aligned}
 A^{min} &= l_1^{min} l_2^{min} \\
 &= l_1 \left(1 - \frac{\sigma_{l_1}}{l_1}\right) l_2 \left(1 - \frac{\sigma_{l_2}}{l_2}\right) \\
 &= l_1 l_2 \left(1 - \frac{\sigma_{l_1}}{l_1} - \frac{\sigma_{l_2}}{l_2} - \frac{\sigma_{l_1}}{l_1} \frac{\sigma_{l_2}}{l_2}\right) \\
 &\approx l_1 l_2 \left(1 - \left[\frac{\sigma_{l_1}}{l_1} + \frac{\sigma_{l_2}}{l_2}\right]\right)
 \end{aligned} \tag{4.8}$$

The expressions for A^{min} and A^{max} are symmetric about the central value of $l_1 l_2$, so we can easily write the value of A and its uncertainty as:

$$A \pm \sigma_A = l_1 l_2 \left(1 \pm \left[\frac{\sigma_{l_1}}{l_1} + \frac{\sigma_{l_2}}{l_2}\right]\right) \tag{4.9}$$

We can thus identify that the central value and uncertainty on A are given by:

$$\begin{aligned}
 A &= l_1 l_2 \\
 \sigma_A &= A \left(\frac{\sigma_{l_1}}{l_1} + \frac{\sigma_{l_2}}{l_2}\right)
 \end{aligned} \tag{4.10}$$

More conveniently, we can see that the relative uncertainty of A is given by summing the relative uncertainties on l_1 and l_2 :

$$\frac{\sigma_A}{A} = \frac{\sigma_{l_1}}{l_1} + \frac{\sigma_{l_2}}{l_2} \tag{4.11}$$

Example 4-2: Use the Min-Max method to determine the uncertainty on the quotient, $Q = \frac{x_1}{x_2}$, of two numbers, x_1 and x_2 , that have uncertainties σ_{x_1} and σ_{x_2}

We proceed in the same way as we did for the product, using the relative uncertain-

ties. The biggest value that the quotient can have is:

$$\begin{aligned} Q^{\max} &= \frac{x_1^{\max}}{x_2^{\min}} \\ &= \frac{x_1 \left(1 + \frac{\sigma_{x_1}}{x_1}\right)}{x_2 \left(1 - \frac{\sigma_{x_2}}{x_2}\right)} \end{aligned}$$

Again, we have to assume that the relative uncertainties are small, in particular, we make use of the binomial approximation:

$$(1 + x)^\alpha \approx 1 + \alpha x$$

which applies when x is small. In our case:

$$\frac{1}{1 - \frac{\sigma_{x_2}}{x_2}} \approx 1 + \frac{\sigma_{x_2}}{x_2}$$

Q^{\max} is thus given by:

$$\begin{aligned} Q^{\max} &= \frac{x_1}{x_2} \left(1 + \frac{\sigma_{x_1}}{x_1}\right) \left(1 + \frac{\sigma_{x_2}}{x_2}\right) \\ &\approx \frac{x_1}{x_2} \left(1 + \left[\frac{\sigma_{x_1}}{x_1} + \frac{\sigma_{x_2}}{x_2}\right]\right) \end{aligned}$$

where, again, we assumed that the product of the relative uncertainties was negligible. The minimum value of the quotient is found in a similar way:

$$\begin{aligned} Q^{\min} &= \frac{x_1^{\min}}{x_2^{\max}} \\ &= \frac{x_1 \left(1 - \frac{\sigma_{x_1}}{x_1}\right)}{x_2 \left(1 + \frac{\sigma_{x_2}}{x_2}\right)} \\ &\approx \frac{x_1}{x_2} \left(1 - \frac{\sigma_{x_1}}{x_1}\right) \left(1 - \frac{\sigma_{x_2}}{x_2}\right) \\ &\approx \frac{x_1}{x_2} \left(1 - \left[\frac{\sigma_{x_1}}{x_1} + \frac{\sigma_{x_2}}{x_2}\right]\right) \end{aligned}$$

where we again used the binomial expansion and the fact that the product of the relative uncertainties is negligible. We end up with a very similar result as for multiplication, where the quotient can be written as:

$$Q \pm \sigma Q = \frac{x_1}{x_2} \left(1 \pm \left[\frac{\sigma_{x_1}}{x_1} + \frac{\sigma_{x_2}}{x_2}\right]\right)$$

and we find that the central value of the quotient is just the quotient of the two numbers without uncertainty and that the relative uncertainty on Q is found by summing the

relative uncertainties of the two numbers:

$$Q = \frac{x_1}{x_2}$$

$$\frac{\sigma Q}{Q} = \frac{\sigma_{x_1}}{x_1} + \frac{\sigma_{x_2}}{x_2}$$

In summary, for multiplication and division, we sum the relative uncertainties to obtain the relative uncertainty on the result, whereas for summation and subtraction we sum the absolute uncertainties.

We also make the same point for multiplication and division as we did for summation and subtraction: the Min-Max method overestimates the uncertainties. It is very unlikely for the “true” values of the measured quantities to be at the extrema of the uncertainty range, and this method really gives an upper limit on the uncertainty of the result.

4.1.2.1 Uncertainties in functions

We can also apply the formalism of the Min-Max method to propagate the uncertainty in a function. If we have a function, $F(x, y)$ of two measured quantities, x and y that have uncertainties σ_x and σ_y , respectively, it may not be trivial to find the particular combination of x and y that maximize and minimize F . If these combinations are found, then F and its uncertainty σ_F , are given by:

$$F \equiv \frac{1}{2}(F^{max} + F^{min})$$

$$\sigma_F \equiv \frac{1}{2}(F^{max} - F^{min}) \quad (4.12)$$

If F is a monotonic function in both x and y , then the situation is relatively straightforward. For example, if F monotonically *increases* as a function of both x and y (e.g. $F(x, y) = \sqrt{x^2 + y^2}$), then we have the simple case where:

$$F^{max} = F(x^{max}, y^{max})$$

$$F^{min} = F(x^{min}, y^{min})$$

In more complicated situations, it may be necessary to find out the answer numerically, which thankfully is easy with a computer!

Example 4-3: Given the function $F(x, y) = \sin(x + \log y)$, and the measured values of $x = 0 \pm 1$ and $y = 0.5 \pm 0.2$, use the Min-Max method to find the best estimate and the uncertainty in F

Since this is not a trivial function, we can solve this numerically using python:

Python Code 4.1: Min-Max method

```

import numpy as np
from math import *

#The function for which we want to find the uncertainties
def F(x,y):
    return np.sin(x+np.log(y))

#x and y, and their uncertainties:
x,sigma_x=0.,1
y,sigma_y=0.5,0.2

#generate 50 values within the range for x and y to scan the values of F
xvalues=np.linspace(x-sigma_x,x+sigma_x,50)
yvalues=np.linspace(y-sigma_y,y+sigma_y,50)

#initialize Fmin and Fmax outside of the possible range (since -1<F<1)
Fmin,Fmax=10,-10

#Now scan all the values of x and y to find the minimum and maximum of F
xmin,xmax,ymin,ymax=0,0,0,0
for xi in xvalues:
    for yi in yvalues:
        f=F(xi,yi)
        if f<Fmin:
            Fmin=f
            xmin=xi
            ymin=yi
        if f>Fmax:
            Fmax=f
            xmax=xi
            ymax=yi

print("The range of F is between {0:.2f} and {1:.2f}".format(Fmin,Fmax))
print("F is thus given by: {0:.2f} +/- {1:.2f}".format(0.5*(Fmax+Fmin),
    0.5*(Fmax-Fmin)))
print("These occurred at x,y = {0:.2f},{1:.2f}, and {2:.2f},{3:.2f}".
    format(xmin,ymin,xmax,ymax))

```

4.1: Output

```

The range of F is between -1.00 and 0.60
F is thus given by: -0.20 +/- 0.80
These occurred at x,y = -0.92,0.52, and 1.00,0.70

```

4.1.2.2 Summary and comments on the Min-Max method

The Min-Max method can be used to “propagate” the uncertainties of measured quantities through summation, subtraction, multiplication and division. In these cases, it is straightforward to apply as the central value of the result is always given by the central values of the measurements without uncertainties (e.g. the central value of a sum of measurements is the sum of the central values of the measurements). **In the case of summation and subtraction, the uncertainty on the result is given by the sum of the uncertainties on the measurements. In the case of multiplication and division, the relative uncertainty of the result is given by the sum of the relative uncertainties of**

the measurements. We can use the Min-Max method to evaluate the uncertainty in a function, but the answer may need to be obtained numerically.

The Min-Max method gives the uncertainty in the result under the “worst case scenario”, where the true values are assumed to be at the extreme values allowed in the uncertainty ranges, in such a way to make the biggest change in the result. If the errors on the measurements are truly random, it is very unlikely for all of the measurements to be at the extrema of their allowed range, so the **Min-Max method gives an upper limit for the uncertainty in the result.**

4.2 Adding in quadrature

It is quite unlikely that the uncertainty in a calculated number is as large as that given by the Min-Max method. For example, in the case of a sum of masses, when we use the Min-Max uncertainty, we are effectively claiming that we think it is quite possible that both measurements were either high or low, which is quite unlikely in reality. This is however not impossible. If there was a bias in the scale that we used to weigh the masses, it is in fact quite likely that both measurements are indeed at one end of the range that we quoted with the uncertainties. If this were the case, then the Min-Max method gives a reasonable uncertainty in the sum of the masses. However, if there was indeed a systematic effect that resulted in the true value being systematically at the higher or lower end of our quoted range, then we should rethink how we determined the uncertainties.

Most often, the uncertainties that we determine have some sort of “statistical nature” to them. For example, when we quote a measured values as $a \pm \sigma_a$, we usually mean that a is equally likely to be bigger or smaller than the central value. In fact, when we determine a and σ_a as the mean and error on the mean of multiple measurements, we are explicitly assuming that the true value is “normally distributed” about the central value. We will define in chapter 7 what we really mean by “normally distributed”, but it is related to saying that there is a 68% chance that the true value is in the quoted range. It turns out that most measurements are normally distributed, which justifies the following procedure for adding uncertainties (which we prove in Chapter 7).

If the the uncertainties in multiple measurements are normally distributed¹ and are independent of each other, then those uncertainties should be added in quadrature, rather than added. Addition in quadrature is performed by adding the squares of the uncertainties and then taking the square root of the sum. For the case of two masses being added together, $M = m_1 + m_2$, the quadrature uncertainty in M is given by:

$$\sigma_M = \sqrt{\sigma_{m_1}^2 + \sigma_{m_2}^2} \quad (4.13)$$

This is also true for the case of subtraction. For multiplication and division, instead of adding the relative uncertainties, we add the relative uncertainties in quadrature. For example, if the area of a rectangle is given by $A = l_1 l_2$, then the relative uncertainty in A

¹In fact, they do not necessarily need to be normally distributed, as long as their variance can be defined.

is given by:

$$\frac{\sigma_A}{A} = \sqrt{\left(\frac{\sigma_{l_1}}{l_1}\right)^2 + \left(\frac{\sigma_{l_2}}{l_2}\right)^2} \quad (4.14)$$

Addition in quadrature is not limited to only 2 terms; if there are a multiple measurements, all of the uncertainties can be squared and added together before taking the square root.

The uncertainties added in quadrature are always smaller or equal than the uncertainties that are found by straight addition (think of the Pythagorean theorem). Adding in quadrature is only correct if the various measurements are independent of each other. That is, the measurement of one quantity does not affect the measurement of another quantity whose uncertainty is being added in quadrature. If possible, one should check to see if one value is systematically dependent on another, in other words, if one of the measurement is correlated with the other (which we show how to handle in section 4.5).

4.3 Uncertainty in a function of one variable

We now introduce a more general way to propagate uncertainties. We start by considering how an uncertainty, σ_x , in a measurement of x should propagate to an uncertainty σ_F in a function $F(x)$. We assume that $F(x \pm \sigma_x)$ is the possible Min-Max range that F could have when x changes by $\pm\sigma_x$. This only makes sense if σ_x is small; for example, if $F = \sin x$ and $\sigma_x \approx \pi$, then the maximum change in $F(x)$ most definitely does not occur at $x \pm \sigma_x$. However, in the case where σ_x is small enough, we can always be assured that in that small range, $x \pm \sigma_x$, $F(x)$ is continuous and monotonous (“well behaved”). We can thus use a Taylor series about $F(x)$ to estimate $F(x \pm \sigma_x)$ (and only keep the first order term):

$$F(x \pm \sigma_x) \approx F(x) \pm \frac{dF}{dx}\sigma_x + \dots \quad (4.15)$$

Which tells us that F and its uncertainty are given by:

$$\begin{aligned} F \pm \sigma_F &= F(x) \pm \frac{dF}{dx}\sigma_x \\ \therefore \sigma_F &= \left| \frac{dF}{dx} \right| \sigma_x \end{aligned} \quad (4.16)$$

That is, the best estimate of $F(x)$ is obtained by evaluating $F(x)$ at the best estimate value for x and the uncertainty, σ_F is given by the derivative of $F(x)$ with respect to x multiplied by σ_x . The derivative is the rate of change of $F(x)$ with respect to x , so it makes sense that the change in $F(x)$ due to a change in x is given by the derivative.

Example 4-4: Show that the uncertainty in $F(x) = ax$, where a is a constant, is given by $\sigma_F = a\sigma_x$

The derivative of $F(x)$ with respect to x is:

$$\frac{dF}{dx} = a$$

Hence the uncertainty is given by:

$$\sigma_F = \frac{dF}{dx} \sigma_x = a \sigma_x$$

as required

Example 4-5: Show that the uncertainty in $F(\theta) = \sin \theta$ is given by $\sigma_F = \cos \theta \sigma_\theta$

The derivative of $F(\theta)$ with respect to θ is:

$$\frac{dF}{d\theta} = \cos \theta$$

Hence the uncertainty is given by:

$$\sigma_F = \left| \frac{dF}{d\theta} \right| \sigma_\theta = \cos \theta \sigma_\theta$$

as required. Note that this only works when θ is expressed in radians!

4.4 The general case: functions of multiple variables

We now consider the case of a function of multiple variables, for example, $F(x, y)$. Again, we assume that the uncertainties in x and y , σ_x and σ_y , are both small, such that $F(x, y)$ is well behaved. That is, the Min-Max range of $F(x, y)$ near the best estimate values of x and y is given by: $F(x \pm \sigma_x, y \pm \sigma_y)$. We again use the Taylor series formula (and limit it to the first order terms), to approximate $F(x, y)$ in the region near the best estimates of x and y :

$$F(x \pm \sigma_x, y \pm \sigma_y) \approx F(x, y) \pm \left(\frac{\partial F}{\partial x} \sigma_x + \frac{\partial F}{\partial y} \sigma_y \right) \quad (4.17)$$

where we have now used the partial derivatives², since F depends on multiple variables. The best estimate of $F(x, y)$ is thus found by evaluating $F(x, y)$ at the estimate values of x and y . The uncertainty in $F(x, y)$ is given by:

$$\sigma_F = \left| \frac{\partial F}{\partial x} \right| \sigma_x + \left| \frac{\partial F}{\partial y} \right| \sigma_y \text{ (upper limit)} \quad (4.18)$$

This derivation however assumed that both x and y have changed in the “worst possible way” so as to lead to the biggest possible change in F . It is more likely that the true values

²Recall that when you take the partial derivative with respect to one variable, you treat all the others as constants

of x and y somewhat offset each other in the effect that they have on $F(x, y)$. For example, if $F(x, y) = x + y$, and the true value of x is in the high side of $x \pm \sigma_x$ and the true value of y is in the low side of $y \pm \sigma_y$, then the true value of F is actually quite close to $x + y$. Again, if the uncertainties in x and y are truly random and independent, then it is quite unlikely that they lead to the maximum possible uncertainty in F . Thus, in the case where the uncertainties are random and uncorrelated, it makes more sense to add them in quadrature:

$$\sigma_F = \sqrt{\left(\frac{\partial F}{\partial x} \sigma_x\right)^2 + \left(\frac{\partial F}{\partial y} \sigma_y\right)^2} \quad (\text{x and y random and uncorrelated}) \quad (4.19)$$

This formula applies no matter how many variables F depends on; you can just add more terms to the sum in quadrature.

Example 4-6: Use the derivative method to find a formula for the uncertainty, σ_F , in $F(x, y) = \frac{x}{y}$ if x and y have uncertainties σ_x and σ_y . Show that this gives the same uncertainty as found by adding the relative uncertainties in x and y together in quadrature.

We have:

$$\begin{aligned} F(x, y) &= \frac{x}{y} \\ \therefore \frac{\partial F}{\partial x} &= \frac{1}{y} \\ \therefore \frac{\partial F}{\partial y} &= \frac{-x}{y^2} \end{aligned}$$

The uncertainty on F from the derivative method is thus:

$$\begin{aligned} \sigma_F &= \sqrt{\left(\frac{\partial F}{\partial x} \sigma_x\right)^2 + \left(\frac{\partial F}{\partial y} \sigma_y\right)^2} \\ &= \sqrt{\left(\frac{1}{y} \sigma_x\right)^2 + \left(\frac{-x}{y^2} \sigma_y\right)^2} \end{aligned}$$

If we factor out $\frac{x}{y}$ from the above expression, we have:

$$\begin{aligned} \sigma_F &= \sqrt{\left(\frac{1}{y} \sigma_x\right)^2 + \left(\frac{x}{y^2} \sigma_y\right)^2} \\ &= \frac{x}{y} \sqrt{\left(\frac{\sigma_x}{x}\right)^2 + \left(\frac{\sigma_y}{y}\right)^2} \end{aligned}$$

which is exactly the expression that one obtains from adding the relative uncertainties together.

Example 4-7: Use the derivative method to evaluate a formula for the uncertainty, σ_F , in the function $F(x, y) = \sin(x + \ln y)$ if x and y have uncertainties σ_x and σ_y . Give the best estimate and uncertainty in $F(x, y)$ for the case of $x = 3.0 \pm 0.5$ rad and $y = 8 \pm 1$.

We have:

$$\begin{aligned} F(x, y) &= \sin(x + \log y) \\ \therefore \frac{\partial F}{\partial x} &= \cos(x + \ln y) \\ \therefore \frac{\partial F}{\partial y} &= \cos(x + \ln y) \frac{d}{dy} x + \log y = \cos(x + \log y) \frac{1}{y} \end{aligned}$$

The uncertainty in F is thus:

$$\begin{aligned} \sigma_F &= \sqrt{\left(\frac{\partial F}{\partial x} \sigma_x\right)^2 + \left(\frac{\partial F}{\partial y} \sigma_y\right)^2} \\ &= \sqrt{(\cos(x + \ln y) \sigma_x)^2 + \left(\cos(x + \log y) \frac{1}{y} \sigma_y\right)^2} \end{aligned}$$

which we can easily do in python:

Python Code 4.2: Symbolic derivative error propagation with sympy

```
import sympy as sym
sym.init_printing()
#Let's use sympy to evaluate the error with the derivatives:
#Declare variables x and y and their uncertainties as symbols
x, y, sigma_x, sigma_y = sym.symbols('x y \sigma_x \sigma_y')
#Build a function out of x and y:
F = sym.sin(x+sym.ln(y))
#Find the derivatives
dFdx=sym.diff(F, x)
dFdy=sym.diff(F, y)
#Add them in quadrature to have an expression for sigma_F
sigma_F=sym.sqrt((dFdx*sigma_x)**2+(dFdy*sigma_y)**2)
#Print out the result:
print("Formula for sigma_F from the derivative:\n", sigma_F)
#In a notebook, we can just print it out in pretty version:
sigma_F
```

4.2: Output

Formula for sigma_F from the derivative:


```
sqrt(\sigma_x**2*cos(x + log(y))**2 + \sigma_y**2*cos(x + log(y))**2/y*
*2)
```

And then evaluating with actual numbers:

Python Code 4.3: Numerical evaluation of symbolic values from sympy

```
#Let us now evaluate this numerically.
#Put the tuple of values into a list
values=[(x,3.0),(sigma_x,0.5),(y,8.0),(sigma_y,1.0)]
#Create variables that have the variables subbed into the expressions.
#and evaluate those as numbers:
nValue_F=sym.N(F.subs(values))
nValue_sigmaF=sym.N(sigma_F.subs(values))
#Print the result
print("The best estimate and uncertainty in F(x,y) is {:.2f} +/- {:.2f}"
      .format(nValue_F,nValue_sigmaF))
```

4.3: Output

The best estimate and uncertainty in $F(x,y)$ is -0.93 ± 0.18

Example 4-8: Evaluate the best estimate of $F(x,y,z)$ and its uncertainty, σ_F , for the case where $F(x,y,z) = \sin(x + \ln y) \cos(z) \sqrt{x^2 + y^2 + 2z}$, where the following values have been measured: $x = 8 \pm 0.1$, $y = 3 \pm 0.2$, and $z = 10 \pm 0.3$

In this case, it is much easier to just use symbolic computation in python.

Python Code 4.4: Symbolic-based derivative error with sympy

```
import sympy as sym
sym.init_printing()
#Let's use sympy to evaluate the error with the derivatives:
#Declare variables x, y and z and their uncertainties as symbols
x,y,sigma_x, sigma_y = sym.symbols('x y \sigma_x \sigma_y')
z,sigma_z = sym.symbols('z \sigma_z')
#Build a function out of x and y:
F=(sym.sin(x+sym.ln(y))*sym.cos(z))*(sym.sqrt(x**2+y**2+2*z))
#Find the derivatives
dFdx=sym.diff(F,x)
dFdy=sym.diff(F,y)
dFdz=sym.diff(F,z)
#Add them in quadrature to have an expression for sigma_F
sigma_F=sym.sqrt((dFdx*sigma_x)**2+(dFdy*sigma_y)**2+(dFdz*sigma_z)**2)
#The result is a little messy to print out...
#print("Formula for sigma_F from the derivative:\n",sigma_F)
#Create an array of tuples for the numerical values:
values=[(x,8),(sigma_x,0.1),(y,3),(sigma_y,0.2),(z,10),(sigma_z,0.3)]
#Create variables that have the variables subbed into the expressions.
#and evaluate those as numbers:
nValue_F=sym.N(F.subs(values))
nValue_sigmaF=sym.N(sigma_F.subs(values))
#Print the result
```

```
print("The best estimate and uncertainty in  $F(x,y)$  is {:.2f} +/- {:.2f}"
.format(nValue_F, nValue_sigmaF))
```

4.4: Output

The best estimate and uncertainty in $F(x,y,z)$ is -2.59 ± 1.02

You can easily modify this code to calculate the error on any function.

4.5 Correlated uncertainties

We have examined two extreme cases for propagating the uncertainties in measured quantities into the uncertainty for a calculated quantity. In the worst case scenario, when the true values of all of the measurements have conspired to result in the largest change in the computed quantity, the uncertainties should be added together directly (as in equation 4.18). In this case, we say that the measurements are “correlated”, since they have all “worked together” to result in the largest possible uncertainty. If the measurements are completely independent from each other, then the uncertainties should be added in quadrature (as in equation 4.19).

m_1 (kg)	m_2 (kg)	$M = m_1 + m_2$ (kg)
399.3	193.2	592.5
404.6	205.1	609.7
394.6	192.6	587.2
396.3	194.2	590.5
399.6	196.6	596.2
404.9	201.0	605.9
387.4	184.7	572.1
404.9	215.2	620.1
398.2	203.6	601.8
407.2	207.8	615.0

Table 4.1: Measurements of masses and their sum, for correlated measurements.

But what if the measurements are “somewhat” correlated? That is, what if measuring one quantity to be high usually makes another quantity high as well, but not always? In the next chapter, we will formalize our definition of the “covariance”, σ_{xy} , which is a measure of how correlated two quantities are. If we have two quantities, x and y , that we measure simultaneously N times, giving us two sets of measurements $x_i = \{x_1, x_2, \dots, x_N\}$ and

$y_i = \{y_1, y_2, \dots, y_N\}$, the covariance, σ_{xy} , is defined to be:

$$\sigma_{xy} \equiv \frac{1}{N-1} \sum_{i=1}^{i=N} (x_i - \bar{x})(y_i - \bar{y}) \quad (4.20)$$

where \bar{x} and \bar{y} are the means (as defined back in equation 3.1) of the measurements of x and y , respectively.

Suppose that we are interested in the sum of two masses $M = m_1 + m_2$, and that we have made several measurements of m_1 and m_2 which are tabulated in Table 4.1. For convenience, we have also added a third column with the sum of the masses measured in each row.

We have determined the mean and the standard deviation (Equation 3.1) for the individual mass measurements to be $\bar{m}_1 = 399.7$ kg, $\sigma_{m_1} = 6.01$ kg, $\bar{m}_2 = 199.4$ kg, and $\sigma_{m_2} = 8.87$ kg (we kept a non-significant figure here to propagate in our uncertainty calculations). So what would should we quote for M and its uncertainty? There are three obvious options:

1. We can use the mean and error on the mean from the values in the third column. This gives $M = 599.1 \pm 4.54$ kg, which must certainly be correct.
2. We can add the standard deviations of m_1 and m_2 in quadrature to get the standard deviation of M , and then divide by the square root of the number of measurements to get the error on the mean for M : $\sigma_M = \frac{1}{\sqrt{N}} \sqrt{\sigma_{m_1}^2 + \sigma_{m_2}^2}$. This gives $M = 599.1 \pm 3.39$ kg, which is an underestimate of the uncertainty.
3. We can add the standard deviations of m_1 and m_2 (not in quadrature) to get the standard deviation in M , and then divide that by the square root of the number of measurements to get the error on the mean for M : $\sigma_M = \frac{1}{\sqrt{N}} (\sigma_{m_1} + \sigma_{m_2})$. This gives $M = 599.10 \pm 4.71$ kg, which is an over estimate.

As expected, the error from the sum in quadrature is smaller than the error from the straight sum. Treating the third column as individual measurements and using the error on the mean from those values must certainly be correct, and should in fact be most representative of the actual variation that we get in the sum. This gives an uncertainty that is larger than the quadrature sum, but not as large as the straight sum. So we have a situation that is closer to the worst case scenario of having to add the uncertainties, but not the worst possible case. We say that the two values of m_1 and m_2 are “correlated”.

We can see this graphically very easily by plotting a scatter plot of m_1 and m_2 , as in Figure 4.1, where we can see that on average, m_2 is higher than its mean (199.4 kg) when m_1 is higher than its mean (399.7 kg). There is a trend and the values are not randomly distributed. If m_1 and m_2 really were randomly distributed, we would expect the scatter plot to look like an ellipse.

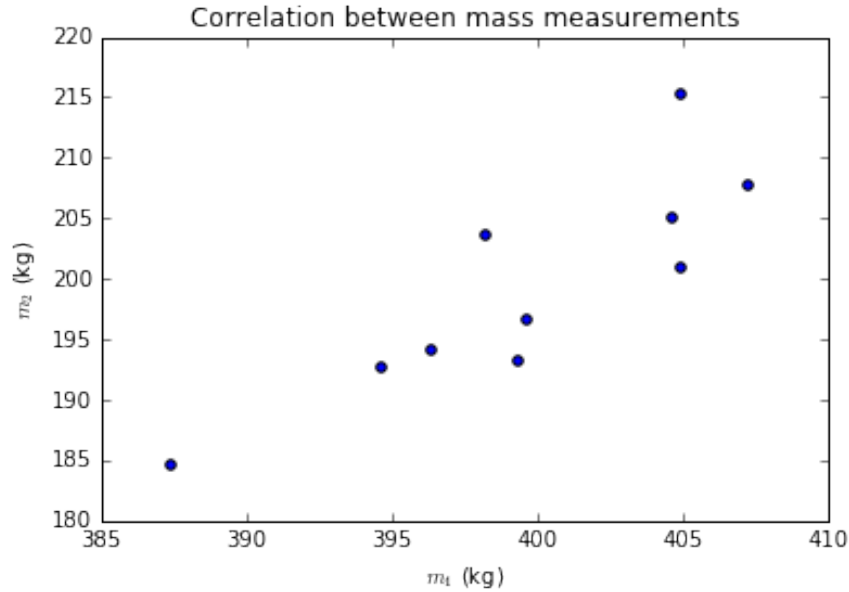


Figure 4.1: Two measurements that are not completely independent of each other (correlated).

The correct way to propagate the uncertainty in quantities that are correlated is to first calculate the covariance factor, σ_{xy} , and then to use a more general version of equation 4.19:

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

$$\sigma_F = \sqrt{\left(\frac{\partial F}{\partial x} \sigma_x\right)^2 + \left(\frac{\partial F}{\partial y} \sigma_y\right)^2 + 2 \frac{\partial F}{\partial x} \frac{\partial F}{\partial y} \sigma_{xy}} \quad (4.21)$$

where the partial derivatives are evaluated at the central values of x and y , and σ_{xy} is the covariance between x and y . The formula is easily extended for more than 2 variables, where there will be one cross term for each pair of correlated variables. We will derive this formula in Chapter 5.

Using equation 4.21 for the example of the two masses in table 4.1, the covariance is determined to be $\sigma_{m_1 m_2} = 45.61$ leading to an uncertainty in the sum of $\sigma_M = \frac{1}{\sqrt{N}} \sqrt{\sigma_{m_1}^2 + \sigma_{m_2}^2 + 2\sigma_{m_1 m_2}} = 4.54$ kg, which is the same result as obtained by treating the third column in Table 4.1 as independent measurements.

We used equation 4.21 to estimate the standard deviation of the sum, and then divided that by the square root of the number of measurements to obtain the error on the mean of the sum. This is because the covariance is a property that does not depend on the number of measurements, so it cannot be added to the errors on the mean of m_1 and m_2 . We will see in chapter 7 that equation 4.21 comes from estimating the variance (the square of the standard deviation) of the calculated quantity in terms of the variances of the measured quantities (the square root then gives the standard deviation, and dividing by \sqrt{N} then leads to the error on the mean of the calculated quantity).

The code below compares the various computations for estimating the error on the sum of the masses from Table 4.1:

Python Code 4.5: Different error calculations for a set of values

```
import numpy as np
from math import *
#Copy and paste the data, add commas by hand
data=np.array([
399.3, 193.2, 592.5,
404.6, 205.1, 609.7,
394.6, 192.6, 587.2,
396.3, 194.2, 590.5,
399.6, 196.6, 596.2,
404.9, 201.0, 605.9,
387.4, 184.7, 572.1,
404.9, 215.2, 620.1,
398.2, 203.6, 601.8,
407.2, 207.8, 615.0])
#Reshape into the correct format
data=data.reshape(10,3)
#Extract the columns:
m1=data[:,0]
m2=data[:,1]
msum=data[:,2]
#Get the mean and standard deviations of the columns:
mlmean=m1.mean() #mean
mlstd=m1.std(ddof=1) #std
m2mean=m2.mean() #mean
m2std=m2.std(ddof=1) #std
msmean=msum.mean() #mean
msstd=msum.std(ddof=1) #std
msemean=msstd/sqrt(msum.size) #error on the mean
#Calculate the covariance
cov=((m1-mlmean)*(m2-m2mean)).sum()/(m1.size-1)
#Print out comparisons of the uncertainties
print("m1 = {:.2f} +/- {:.2f}".format(mlmean,mlstd))
print("m2 = {:.2f} +/- {:.2f}".format(m2mean,m2std))
print("covariance = {:.2f}".format(cov))
print("Treat as independent measurements: M = {:.2f} +/- {:.2f} (correct)".
      format(msmean, msemean))
print("Quadrature: M = {:.2f} +/- {:.2f} (underestimate)".format(msmean, sqrt(
    mlstd**2+m2std**2)/sqrt(msum.size)))
print("Sum error: M = {:.2f} +/- {:.2f} (overestimate)".format(msmean, (mlstd+
    m2std)/sqrt(msum.size)))
print("With covariance: M = {:.2f} +/- {:.2f} (correct)".format(msmean, sqrt(
    mlstd**2+m2std**2+2*cov)/sqrt(msum.size)))
```

4.5: Output

```
m1 = 399.70 +/- 1.90
m2 = 199.40 +/- 2.81
covariance = 45.67
Treat as independent measurements: M = 599.10 +/- 4.54 (correct)
Quadrature: M = 599.10 +/- 3.39 (underestimate)
Sum error: M = 599.10 +/- 4.71 (overestimate)
```

With covariance: $M = 599.10 \pm 4.54$ (correct)

It is worth noting that the covariance, $\sigma_{m_1 m_2}$, is negative if the variables are “anti-correlated”; that is, if one value being predictably higher than its mean leads to the other variable being predictably lower than its mean. A negative covariance can lead to a smaller overall uncertainty in equation 4.21 than one would get if the variables were independent (by adding in quadrature). Such an example is illustrated by the data in Table 4.2 and shown in the scatter plot in Figure 4.2. The output of the program above, using the data from Table 4.2 is:

$m_1 = 397.60 \pm 3.58$

$m_2 = 196.94 \pm 5.42$

covariance = -148.24

Treat as independent measurements: $M = 594.55 \pm 3.54$ (correct)

Quadrature: $M = 594.55 \pm 6.50$ (overestimate)

Sum error: $M = 594.55 \pm 9.00$ (overestimate)

With covariance: $M = 594.55 \pm 3.54$ (correct)

and shows that the correct uncertainty obtained using the negative covariance is in fact smaller than that obtained by adding in quadrature.

m_1 (kg)	m_2 (kg)	$M = m_1 + m_2$ (kg)
397.4	184.1	581.5
388.8	205.4	594.2
400.5	199.4	599.9
416.2	172.0	588.2
396.5	212.7	609.2
375.2	226.4	601.6
409.4	176.9	586.4
400.5	209.6	610.1
390.0	187.0	577.0
401.5	195.9	597.4

Table 4.2: Measurements of masses and their sum, for anti-correlated measurements.

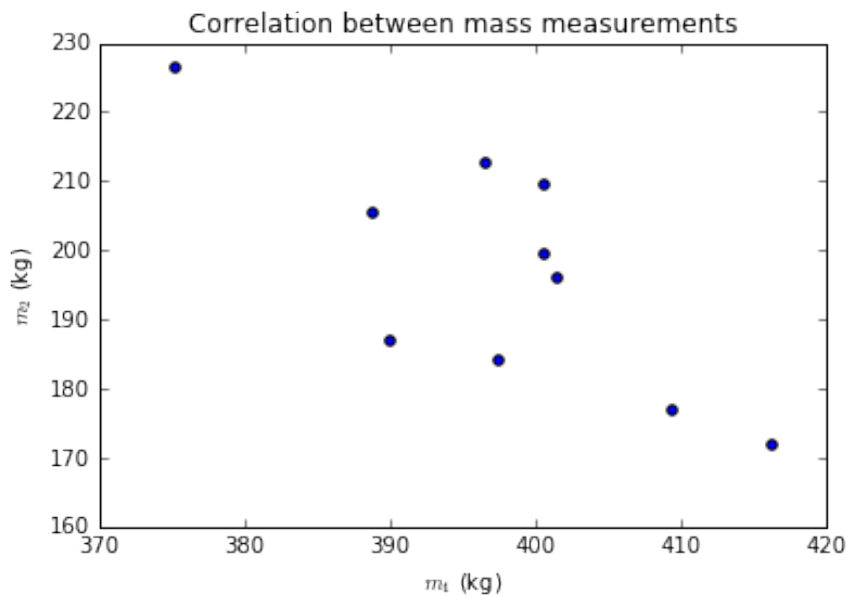


Figure 4.2: Two measurements that are not completely independent of each other (anti-correlated).

4.6 Averaging numbers with uncertainties

Suppose that two different experiments have measured the same quantity, X , and obtained, say $x_1 = 10 \pm 0.1$ and $x_2 = 11 \pm 1$, respectively. We assume that both measurements have correctly estimated their uncertainties, and that these each represent a 68% confidence of the true value being within their quoted uncertainty range. How do we combine both of these measurements into a single average measurement? How do we compute the uncertainty in the average measurement?

If we use the derivative formula to get the average, we find:

$$X = \frac{1}{2}(x_1 + x_2) = 10.5$$

$$\sigma_X = \frac{1}{2}\sqrt{\sigma_{x_1}^2 + \sigma_{x_2}^2} = 0.5$$

which does not make much sense. The first measurement, $x_1 = 10 \pm 0.1$, is clearly more precise than the second one, $x_2 = 11 \pm 1$. Since we trust both measurements, our average must somehow include the fact that x_1 is more precise, and we thus expect that it should be closer to 10 than to 11. We also expect that averaging in a less precise result should not “blow up” the uncertainty on our number; at worst it should do nothing to the uncertainty (providing the second measurement is consistent with the first one, which it is in this case). In the extreme case, you can imagine that x_1 is well measured, and x_2 is not measured at all (maybe Team 2 were busy doing other things and forgot to perform the experiment); in this extreme case, a bad measurement of x_2 should not impact our knowledge of X or the uncertainty that we obtained from the x_1 measurement. If the two measurements are consistent, we expect that the overall uncertainty should decrease.

The correct way to combine multiple measurements of the same quantity is to perform a

weighted average of the measurements, where the uncertainties (squared) are used as the weights. We will justify this approach in chapter 7. If N measurements of X , $\{x_1, x_2, \dots\}$ each have uncertainties, $\{\sigma_{x_1}, \sigma_{x_2}, \dots\}$, then the average value is given by:

$$X = \frac{\sum_{i=1}^{i=N} w_i x_i}{\sum_{i=1}^{i=N} w_i}$$

$$\sigma_X = \frac{1}{\sqrt{\sum_{i=1}^{i=N} w_i}}$$

$$w_i \equiv \frac{1}{\sigma_{x_i}^2}$$

If we use this formula for two measurements that we had above, we get:

$$w_1 = \frac{1}{0.1^2} = 100$$

$$w_2 = \frac{1}{1^2} = 1$$

$$X = \frac{\sum_{i=1}^{i=N} w_i x_i}{\sum_{i=1}^{i=N} w_i} = \frac{100 \times 10 + 1 \times 11}{101} = 10.0099$$

$$\sigma_X = \frac{1}{\sqrt{\sum_{i=1}^{i=N} w_i}} = \frac{1}{\sqrt{101}} = 0.09950$$

where we kept the extra decimals to show that the uncertainty gets a tiny bit smaller, and the central value of the measurement is shifted very slightly in the direction of x_2 . As you can see, the first measurement has a weight of 100 and the second measurement only has a weight of 1, in the weighted average. This reflects the factor of 10 difference in the uncertainties from the two measurements, which gets squared when the measurements are combined.

4.7 Summary

In this chapter, we showed how to propagate the uncertainties from a set of quantities through to results calculated from those quantities. We introduced the Min-Max method as a way to understand the biggest possible change in a calculated quantity resulting from variations in the measured quantities. We also introduced the “derivative” method which applies more generally than the Min-Max method. We found that the uncertainty in a calculated quantity is obtained by summing uncertainties in measured quantities. We argued that a quadrature sum of the uncertainties was usually more appropriate in the case where the measured quantities are independent from each other and have random errors. We argued that a straight sum generally provides an upper limit on the uncertainty in a calculated result. The sum in quadrature assumes that the individual quantities are independent from each other (see below for the general case when quantities are allowed to be correlated).

The main formula for propagating uncertainties is given by the derivative method. If a

function, $F(x_1, x_2, \dots)$, depends on N independent quantities, x_1, x_2, \dots , each with uncertainties $\sigma_{x_1}, \sigma_{x_2}, \dots$, then the best estimate of F is obtained by evaluating F at the best estimate values of the x_i , and the uncertainty on F , σ_F , is given by the following quadratic sum:

$$\sigma_F = \sqrt{\sum_{i=1}^{i=N} \left(\frac{\partial F}{\partial x_i} \sigma_{x_i} \right)^2} \quad (4.22)$$

where $\frac{\partial F}{\partial x_i}$ is the partial derivative of $F(x_1, x_2, \dots)$ with respect to x_i .

When averaging n measurements x_i , each with their own individual uncertainty σ_{x_i} , the average, X , and its uncertainty, σ_X , are given by:

$$\begin{aligned} X &= \frac{\sum_{i=1}^{i=n} w_i x_i}{\sum_{i=1}^{i=n} w_i} \\ \sigma_X &= \frac{1}{\sqrt{\sum_{i=1}^{i=n} w_i}} \\ w_i &\equiv \frac{1}{\sigma_{x_i}^2} \end{aligned}$$

If we have two quantities, x and y , that we measure simultaneously N times, giving us two sets of measurements $x_i = \{x_1, x_2, \dots, x_N\}$ and $y_i = \{y_1, y_2, \dots, y_N\}$, the covariance between x and y , σ_{xy} , is defined to be:

$$\sigma_{xy} \equiv \frac{1}{N-1} \sum_{i=1}^{i=N} (x_i - \bar{x})(y_i - \bar{y}) \quad (4.23)$$

where \bar{x} and \bar{y} are the means of the measurements of x and y , respectively. If the covariance is non-zero, then we need to take it into account and have to modify the formula from the derivative method:

$$\begin{aligned} F &= F(x, y) \\ \sigma_F &= \sqrt{\left(\frac{\partial F}{\partial x} \sigma_x \right)^2 + \left(\frac{\partial F}{\partial y} \sigma_y \right)^2 + 2 \frac{\partial F}{\partial x} \frac{\partial F}{\partial y} \sigma_{xy}} \end{aligned} \quad (4.24)$$

where σ_x and σ_y are the standard deviations of the measurements of x and y , respectively. If there are more than two variables, one can simply add more terms to the sum (including 1 covariance factor between each pair of variables). This is the most general formula and is always correct. Furthermore, it does not require the individual variables to be normally distributed; this formula gives the correct estimate of the standard deviation in a result, regardless of the distributions for the underlying measurements. In practice, one should estimate whether including the covariance makes a difference, and include it if it is significant.

5

Statistics - Presenting Data

5.1 Data and statistics

Statistical analysis is useful in many field, including physics, because virtually any measurement that is performed multiple times will show a variation, either due to the imperfection of the measuring apparatus, because of variations in the samples, or because the quantity truly has a variance (e.g. the speed of gas molecules at a specific temperature and pressure).

For example, if you wanted to model the length of 8 month old cats and set out to measure cats, you would get a variation in your measurements: you would get variations between different cat breeds, you would get variations because the cats will have different lengths of hair, you would get variations because some of the cats would move more than others while you try to measure them, and most likely, you will get variations because not all 8 month old cats are identical. Even if you tried to minimize systematic effects (e.g. by shaving and sedating all the cats and working with a single breed), you will still get variations.

The goal of statistics would be to understand if your measurements allow you to infer something about a model for cat length, perhaps that 8 month old cats are on average $L \pm dL$ long. Maybe you would find that the average cat length is different between countries, or different types of cat owners. As a scientist, you want to be assured that you can make rigorous statements about your findings and rigorously test different cat length models even when the data are expected to show a spread.

For the specific case of many measurements of a single value (e.g cat length), there are several common quantities that can be calculated to provide some description of the set (i.e. the distribution) of measurements that were obtained. We start by introducing those quantities (such as the mean and standard deviation), and then describe the use of the histogram to visualize data of a statistical nature.

5.2 Basic statistical parameters of a set of measurements

5.2.1 Mean and variance

The mean and variance are the most common quantities to characterize a distribution of measurements. If a set of N measurements, $\{x_1, x_2, \dots, x_N\}$, of a single quantity were performed, the “sample mean” is defined as:

$$\bar{x} \equiv \frac{1}{N} \sum_{i=1}^{i=N} x_i \quad (5.1)$$

which is simply the algebraic average. The “sample variance”, σ_x^2 , is defined as:

$$\text{var} \equiv \sigma_x^2 = \frac{1}{N-1} \sum_{i=1}^{i=N} (x_i - \bar{x})^2 \quad (5.2)$$

and the “sample standard deviation”, σ_x , is defined as the square root of the sample variance:

$$\sigma_x \equiv \sqrt{\text{var}} \quad (5.3)$$

While the sample mean gives the average value of the measurement, the sample variance and standard deviation indicate the average distance between the measurements and the sample mean. That is, whether the measured values are all really close to the sample mean, or whether they are spread out. The sample standard deviation is also a representative measure of the uncertainty in a single measurement.

Note the $(N-1)$ in the definition of the sample variance, instead of N . This is a subtlety that arises from the fact that the formula for the sample variance uses the sample mean as determined from the data. If N is large, there will not be a big difference in the result from including the correct $N-1$.

This subtlety arises due to the fact that the “true” mean is not known, rather it has been estimated from the data. One way to think of this is that there is a large “population” of values, and we have only measured a small sample of them (for example, you would measure a small subset of cats and try to infer something about all cats). It is likely that, by chance, the sample mean that you measured is not exactly the population mean. We think of the sample mean as an “estimator”, also referred to as a “statistic”, whereas the true population mean can be referred to as a “parameter” of the population (which you are trying to estimate from data). The “population” may represent an actual group of which you have measured a subset, or it could be a theoretical model. That is, your sample mean could be an estimator for the mean of a model that you are using to describe the data.

Similarly, the sample variance is only an estimator for the true variance in the population. It has been shown (Bessel’s correction), that dividing by N instead of $(N-1)$ would yield a sample variance that is systematically smaller than the true population variance (we would call this a “biased estimator”). Dividing by $(N-1)$ removes the bias.

Note that if the true mean of the population, μ , is known, then the sample variance as an estimator of the population variance is given by:

$$\text{var} \equiv \sigma_x^2 = \frac{1}{N} \sum_{i=1}^{i=N} (x_i - \mu)^2 \quad (5.4)$$

where we now divide by N since we did not use an estimate for the mean of the population.

5.2.2 Error on the mean

The variance and standard deviation are measurements of how spread out the measurements are relative to the mean. We can also determine how well the mean is determined from our sample by defining the “error on the mean”, $\sigma_{\bar{x}}$, which is given by the standard deviation divided by the square root of the number of measurements:

$$\sigma_{\bar{x}} = \frac{\sigma_x}{\sqrt{N}} \quad (5.5)$$

We will prove this result in Chapter 7, but we can observe here that it makes intuitive sense. As we gather more samples, then the error on the mean decreases (as the square root of the number of samples), which makes sense.

5.2.3 Mode and median

The mode and median are also of use to describe data that have a distribution of values. The mode is the value that occurs the most often (sometimes also called the “most probable value”). The median of the data set is the value for which there are an equal number of values that are bigger and smaller. For a set of measurements that are normally (gaussian) distributed (as described in Chapter 7), the mean, mode and median overlap. In general, all three parameters can be different.

Example 5-1: Calculate the mean, variance, error on the mean, mode and median for the following set of measurements: {1.1, 1.2, 1.0, 1.3, 1.4, 1.5, 1.2, 1.3, 1.3, 1.1, 1.0, 1.6 }

We can compute these quantities using a simple python program:

Python Code 5.1: Basic statistical quantities

```
import numpy as np
import scipy.stats as stats
from math import *
#Copy the data from the book into a numpy array
xi=np.array([1.1, 1.2, 1.0, 1.3, 1.4, 1.5, 1.2, 1.3, 1.3, 1.1, 1.0,
1.6])
N=xi.size
#Use the numpy mean() function to get the mean:
xmean=xi.mean()
print("mean: {:.2f}".format(xmean))
#The variance (note that we have to specify ddof=1, which results in
using the
# N-1 in the denominator instead of N)
xvar=xi.var(ddof=1)
print("variance: {:.3f}".format(xvar))
```

```

#The standard deviation is either the square root of the variance,
#or we can get it from numpy
xstd=xi.std(ddof=1)
print("std. dev.: {:.3f}".format(xstd))
#The error on the mean
xmeanerr=xstd/sqrt(N)
print("error on the mean: {:.3f}".format(xmeanerr))

#Sort the array to make it easier to see the mode and median:
xi_sorted = np.sort(xi)
print("Sorted:", xi_sorted)
#The median can be found by:
xmedian=np.median(xi)
print("The median is:", xmedian)
#The mode must be found using the scipy.stats module
xmode, modecount=stats.mode(xi)
print("The mode is:", xmode[0])

```

5.1: Output

```

mean: 1.25
variance: 0.035
std. dev.: 0.188
error on the mean: 0.054
Sorted: [ 1.   1.   1.1  1.1  1.2  1.2  1.3  1.3  1.3  1.4  1.5  1.6]
The median is: 1.25
The mode is: 1.3

```

Note that by default, the *numpy* module uses N in the denominator instead of $N - 1$ when calculating the sample variance and standard deviation. One needs to specify the “delta degrees of freedom” parameter, *ddof*, which is the number to subtract from N (in our case it’s 1).

5.2.4 Moments, skewness and kurtosis

The skewness and kurtosis are higher order “moments” of the data that describe how symmetric the data are with respect to the mode (skewness) and whether the data are grouped tightly at the mode (kurtosis).

The α “sample moment” of the data is defined as:

$$m_{\alpha} = \frac{1}{N-1} \sum_{i=1}^{i=N} (x_i - \bar{x})^{\alpha} \quad (5.6)$$

The sample variance is thus the “second” sample moment. The skewness is defined as:

$$\text{skew} = \frac{m_3}{m_2^{3/2}} \quad (5.7)$$

The skewness will be positive if a larger fraction of the measurements are bigger than the mode. It will be negative if most of the measurements are smaller than the mode. Effectively it tells you whether the mean is to the right or the left of the mode. If the skewness is 0, then mean and mode coincide.

The kurtosis is defined as:

$$\text{kurt} = \frac{m_4}{m_2^2} \quad (5.8)$$

If the data are normally distributed about the mean, then the kurtosis is equal to 3. If the data have a kurtosis higher than 3, then the distribution has a sharper peak than a normal distribution. The smallest kurtosis is 1, corresponding to data that are not concentrated about any specific value.

5.2.5 Covariance and correlation

Often, we may measure two different quantities simultaneously multiple times. For example, we may measure both the length and the weight of 8 month old cats, and we probably would expect that the two are “correlated”; that is, that if one quantity is bigger, then the other one tends to be bigger as well. We can quantify how correlated two quantities are by measuring the covariance and the correlation factor.

Let us suppose that we have two quantities, x and y , that we measure simultaneously N times, giving us two sets of measurements $x_i = \{x_1, x_2, \dots, x_N\}$ and $y_i = \{y_1, y_2, \dots, y_N\}$. The covariance, σ_{xy} , is defined to be:

$$\sigma_{xy} \equiv \frac{1}{N-1} \sum_{i=1}^{i=N} (x_i - \bar{x})(y_i - \bar{y}) \quad (5.9)$$

where \bar{x} and \bar{y} are the means of the measurements of x and y , respectively. The $N-1$ in the denominator has the same origin as when calculating the variance, and is required to recover the correct formula for the variance if $y = x$. The covariance is thus positive when x and y tend to both be larger or smaller than their mean at the same time (for example x_i bigger than \bar{x} means that y_i is likely bigger than \bar{y}). The covariance is negative when x and y tend to be in opposite directions from their means at the same time (e.g. x_i bigger than \bar{x} leads to y_i being smaller than \bar{y}). The covariance is near 0 when a value of x bigger or smaller than \bar{x} does not lead to a value of y that is predictably bigger or smaller than \bar{y} . When the covariance is positive we say that x and y are correlated, whereas when it is negative, we say that they are “anti-correlated”.

The correlation factor, ρ_{xy} , sometimes also called r or R or “the r-factor”, is simply the covariance divided by the standard deviations of x and y , σ_x and σ_y , respectively. This leads to ρ_{xy} being bound between -1 and 1. A correlation factor of 1 means that x and y are perfectly correlated, a value of 0 indicates that they are un-correlated, and a value of -1 that they are perfectly anti-correlated. The correlation factor, ρ_{xy} is given by:

$$\rho_{xy} \equiv \frac{\sigma_{xy}}{\sigma_x \sigma_y} \quad (5.10)$$

Example 5-2: Calculate the covariance and correlation factors for the following 12 measurements of the quantities x and y : $x_i = \{1.1, 1.2, 1.0, 1.3, 1.4, 1.5, 1.2, 1.3, 1.3, 1.1, 1.0, 1.6\}$ and $y_i = \{1.3, 0.9, 0.9, 1.0, 1.6, 1.1, 1.2, 1.2, 1.2, 0.9, 0.9, 1.1\}$

The quantities can easily be evaluated in python using the following code:

Python Code 5.2: Calculating covariance and correlation

```
import numpy as np
import pylab as pl

#copy the values into numpy arrays:
xi=np.array([1.1, 1.2, 1.0, 1.3, 1.4,1.5, 1.2, 1.3, 1.3, 1.1, 1.0, 1.6])
yi=np.array([1.3, 0.9, 0.9, 1.0, 1.6, 1.1, 1.2, 1.2, 1.2, 0.9, 0.9,
1.1])
#get the mean and standard deviations:
xmean=xi.mean()
ymean=yi.mean()
xstd=xi.std(ddof=1)
ystd=yi.std(ddof=1)
#calculate the covariance and correlation:
covxy=((xi-xmean)*(yi-ymean)).sum()/(xi.size-1)
corxy=covxy/xstd/ystd
#make some text with the results
text=''
Covariance: {:.2f}
Correlation: {:.1f}'''.format(covxy, corxy)
#plot these as a scatter plot, including the text
pl.scatter(xi, yi)
pl.xlabel('x')
pl.ylabel('y')
pl.title("scatter plot of x and y")
pl.text(0.95,1.5,text,fontsize=14)
pl.show()
```

5.2: Output

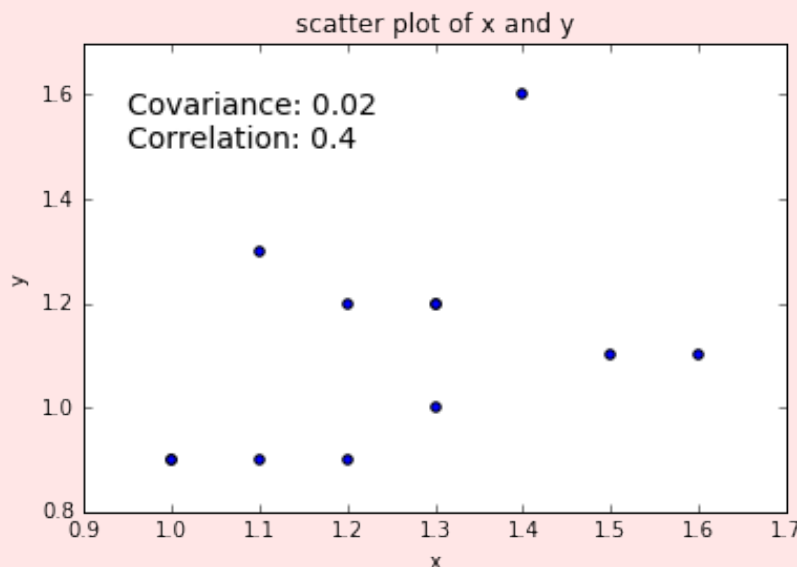


Figure 5.1: Scatter plot of x and y values for different measurements (example 5-2), showing the correlation between the two quantities.

The above code determined the covariance to be 0.02 and the correlation to be 0.42. The values of x and y are thus correlated, and on average, a bigger value of x will result in a larger value of y . A scatter plot of x and y is shown in Figure 5.1, highlighting the trend in the two correlated variables.

The correlation factor is a measure of whether two quantities appear to be related in some manner. For example, a positive correlation between x and y indicates that if one measures a high value of x , then one is also likely to measure a high value of y . This does not mean that a high value of x *causes* a high value of y . “Correlation does not imply causation” is a popular slogan in statistics (strongly emphasized in social studies). Measuring the correlation should never be confused with trying to model one variable as a function of the other. For example, it is likely that the length of cats is correlated with their weights, but a simple confirmation of a strong correlation factor in no way implies that the length of cats causes them to be heavier. For example, it could be measured that going to sleep with one shoe on is correlated with having a headache in the morning. It would be wrong to conclude that sleeping while wearing a shoe causes headaches. Perhaps, there is a third variable that leads to both observations, for example, going to bed intoxicated.

We can ask ourselves whether a particular correlation is “significant”, that is, whether there is strong evidence to support a claim that two variables are correlated. If we take purely random measurements of two quantities, x and y , it is unlikely that their correlation factor would come out to be exactly 0. Rather, we would expect that as the number of measurements increases, their correlation would approach 0 as it becomes more precisely defined. The question is thus to quantify the significance of a measured correlation. This can be done by considering the value of the correlation factor that is unlikely to be possible if the measurements are uncorrelated. Specifically, for N measurements, we can determine

a threshold in correlation for which it is unlikely for N uncorrelated measurements to give a correlation that is higher than the threshold. A sample of these “Pearson correlation” threshold values are shown in Table 5.1. For example, if 8 measurements are used to determine the correlation factor, if it is less than 0.621 it is not significant at the 95% level. That is, there is a 5% chance that 8 random uncorrelated measurements would have a correlation factor of 0.621 or bigger.

N	5%	2.5%
4	0.900	0.95
5	0.805	0.878
6	0.729	0.811
7	0.669	0.754
8	0.621	0.707
9	0.582	0.666
10	0.549	0.632

Table 5.1: Threshold values for a correlation determined from N measurements to be significant at the 95% and 97.5% levels. The values corresponds to either a positive or negative correlation. If the sign of the correlation is not important, these correspond to 90% and 95% significance.

5.3 Statistical parameters for a function of a set of measurements

Suppose that we have made N measurements, $\{x_1, x_2, \dots, x_N\}$, of some quantity x , and wish to apply a function, $F(x)$, to those values. This will lead to a set of values for the function, $\{F(x_1), F(x_2), \dots, F(x_N)\}$, for which we can also evaluate statistical parameters such as the mean and variance.

If we assume that the variance of the values of x is small, so that all of the values of x_i are relatively close to their mean, \bar{x} , we can approximate $F(x)$ using the first term of a Taylor series:

$$F(x_i) = F(\bar{x}) + \left. \frac{dF}{dx} \right|_{\bar{x}} (x_i - \bar{x}) \quad (5.11)$$

5.3.1 The mean

The mean value of the values of $F(x)$ is thus given by:

$$\begin{aligned}
 \bar{F} &= \frac{1}{N} \sum_{i=1}^N F(x_i) = \frac{1}{N} \sum_{i=1}^N \left(F(\bar{x}) + \left. \frac{dF}{dx} \right|_{\bar{x}} (x_i - \bar{x}) \right) \\
 &= \frac{1}{N} \left(\sum_{i=1}^N F(\bar{x}) + \sum_{i=1}^N \left. \frac{dF}{dx} \right|_{\bar{x}} x_i - \sum_{i=1}^N \left. \frac{dF}{dx} \right|_{\bar{x}} \bar{x} \right) \\
 &= \frac{1}{N} \left(N F(\bar{x}) + \left. \frac{dF}{dx} \right|_{\bar{x}} \sum_{i=1}^N x_i - \left. \frac{dF}{dx} \right|_{\bar{x}} N \bar{x} \right) \\
 &= F(\bar{x}) + \frac{1}{N} \left(\left. \frac{dF}{dx} \right|_{\bar{x}} N \bar{x} - \left. \frac{dF}{dx} \right|_{\bar{x}} N \bar{x} \right) \\
 \therefore \bar{F} &= F(\bar{x})
 \end{aligned} \tag{5.12}$$

where we used the fact that:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i \rightarrow \sum_{i=1}^N x_i = N \bar{x}$$

Thus, the mean value of $F(x)$ can be found simply by evaluating F at the mean value of x ! Note that this is only valid in the case where the Taylor series is a valid approximation, namely when the values of x are close to their mean value (they have a small standard deviation).

5.3.2 The variance and standard deviation

We can proceed similarly to determine the variance in the values of F :

$$\begin{aligned}
 \sigma_F^2 &= \frac{1}{N-1} \sum_{i=1}^N (F(x_i) - F(\bar{x}))^2 = \frac{1}{N-1} \sum_{i=1}^N \left(F(\bar{x}) + \left. \frac{dF}{dx} \right|_{\bar{x}} (x_i - \bar{x}) - F(\bar{x}) \right)^2 \\
 &= \frac{1}{N-1} \sum_{i=1}^N \left(\left. \frac{dF}{dx} \right|_{\bar{x}} (x_i - \bar{x}) \right)^2 \\
 &= \left(\left. \frac{dF}{dx} \right|_{\bar{x}} \right)^2 \frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2 \\
 \therefore \sigma_F^2 &= \left(\left. \frac{dF}{dx} \right|_{\bar{x}} \right)^2 \sigma_x^2
 \end{aligned} \tag{5.13}$$

and the standard deviation is simply given by the square root:

$$\sigma_F = \left. \frac{dF}{dx} \right|_{\bar{x}} \sigma_x \tag{5.14}$$

Thus, in the limit where the first order Taylor series approximation is valid, the standard deviation in the values of $F(x)$ is simply given by the standard deviation of the values of the x_i multiplied by the derivative of F evaluated at the mean value of x .

5.3.3 Multiple variables

Let us suppose that we have a function, $F(x, y)$, of two variables, x_i and y_i . We have made N measurements, $\{x_1, x_2, \dots, x_N\}$ and $\{y_1, y_2, \dots, y_N\}$ and wish to determine the mean for F and its variance, σ_F^2 . We can evaluate $F(x, y)$ at all of the values of x and y , and obtain the mean, $\bar{F}(x, y)$, and variance, σ_F , as we did for the case of a single variable.

We assume that the variances σ_x^2 and σ_y^2 are small and that all of the measurements of x and y are therefore close to their respective means. The value of $F(x_i, y_i)$ evaluated at one of the data points can thus be approximated by a Taylor series of $F(x, y)$ evaluated near \bar{x} and \bar{y} :

$$F(x_i, y_i) \approx F(\bar{x}, \bar{y}) + \left. \frac{\partial F}{\partial x} \right|_{\bar{x}, \bar{y}} (x_i - \bar{x}) + \left. \frac{\partial F}{\partial y} \right|_{\bar{x}, \bar{y}} (y_i - \bar{y}) \quad (5.15)$$

where the partial derivatives are always evaluated at \bar{x} and \bar{y} (we will thus drop the $|_{\bar{x}, \bar{y}}$ as we proceed).

First, the mean value of $F(x, y)$ is given by:

$$\begin{aligned} \bar{F}(x, y) &= \frac{1}{N} \sum_{i=1}^{i=N} F(x_i, y_i) \\ &= \frac{1}{N} \sum_{i=1}^{i=N} \left[F(\bar{x}, \bar{y}) + \frac{\partial F}{\partial x} (x_i - \bar{x}) + \frac{\partial F}{\partial y} (y_i - \bar{y}) \right] \\ &= \left[F(\bar{x}, \bar{y}) \frac{1}{N} \sum_{i=1}^{i=N} 1 \right] + \left[\frac{\partial F}{\partial x} \frac{1}{N} \sum_{i=1}^{i=N} (x_i - \bar{x}) \right] + \left[\frac{\partial F}{\partial y} \frac{1}{N} \sum_{i=1}^{i=N} (y_i - \bar{y}) \right] \\ &= F(\bar{x}, \bar{y}) \end{aligned} \quad (5.16)$$

where the last two terms in the sum are identically zero, since:

$$\begin{aligned} \bar{x} &\equiv \frac{1}{N} \sum_{i=1}^{i=N} x_i \\ \bar{y} &\equiv \frac{1}{N} \sum_{i=1}^{i=N} y_i \end{aligned} \quad (5.17)$$

Thus the mean value, $\bar{F}(x, y)$, is given by evaluating $F(x, y)$ at the mean values, \bar{x} and \bar{y} .

We now evaluate the variance of $F(x, y)$ (and we can take the square root later to get the

standard deviation), by using the definition of the variance:

$$\begin{aligned}
\sigma_F^2 &= \frac{1}{N-1} \sum_{i=1}^{i=N} \left[F(x_i, y_i) - \bar{F}(x, y) \right]^2 \\
&= \frac{1}{N-1} \sum_{i=1}^{i=N} \left[F(\bar{x}, \bar{y}) + \frac{\partial F}{\partial x}(x_i - \bar{x}) + \frac{\partial F}{\partial y}(y_i - \bar{y}) - F(\bar{x}, \bar{y}) \right]^2 \\
&= \frac{1}{N-1} \sum_{i=1}^{i=N} \left[\frac{\partial F}{\partial x}(x_i - \bar{x}) + \frac{\partial F}{\partial y}(y_i - \bar{y}) \right]^2 \\
&= \frac{1}{N-1} \sum_{i=1}^{i=N} \left[\left(\frac{\partial F}{\partial x}(x_i - \bar{x}) \right)^2 + \left(\frac{\partial F}{\partial y}(y_i - \bar{y}) \right)^2 + 2 \frac{\partial F}{\partial x} \frac{\partial F}{\partial y}(x_i - \bar{x})(y_i - \bar{y}) \right] \\
&= \left(\frac{\partial F}{\partial x} \right)^2 \frac{1}{N-1} \sum_{i=1}^{i=N} (x_i - \bar{x})^2 + \left(\frac{\partial F}{\partial y} \right)^2 \frac{1}{N-1} \sum_{i=1}^{i=N} (y_i - \bar{y})^2 \\
&\quad + 2 \frac{\partial F}{\partial x} \frac{\partial F}{\partial y} \frac{1}{N-1} \sum_{i=1}^{i=N} (x_i - \bar{x})(y_i - \bar{y}) \\
&= \left(\frac{\partial F}{\partial x} \sigma_x \right)^2 + \left(\frac{\partial F}{\partial y} \sigma_y \right)^2 + 2 \frac{\partial F}{\partial x} \frac{\partial F}{\partial y} \sigma_{xy}
\end{aligned} \tag{5.18}$$

where we have used the definition of the covariance factor, σ_{xy} :

$$\sigma_{xy} \equiv \frac{1}{N-1} \sum_{i=1}^{i=N} (x_i - \bar{x})(y_i - \bar{y}) \tag{5.19}$$

We can take the square root to get the standard deviation of the values of $F(x_i, y_i)$:

$$\sigma_F = \sqrt{\left(\frac{\partial F}{\partial x} \sigma_x \right)^2 + \left(\frac{\partial F}{\partial y} \sigma_y \right)^2 + 2 \frac{\partial F}{\partial x} \frac{\partial F}{\partial y} \sigma_{xy}} \tag{5.20}$$

which is the origin for the formula that is used for propagating errors in a function of multiple variables (recall equation 4.21). It is important to note when this formula can be used for propagating uncertainties. Namely, to be valid it requires that:

- The values of x_i and y_i have a small standard deviation (that is, a small relative uncertainty)
- The uncertainties in x and y can be thought of as the standard deviations of the corresponding values if multiple measurements were made (this also justifies determining the uncertainties by making repeated measurements)

The formula is trivially extended to the case of more than two variables by simply adding a covariance term for each pair of correlated variables, and the corresponding derivative term for each variable.

If the variables x and y are independent, the covariance factor will be zero, and the formula

simplifies to:

$$\sigma_F = \sqrt{\left(\frac{\partial F}{\partial x}\sigma_x\right)^2 + \left(\frac{\partial F}{\partial y}\sigma_y\right)^2}$$

Again, we see that if the uncertainties are representative of the statistical spread in values that we obtain when repeating measurements, we naturally arrive at a prescription for adding in quadrature.

5.4 The histogram

The histogram is a way to visualize a set of measurements of the same quantity. With repeated measurements, we generally obtain a “distribution” of values and the histogram is a way to visualize that distribution. In example 5-1, we had the following measurements: $x_i = \{ 1., 1., 1.1, 1.1, 1.2, 1.2, 1.3, 1.3, 1.3, 1.4, 1.5, 1.6 \}$ for the value of some parameter, x . One way to visualize these data is to count how many times each value occurs and to make a table, as in Table 5.2

value	frequency
1.0	2
1.1	2
1.2	2
1.3	3
1.4	1
1.5	1
1.6	1

Table 5.2: Frequency of occurrence of each measurement.

We can visualize this table easily using a “histogram”, as in Figure 5.2.

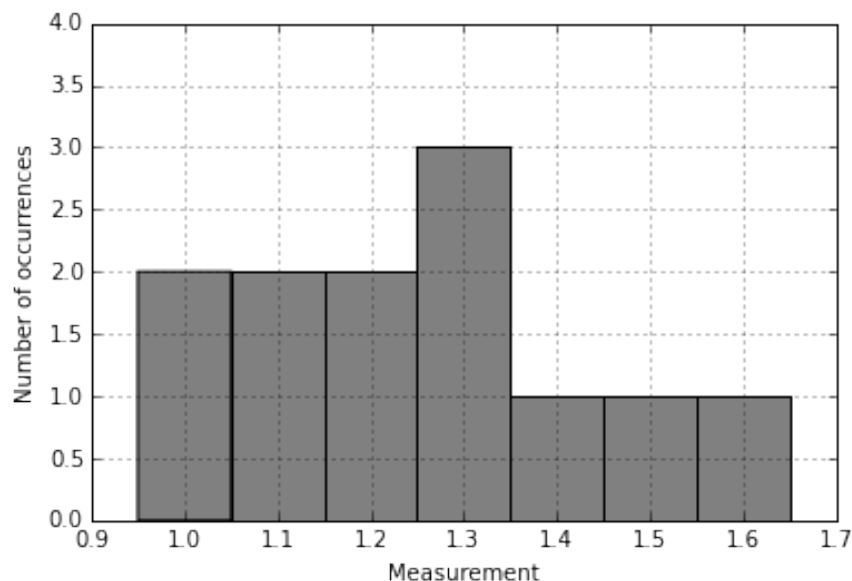


Figure 5.2: A simple histogram of the measurements from Example 5-1 and Table 5.2.

Histograms are drawn as “bar charts”, since measurements at different values on the x-axis are unrelated (it would not be meaningful to treat the columns in Table 5.2 as x and y values for data points and connect them with a line). We call each bar on the chart a “bin”, since it “holds” the number of times each measurement occurred. The histogram is a good way to visualize a distribution of values; in our case, we can see that the value 1.3 comes up the most, and that the values below 1.3 come up more often than the ones above (the distribution is asymmetric about the mode). With only 12 measurements, it is not too difficult to sort the values and come to these conclusions without drawing a histogram; in the case where we have many measurements, histograms become an extremely useful visualization (and analysis) tool.

In the case that we just illustrated, we counted the occurrences of specific values, such as 1.3 or 1.2. If we had measured these numbers with additional precision (more decimals), it is likely that they each would be different. In this case, it would not make sense to count how many times each specific value occurred, since each value would likely occur only once. What we really want to do, is count how many times we get a value that is within a certain range. That is, we define the left and right edge for each bin, and then count how many times we get a value within that range. The histogram in Figure 5.2 was in fact defined to have 7 bins: the first bin from 0.95 to 1.05, the second bin from 1.05 to 1.15, etc, in such a way that the numbers 1.0, 1.1, etc ended up at the center of each bin. In Figure 5.3, we have made a histogram of the same data, but using only three, unequally-sized, bins: from 0.9 to 1.1, from 1.1 to 1.4, and from 1.4 to 1.7, which results in an equally valid histogram of the data.

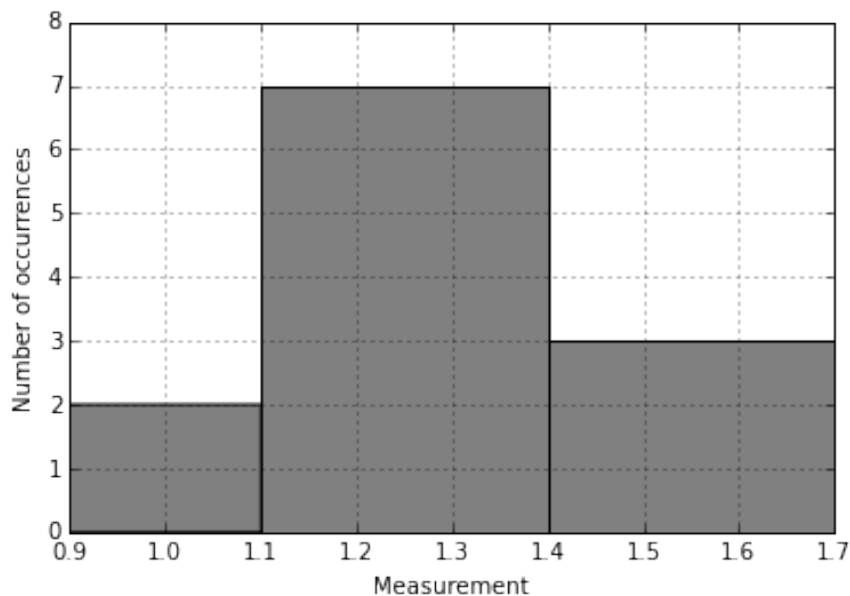


Figure 5.3: The same histogram as in Figure 5.2 but with different size bins.

As you can see, we have some liberty in how we define the “binning” of our histogram. If we make all of the bins really narrow, then they would typically have counts of either 0 or 1 and we would not see much structure in the data. Conversely, if we make the bins very wide, then all the measurements will end up into one bin, and the histogram will not help us to visualize anything either.

In general, we need to play with the binning for plotting data so that the histogram looks reasonable. A good rule of thumb is that the bins should have at least 5 counts in them. Ideally, we should also try to have equally sized bins (rather than varying bin widths as in Figure 5.3), as it makes it easier to analyse the histogram. Refer to section 2.6.2.3 for all of the commands to make histograms in python, as well as the code at the end of this section.

It is sometimes useful to “normalize” a histogram such that the area of the bins of the histogram sum to 1. If each bin, i , contains n_i counts (the bin height) and has a bin width, dx_i , then the area under a histogram, A , is simply given by summing the area of each rectangular bin:

$$A = \sum_{i=1}^{i=N} n_i dx_i \quad (5.21)$$

In order to normalize a histogram, we must thus divide each bin content by A . If all of the bins in the histogram have equal width, dx , (say all $dx_i = dx$), then the area of the histogram is given by:

$$A = dx \sum_{i=1}^{i=N} n_i \quad (5.22)$$

where $\sum_{i=1}^N n_i$ is equal to the number of measurements (the sum of the contents of each bin). In Figure 5.2 from above, we had a fixed bin width of 0.1 and a total of 12 measurements; the area is thus given by $A = 1.2$. We can thus “normalize” that histogram by dividing the content of each bin by 1.2, as in Figure 5.4. In python, histograms can easily be normalized by passing the argument `normed=True` to the `hist()` function.

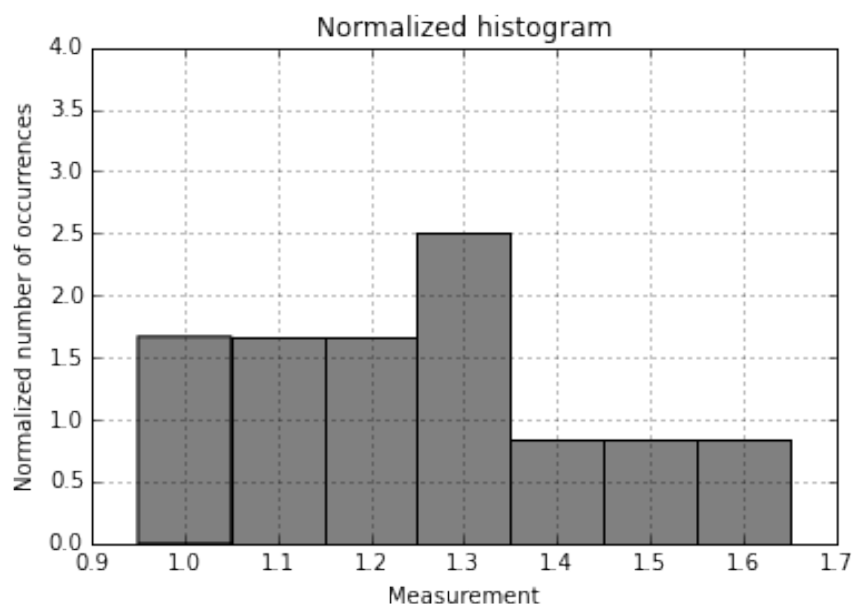


Figure 5.4: The same histogram as in Figure 5.2 but normalized to have an area of 1.

As we will see in the next chapters, a normalized histogram can be thought of as a “probability density function”. If we assume that the data in the histogram are representative of the distribution of measurements that we would get if we repeated the measurements, then the normalized bin contents are representative of the probability of obtaining a certain measurement that would fall in that bin. For example, using Figure 5.4, we could quote that the probability of obtaining a measurement between 1.25 and 1.35 is given by (bin content) \times (bin width) = $2.5 \times 0.1 = 25\%$. In our original 12 measurements, we had obtained the number 1.3 three times, or 25% of the time.

The example python code below shows how to obtain the histograms shown in this section:

Python Code 5.3: Making histograms

```
import numpy as np
import pylab as pl
#Make the histograms in Figures 5-2, 5-3, 5-4
#The data:
xi=np.array([1.1, 1.2, 1.0, 1.3, 1.4,1.5, 1.2, 1.3, 1.3, 1.1, 1.0, 1.6])

#Make a histogram with equal bin width and plot it:
#Here we specify the edges of the bins:
pl.hist(xi, bins=[0.95,1.05,1.15,1.25,1.35,1.45,1.55,1.65], color='gray')
pl.axis([0.9,1.7,0,4])
```



```

pl.grid()
pl.xlabel("Measurement")
pl.ylabel("Number of occurrences")
pl.show()

#Use 3 unequal bins which we specify
#Also get the number of counts in each bin:
n,bins,patches=pl.hist(xi, bins=[0.9,1.1,1.4,1.7], color='gray', normed=False)
pl.axis([0.9,1.7,0,8])
pl.grid()
pl.xlabel("Measurement")
pl.ylabel("Number of occurrences")
pl.show()
print("Bins used:",bins)
print("Counts in each bin",n)

#Use original binning, but normalize the histogram:
pl.hist(xi, bins=[0.95,1.05,1.15,1.25,1.35,1.45,1.55,1.65], color='gray',
        normed=True)
pl.axis([0.9,1.7,0,4])
pl.grid()
pl.xlabel("Measurement")
pl.ylabel("Normalized number of occurrences")
pl.title("Normalized histogram")
pl.show()

```

5.3: Output

```

Bins used: [ 0.9  1.1  1.4  1.7]
Counts in each bin [ 2.  7.  3.]

```

5.5 Comparing statistical data to a model

We now have a few tools that can allow us to compare statistical data to a model (or between different data sets). For example, if one researcher has measured the lengths of 8 month old Canadian sphynx cats and another researcher has measured the lengths of 8 month old American sphynx cats, we can think about whether the Canadian and American cats are statistically similar. We could compare the mean lengths of the two samples using the corresponding errors on the mean to see if they agree. If the means agree within their respective errors on the mean, we could still conclude that Canadian and American cats are different if the shapes of the normalized histograms of the measurements were very different (e.g. you could get the same mean, but vastly different distributions).

5.6 Summary

In this chapter, we covered a few useful metrics to characterize a set of N measurements, $\{x_1, x_2, \dots, x_N\}$, of a single quantity. In particular, we defined the mean:

$$\bar{x} \equiv \frac{1}{N} \sum_{i=1}^{i=N} x_i \quad (5.23)$$

The “sample variance”:

$$\text{var} \equiv \sigma^2 = \frac{1}{N-1} \sum_{i=1}^{i=N} (x_i - \bar{x})^2 \quad (5.24)$$

and the “sample standard deviation”:

$$\sigma \equiv \sqrt{\text{var}} \quad (5.25)$$

We also introduced the error on the mean:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{N}} \quad (5.26)$$

If we have a set of simultaneous measurement of two quantities, $x_i = \{x_1, x_2, \dots, x_N\}$ and $y_i = \{y_1, y_2, \dots, y_N\}$, the covariance, σ_{xy} , is defined to be:

$$\sigma_{xy} \equiv \frac{1}{N-1} \sum_{i=1}^{i=N} (x_i - \bar{x})(y_i - \bar{y}) \quad (5.27)$$

where \bar{x} and \bar{y} are the sample means of the measurements of x and y , respectively. The correlation factor, ρ_{xy} is defined to be:

$$\rho_{xy} \equiv \frac{\sigma_{xy}}{\sigma_x \sigma_y} \quad (5.28)$$

where σ_x and σ_y are the sample standard deviations of the measurements of x and y , respectively. The correlation is a measure of whether the quantities x and y tend to be simultaneously big or small when measured.

We then considered the statistical properties of a function $F(x, y)$, when evaluated for a set of measurements x_i and y_i . We found that, if the values of x and y had small standard deviations, then the mean and standard deviation of the values of $F(x_i, y_i)$ were given by:

$$\bar{F} = F(\bar{x}, \bar{y}) \quad (5.29)$$

$$\sigma_F = \sqrt{\left(\frac{\partial F}{\partial x} \sigma_x\right)^2 + \left(\frac{\partial F}{\partial y} \sigma_y\right)^2 + 2 \frac{\partial F}{\partial x} \frac{\partial F}{\partial y} \sigma_{xy}} \quad (5.30)$$

which is the basis of the error propagation formula presented in the previous chapter. In order for the formula to be valid for error propagation, the errors in x and y must be well modelled by the standard deviation of the corresponding values, and the uncertainties must be small enough so that the first order Taylor series used to derive the formula is a valid approximation.

We finished by introducing the histogram as a way to visualize the distribution of measurements of a single quantity. We saw that we have liberty in how we define the location and sizes of the bins. We saw that by normalizing the area of the histogram, we could approximate a probability density function for the data.

6

Statistics - Distributions

In the previous chapter, we considered a sample of data that consisted of measurements of a certain quantity (e.g. cat length), and defined a few simple variables to describe that sample (e.g. mean, variance). Those were general quantities that can be applied to any sample of measurements.

In many physical situations, the data will follow particular “distributions”. That is, if one makes a histogram of the measurements, the shape of that histogram will be well described by a function. Certain physical processes lead to very specific distribution functions, and we will examine two of those in this chapter (the binomial and the Poisson distributions).

6.1 The binomial distribution

The binomial distribution is used to describe the result of an experiment that can go in one of two ways (for example, flipping a coin can be either heads or tails). Very generally, a lot of processes can be thought of as binomial even if there are more than two possible outcomes. If you are interested in the number 5 coming up when rolling a single dice, even though there are 6 possible outcomes, you can think of it as a binomial problem (you either get 5 or you do not).

The binomial distribution describes **how often we will get k successes in N independent trials, when we expect pN successes**. We call p the probability of getting the “successful” outcome”, and reserve a more formal description of probability for a later chapter. For example, if we have a coin and we are interested in the number of times that we get “heads”, we can think of k as the number of times we get heads in N coin tosses. We can think of p as the fraction of times we get heads (either that we expect or that we measured). If we have a biased coin that, on average, gives 70% heads, then $p = 0.7$ which can be measured by tossing the coin many times and measuring the fraction of time that we get heads (this would be the “frequentist” definition of the probability, p).

Example 6-1: For an unfair coin that has a $p = 0.7$ of being heads on any given toss, what is the probability of obtaining exactly 2 heads in 3 coin tosses?

If we label each toss by H (heads) or T (tails), then there are three possible sequences of 3 tosses that give two heads: HHT, HTH, and THH. The probability of obtaining the first sequence (HHT) is given by:

$$P_{HHT} = pp(1 - p)$$

p for the first toss, multiplied by p for the second toss and $(1 - p)$ for the third toss. The other two sequences (HTH and THH) have the same probability. We only care about obtaining two heads in three tosses (not the order), so the probability of obtaining two heads in 3 tosses is simply the sum of the (identical) probabilities of obtaining each sequence:

$$\begin{aligned} P_{2H} &= P_{HHT} + P_{HTH} + P_{THH} = 3pp(1 - p) \\ &= 3 \times 0.7^2 \times 0.3 = 0.441 \end{aligned}$$

Thus, there is a 44.1% chance that with this biased coin, we would obtain exactly 2 heads in 3 tosses.

A general formula for the probability of obtaining k successes in N trials, when the probability of a success is p , can easily be derived. If there are k successes, then there will be $N - k$ failures. One possible way to obtain this is to have the first k trials be successes, and the next $N - k$ trials be failures. The probability of that particular sequence would be:

$$P_{\text{success first}} = ppp \dots (1 - p)(1 - p) = p^k(1 - p)^{N-k}$$

However, we do not care about the particular order of the successes, so we must think about all of the possible sequences in which we can have k successes in N trials. This is found by the combinatorial formula for “N choose k”:

$$\binom{N}{k} = \frac{N!}{k!(N - k)!}$$

and corresponds to the number of ways of choosing the k “positions” of the successes in a sequence of N trials. The total probability of obtaining exactly k successes in N trials is thus given by multiplying the probability of a single sequence (e.g. $P_{\text{success first}}$) by the number of possible sequences:

$$P^{\text{binom}}(k, N, p) = \frac{N!}{k!(N - k)!} p^k (1 - p)^{N-k} \quad (6.1)$$

which is called the **binomial probability** mass function¹.

¹Since this is a function of a discrete variable, k , we call it a “probability mass function” instead of a “probability density function”.

Since this represents a probability, it naturally follows that:

$$\sum_{k=0}^{k=N} P^{binom}(k, N, p) = 1 \quad (6.2)$$

That is, the sum of the probabilities of all possible outcomes (from $k = 0$ to $k = N$ successes) must equal 1. We say that the probability mass function is normalized.

Let us suppose that we run a food stand that sells three different kinds of poutines: a classic poutine, a poutine with pulled pork, and a vegetarian poutine (with vegetarian gravy). Our historical data show that 70% of customers purchase a classic poutine, 20% of customers purchase the pork poutine, and 10% purchase the vegetarian option. It is one hour before closing time and we are running low on pork; we only have enough pork for 5 poutines and want to know if we should send someone to buy more. From our previous data, we also know that in the hour before closing, we should expect about 15 customers. Thus, we can ask ourselves, what is the probability that we will get $k = 5$ customers out of $N = 15$ customers that order pork poutines in the next hour? Using the binomial distribution, we can easily estimate that quantity to be:

$$P(k = 5, N = 15, p = 0.2) = \frac{N!}{k!(N - k)!} p^k (1 - p)^{N - k} = 0.103$$

There is thus a 10.3% chance that we will sell exactly 5 pork poutines to the next 15 customers.

However, this is not quite the information that we need in order to know if we should purchase more pork. We really want to know what the chances are of the next 15 customers purchasing 6 or more pork poutines, in which case we would not have enough pork. So we really need to add the probabilities for people to buy 6 or more pork poutines, to get the probability that we do not have enough pork:

$$P_{Not\ enough} = P(k = 6, N = 15, p = 0.2) + P(k = 7, N, p) + \cdots + P(k = 15, N, p)$$

which may be tedious to calculate by hand. An equivalent calculation would be to sum the probabilities of people buying 5 or less poutines, giving the probability that we will have enough pork:

$$P_{Have\ enough} = P(k = 0, N = 15, p = 0.2) + P(k = 1, N, p) + \cdots + P(k = 5, N, p)$$

where it should be clear that $P_{Not\ enough} + P_{Have\ enough} = 1$.

Fortunately, we do not have to calculate this by hand and can use a simple python program. The code below shows how to calculate the sum, but also illustrates that there are built-in functions to calculate these sums, as they come up very often. The `cdf()` functions sums the probabilities of getting k or less, and the `sf` function, which is 1-cdf, gives the probability of getting k or higher.

Python Code 6.1: Evaluating binomial probabilities

```

#The binomial distribution
import numpy as np
import pylab as pl
import scipy.stats as stats #for binomial distribution

N=15 # number of customers (trials)
p=0.2 # probability of ordering a pork poutine

print("Prob of exactly 5 pork poutines: {:.3f}".format(stats.binom.pmf(5,N,p)
))

#An array to hold all possible outcomes:
allOutcomes=np.arange(0,N+1)
#An array to hold the probabilities corresponding to each outcome:
probOutcomes=stats.binom.pmf(allOutcomes,N,p)

#Can sum the probabilities (or use sf and cdf):
print("Prob of 6 or more (sum): {:.3f}".format(probOutcomes[6:].sum()))
print("Prob of 6 or more (sf): {:.3f}".format(stats.binom.sf(5,N,p)))
print("Prob of 5 or less (sum): {:.3f}".format(probOutcomes[:6].sum()))
print("Prob of 5 or less (cdf): {:.3f}".format(stats.binom.cdf(5,N,p)))

#Make a plot of the probability of each outcome
pl.plot(allOutcomes,probOutcomes,'o',color='black')
pl.xlabel("Number of pork poutines ordered (k)")
pl.ylabel("Prob of ordering")
pl.title("Binomial prob. distribution for N=15, p=0.2")
pl.axis([-1,16,0,0.3])
pl.grid()
pl.show()

```

6.1: Output

```

Prob of exactly 5 pork poutines: 0.103
Prob of 6 or more (sum): 0.061
Prob of 6 or more (sf): 0.061
Prob of 5 or less (sum): 0.939
Prob of 5 or less (cdf): 0.939

```

and we can see that there is 94% chance that we do not need to purchase more pork (the probability of having 5 or less pork poutine orders is 94%). In this case, we would probably take the 6% chance that we will not have enough pork poutines.

We also used the code to create a graph of the probability of k outcomes as a function of k for a fixed value of $N = 15$ and $p = 0.2$, which is shown in Figure 6.1. We call this plot a “distribution” as it describes the distribution of outcomes that we expect. That is, given the values of N and p for our situation, we can use Figure 6.1 to model how many people purchase pork poutines in the hour preceding the closure of our poutine shop over many nights. For example, we would expect that 23% of the time, 2 pork poutines will be ordered in the last hour. The most likely number of pork poutines to be ordered in the last hour is

3, which is equal to pN (0.2×15).

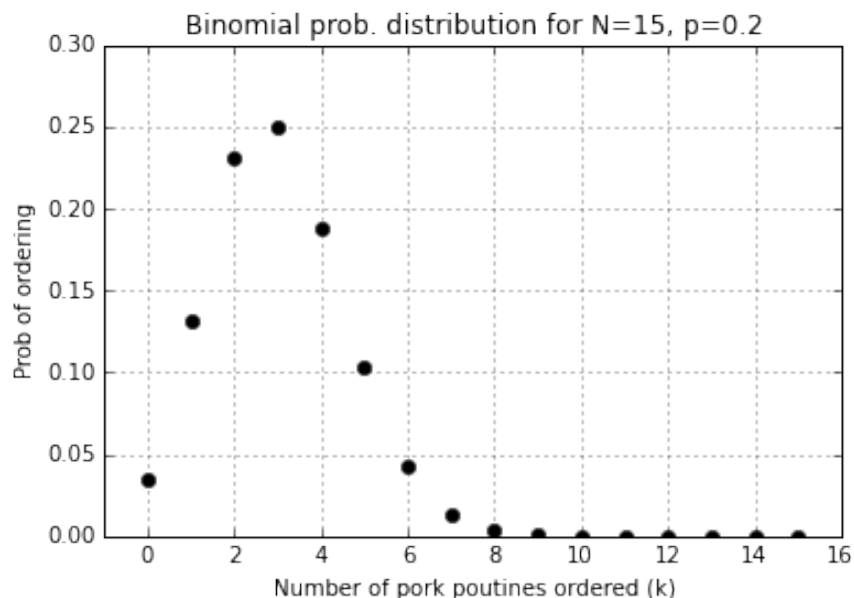


Figure 6.1: Binomial distribution for $N = 15$ and $p = 0.2$.

We can also turn the problem around and test the hypothesis that customers in our poutine store can be modeled by the binomial distribution. To do this, we could collect data to measure, over many nights, the number of customers that bought pork poutine in the hour preceding the store closing. In order to keep things simple, we should measure the number of pork poutines bought by the last 15 customers, rather than in the last hour (since we would not expect exactly 15 customers in the last hour every time). The data could look like that in table 6.1.

Number of pork poutines ordered by last 15 customers
3
2
0
0
3
2
...

Table 6.1: Sample data on pork poutine orders by the last 15 customers in the store over many nights

We can plot a normalized histogram of these numbers and overlay the binomial distribution that we obtained in Figure 6.1 to see if they agree. This is done in Figure 6.2 where Table 6.1 was generated with 100 lines (simulating 100 nights of data taking).

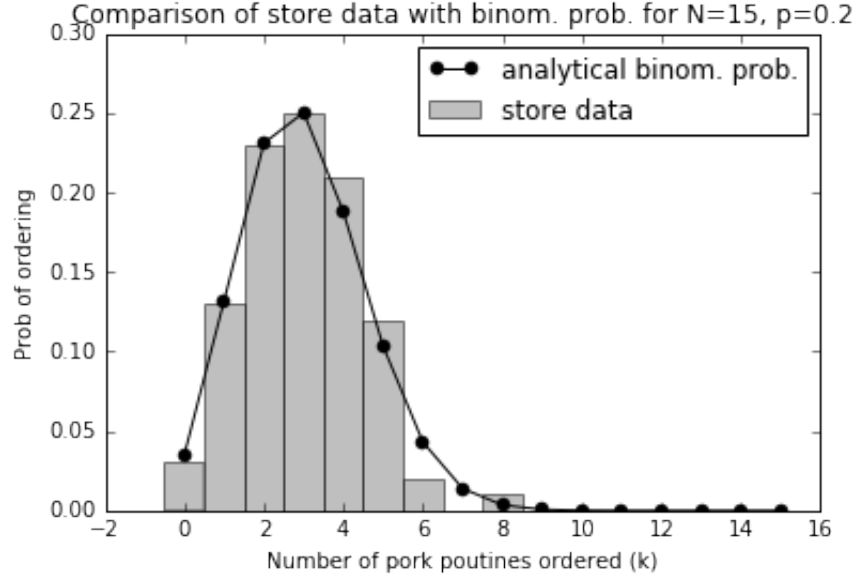


Figure 6.2: Binomial distribution for $N = 15$ and $p = 0.2$ overlaid with simulated data from 100 measurements of the number of customers out of that bought a pork poutine (simulated with a binomial distribution with $p = 0.2$).

We do not yet have the tools to quantitatively conclude whether the distribution of the data (the histogram) agrees with the predicted distribution from our binomial distribution model, but looking by eye, it does seem that the binomial distribution is a good model for the histogram of the data. As we have already seen, the histogram itself, when normalized, can be thought of as a probability distribution. With no model, we could generate the histogram using the data, and then use that histogram to make statements about the probability of having between, say, 3 and 4 pork poutine orders from the last 15 customers.

6.1.1 Statistical properties of the binomial distribution

The mean of measurements that follow the binomial distribution, \bar{k} , for N trials with probability p of success is given by:

$$\bar{k} = \sum_{k=0}^{k=N} k P^{binom}(k, N, p) = Np \quad (6.3)$$

and their variance is given by:

$$\sigma_k^2 = \sum_{k=0}^{k=N} (k - \bar{k})^2 P^{binom}(k, N, p) = Np(1 - p) \quad (6.4)$$

If one has measured k and N , then the mean value of p and its standard deviation are given

by:

$$\bar{p} = \frac{k}{N} \quad (6.5)$$

$$\sigma_p = \frac{\sigma_k}{N} = \sqrt{\frac{p(1-p)}{N}} \quad (6.6)$$

which is the origin for the prescription for specifying the error in a fraction. Note that this holds only if p is not too close to 0 or to 1, and the number k is not too small ². A more robust (and numerical) approach to determining p and its standard deviation are given in section 9.3.1.

A simple way to test if a set of measurements are binomially distributed is to calculate their mean and variance to see if they agree with that expected from the binomial distribution.

Example 6-2: In a class of 20 students, we have counted over 15 different lectures how many of the students checked their smart phone instead of paying attention to class. Over the 15 different lectures, the following number of students checked their phone: {8,10,7,6,5,9,5,4,4,5,6,4,5,7,6}.

For example, in the first lecture, 8 out of 20 students checked their phone, in the second lecture, 10 students checked their phone, etc. Use some simple statistics to determine whether the number of students checking their phone during a given lecture is consistent with being binomially distributed.

We wish to check that the number of students who check their phone follows a binomial distribution. In each lecture, we have $N = 20$ students and we thus wish to see if we can predict, on average, how many of those students will check their phone. We first need to determine the probability of checking their phone, p , which we must estimate from the data. We can estimate p by averaging the fraction of students that check their phone at each lecture. We can then compare the mean and variance of the data with what we expect for a binomial distribution with the value of p that we estimated. We can also compare the histogram of the data with the analytical prediction from the binomial probability mass function.

This is shown in the following python code:

Python Code 6.2: Evaluating binomial quantities

```
#Example 6-2
import numpy as np
import scipy.stats as stats
#N is the number of students
N=20
#put the data into an array, this is the number, k, during each
```

²This formula holds for the case when the binomial probability for k approaches a normal distribution, namely when the mean value of k is relatively large.

```

    experiment
    xi=np.array([8,10,7,6,5,9,5,4,4,5,6,4,5,7,6])
    #get the average (mean) value of xi/N, which is our estimate of p
    p=np.mean(xi/N)
    print("p is {:.2f}".format(p))
    #We can now compare the mean and variance of the data with what we
    expect
    #for the binomial distribution:
    print("data mean: {:.1f}, data variance: {:.1f}".format(xi.mean(), xi.var(
        ddof=1)))
    print("binom. mean: {:.1f}, binom. variance: {:.1f}".format(N*p, N*p*(1-p
        )))

    #Let's also compare this using a histogram
    binedges=np.linspace(-0.5, N+0.5, N+2)
    pl.hist(xi, bins=binedges, normed=True, color='grey', alpha=0.5, label='data',
        )
    #An array to hold all possible outcomes:
    allOutcomes=np.arange(0, N+1)
    #An array to hold the probabilities corresponding to each outcome:
    probOutcomes=stats.binom.pmf(allOutcomes, N, p)
    pl.plot(allOutcomes, probOutcomes, 'o-', color='black', label='analytical
        binom. prob.')
    pl.xlabel("number of students checking phone")
    pl.ylabel("Prob of checking phone")
    pl.title("Comparison of data and binom. prob. for N={}, p={:.2f}".format
        (N, p))
    pl.legend()
    pl.show()

```

The output is:

6.2: Output

```

p is 0.30
data mean: 6.1, data variance: 3.4
binom. mean: 6.1, binom. variance: 4.2

```

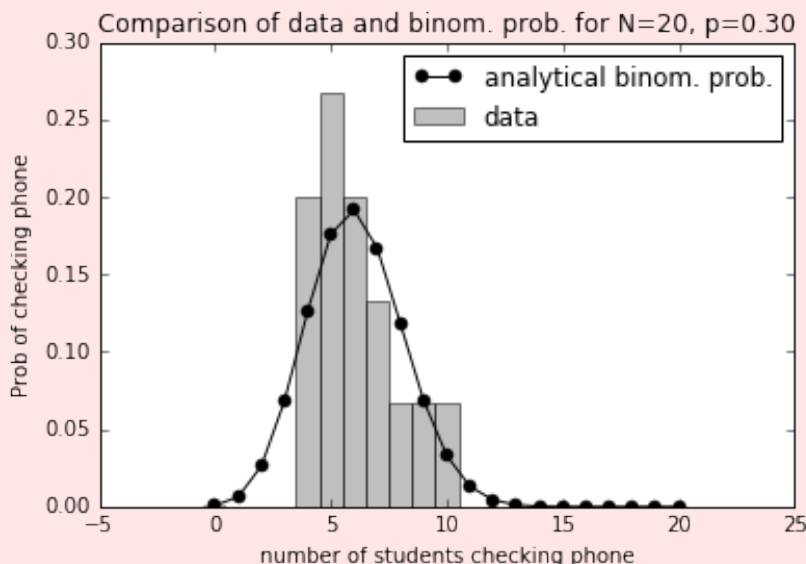


Figure 6.3: Binomial distribution for $N = 20$ and $p = 0.3$ overlaid with the data of 15 lectures where the number of students (out of 20) checking their phone is plotted.

The mean from the data and that predicted for a binomial distribution agree exactly, and the variances are difficult to compare without uncertainties, although they appear to be consistent. We should not be surprised that the mean from the binomial prediction is exactly equal to the mean from the data, since we used the data to evaluate p in such a way that Np is precisely equal to the mean of the data!

Figure 6.3 shows a comparison of the data plotted into a normalized histogram and the prediction from the binomial distribution.

In order to make a definitive statement, we would need more data to see if the variance of the data converges to that predicted for a binomial distribution and to see if the histogram of the data approaches the known distribution. If we had other competing models that also predicted a variance, we could see which model predicts the closest variance. Ultimately, it is very difficult to confirm if data come from a specific distribution if there are few data points.

6.2 The Poisson distribution

The Poisson distribution is used to predict **the probability of observing k successes when we expect n** . One can show that the Poisson distribution is the limiting case of the binomial distribution when N is very large and p is very small, but Np (the mean of the binomial distribution) is finite (see Figure 6.5). The n in the Poisson distribution is equal to Np from the binomial distribution, which is the mean of the binomial distribution. The Poisson probability mass function is given by:

$$P(k, n) = \frac{n^k e^{-n}}{k!}$$

and is much easier to calculate than the binomial distribution. The Poisson distribution is said to apply to “rare processes”, as the probability, p , of a success occurring is small. Note that the Poisson distribution only has one parameter, n , the expected number of successes.

One common example is to predict the expected number of radioactive decays in a period of time, t , for a given radioactive decay process with “time constant” λ . Generally, we would have a large number of atoms, N , and the probability of a particular atom to decay, p , is very small. However, the average number of decays in a certain time interval, Np is finite. We could model this with a binomial distribution, but the Poisson distribution is in this case a very good approximation, and much easier to calculate. In fact, for a process with radioactive decay, N would be of order Avogadro’s number 10^{23} , and taking the factorial of such a large number (as in 6.1) is not practical.

Recall, the radioactive decay law, which tells us how many un-decayed atoms we should have at time t , $N(t)$, if we started with N_0 at time $t = 0$:

$$N(t) = N_0 e^{-\lambda t} \quad (6.7)$$

The rate of decays (how many decays we expect per unit time) is simply given by the time derivative (in this case we take the absolute value since we are interested in the rate of change of N and do not explicitly care that it is decreasing):

$$\left| \frac{dN}{dt} \right| = \lambda N_0 e^{-\lambda t} = \lambda N \quad (6.8)$$

In a given time interval, T , we thus expect an average of n decays given by:

$$n = \lambda N T \quad (6.9)$$

n is thus our expected number of decays in a period of time T for a radioactive decay constant λ given that we have N un-decayed atoms. We can now use the Poisson probability mass function to evaluate the probability of getting k decays in that period of time.

Example 6-3: A typical germanium detector is made of ultra pure germanium (100% germanium with a weight per atom of 72.64 atomic mass unit) and has a mass of 1 kg. We are interested in using a single germanium detector to observe a rare form of radioactive decay called “double-beta decay”. The decay only occurs for the isotope ^{76}Ge which has a 7.4% natural abundance. The half-life of the decay is very long and has been determined to be 1.8×10^{21} years. Due to natural radioactivity in our laboratory, we have a background rate in our germanium detector of 10 counts per month, so we need to make sure that we expect a rate for the radioactive decay that is well above that in order to measure it and justify the expense. What is the probability of observing exactly 15 double-beta decays in our detector if we operate it for 1 month? What is the probability that we will see 15 or more events from double-beta decay?

First we must find out how many ^{76}Ge are present in our 1 kg detector. This is given by using the average mass of a germanium atom (72.63 amu) and the natural abundance.

$$N_{^{76}\text{Ge}} = 0.074 \frac{1000 \text{ g}}{72.63 \text{ amu} \times 1.66 \times 10^{-24} \text{ g/amu}} = 6.14 \times 10^{23}$$

We must then convert the half-life to a decay rate. The half-life is the time that it takes for half of the atoms to decay, so the radioactive decay law gives:

$$\begin{aligned} N(t) &= N_0 e^{-\lambda t} \\ \therefore \frac{1}{2} N_0 &= N_0 e^{-\lambda t_{1/2}} \\ \therefore \lambda &= \frac{\ln(2)}{t_{1/2}} = 3.85 \times 10^{-22} \text{ yr}^{-1} \end{aligned}$$

The expected number of decays in a 1 month period ($T = 1/12$ years) is given by:

$$\begin{aligned} n &= N_{^{76}\text{Ge}} \lambda T \\ n &= 6.14 \times 10^{23} \times 3.85 \times 10^{-22} \text{ yr}^{-1} \frac{1 \text{ yr}}{12} \\ n &= 19.7 \end{aligned}$$

Evaluating the probability from the Poisson distribution with $n = 19.7$ and $k = 15$ gives 5.6%.

In order to get the probability of observing 15 or more events, we must sum the Poisson probabilities for $k = 15, 16, \dots$ to infinity. The infinite sum can be avoided by instead summing the probabilities for $k = 0, 1, \dots, 14$ and subtracting that from 1. The result is that the probability of getting 15 or more decays in 1 month is 88.3%.

We thus conclude that we have a good (88.3%) chance to detect double-beta decay given the requirement that we want to see at least 15 counts so that our signal is distinguishable from the background^a. The python code below shows how to calculate everything and uses the function `sf` to sum the probabilities of obtaining $k = 15$ counts or more:

Python Code 6.3: Computing probabilities for the Poisson distribution

```
import numpy as np
import scipy.stats as stats # to use Poisson
from math import * # to use log

#Need the number of Ge atoms in 1 kg of germanium
mGe=72.63 #amu
amu=1.66054e-24 #amu to grams
abundance=0.074
mexp=1000 #total mass of Ge in grams
NGe=abundance*mexp/(amu*mGe)
print("There are {:.2e} 76Ge atoms".format(NGe))
```

```

#from half-life, need decay rate
thalf=1.8e21 #half-life in years
lam=log(2)/thalf #lambda, but cannot call it lambda as that is a python
keyword
rate=NGe*lam
print("Lambda is {:.2e}".format(lam))
print("The decay rate is {:.1f} per year".format(rate))
print("The decay rate is {:.1f} per month".format(rate/12))

n=rate/12 #The expected number of decays in 1 month

print("The prob. of observing exactly 15 decays in one month is: {:.3f}"
      .format(stats.poisson.pmf(15,n)))
#For sf, need k -> k-1 if we want to include k:
print("The prob. of observing 15 or more decays in one month is: {:.3f}"
      .format(stats.poisson.sf(14,n)))

```

6.3: Output

```

There are 6.14e+23 76Ge atoms
Lambda is 3.85e-22
The decay rate is 236.3 per year
The decay rate is 19.7 per month
The prob. of observing exactly 15 decays in one month is: 0.056
The prob. of observing 15 or more decays in one month is: 0.883

```

^aIn reality, achieving a background rate of 10 counts per month in a germanium detector is very challenging, and it is not nearly as easy to observe double-beta decay.

Figure 6.4 shows the Poisson distribution as a function of k plotted for different values of the expected number of successes, n . For low values of n , the distribution is asymmetric, but as n becomes bigger, the distribution becomes more symmetric and eventually will approach the normal distribution (discussed in Chapter 7).

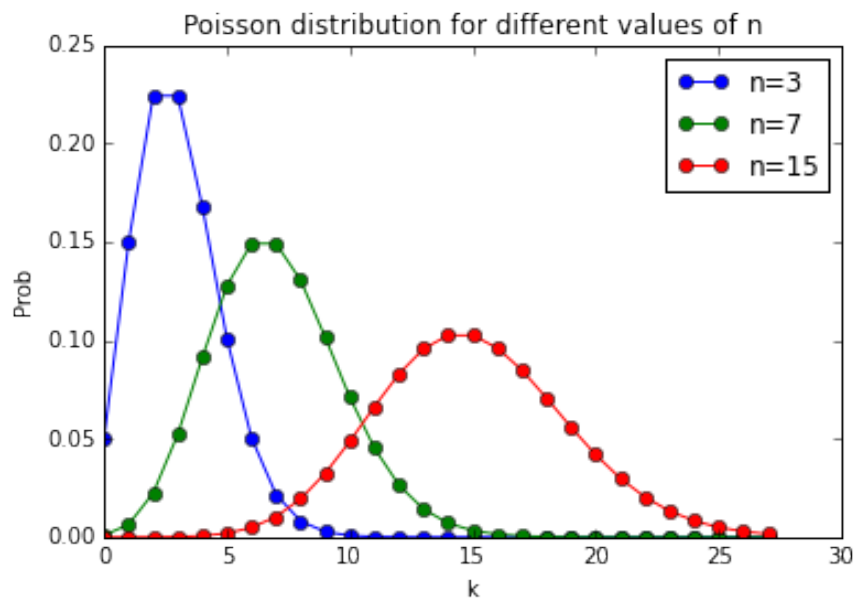


Figure 6.4: Poisson distribution for different values of the expected number of successes.

Figure 6.5 shows a comparison of the Poisson distribution for $n = 3$ along with the binomial distribution with different values of N and p such that $Np = 3$. As p gets smaller, the binomial distribution is indeed seen to approach the Poisson distribution.

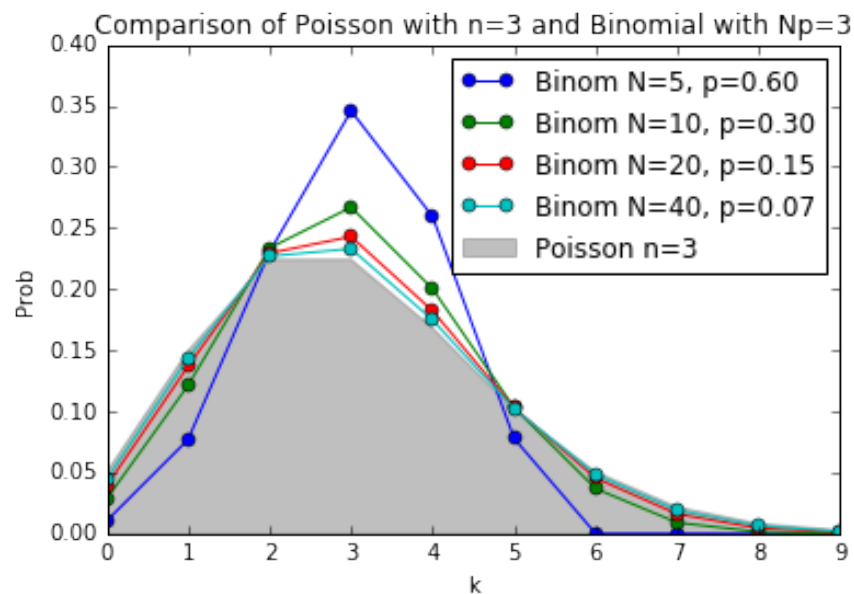


Figure 6.5: Comparison of the Poisson distribution with $n = 3$ and the binomial distribution for $Np = 3$, using different combination of N and p . As p gets smaller, the binomial distribution converges to the Poisson distribution.

The Poisson distribution is extremely useful and appears every time that one carries out experiments that involve counting. This is because one can phrase any experiment where

something is to be counted into a problem of asking: “what is the probability that I count k occurrences when I expect n ?”.

For example, what is the chance that 14 babies will be born at the hospital tonight, when on average 10 are born per night? Or, what is the probability that we will have less than 3 accidents this week in Kingston when we usually have 6 accidents per week? Again, all of these scenarios could be modeled by the binomial distribution, but they correspond to cases where p is very small and where the Poisson distribution is much easier to use (the probability of a specific mother giving birth on a given night is relatively small, as is the probability of a specific car getting into an accident).

It can sometimes be difficult to conceptualize a counting experiment with the concept of a probability in a binomial process being small. Usually, the way to think about it is that there is a small probability for a specific item out of many to be the one that is counted, but a finite number of items that will be counted.

When we count something that has intrinsic randomness (as the examples above), the Poisson distribution can help us to understand the uncertainty on a measured number of counts. Since we understand that when we count something, the result we get follows the Poisson distribution, we can very precisely quantify the probability of getting a number that is bigger or smaller than what we obtained. In fact, we can use the Poisson distribution to define a range in which we are, say, 90% confident that the true value lies.

Example 6-4: Suppose that we spent one week in an all-inclusive vacation in the Caribbean, lounging on the beach and watching tourists get hit on the head by falling coconuts. In one week, we counted that 10 tourists were hit on the head by coconuts. Upon returning to our university, we want to tell our physics professor about our measurement and wish to report with a high degree of confidence (90%) how many tourists get hit by coconuts every week. What range should we quote?

Note that this is a Poisson process, and the probability of a specific tourist getting hit by a coconut is small, but the number of tourists that will get hit is finite.

We want to find a range over which the number of coconut hits per week covers 90% of the probability. For example, we would like to be able to say that we are 90% confident that every week between 8 and 12 tourists get hit.

We only have 1 data point, which is the 10 hits that we counted in the week that we were there. Our “best estimate” for the expected number of hits each week is thus $n = 10$. We can tabulate the Poisson probabilities for $n = 10$:

<i>number of hits</i>	<i>Poisson prob</i>
5	0.038
6	0.063
7	0.090
8	0.113
9	0.125
10	0.125
11	0.114
12	0.095
13	0.073
14	0.052
15	0.035
16	0.022

Table 6.2: Poisson probabilities of the number of tourists getting hit by coconuts in a week for an expected number of 10.

We can use different ways to choose the range; one way is to symmetrically expand the range on either side of 10 and add the probabilities until we reach 90%. The following python code does this:

Python Code 6.4: Computing a 90% confidence interval

```
import numpy as np
import scipy.stats as stats # to use Poisson

n=10 # the expected number of coconut hits
#Sum the probabilities on either side of n until we reach 90%
prob=stats.poisson.pmf(n,n)#the probability of getting exactly n
i=0
while (prob<0.9):
    i=i+1
    #add in the probability at n-i and n+i
    prob=prob+stats.poisson.pmf(n-i,n)+stats.poisson.pmf(n+i,n)

print("Between {} and {}, the probability is {:.1f}%".format(n-i,n+i,100
    *prob))
```

6.4: Output

Between 5 and 15, the probability is 92.2%

Thus, if we observed 10 coconut hits in one week, we can claim with 92.2% confidence that every week, between 5 and 15 people get hit by coconuts!

We can also easily modify this code to get the common 68% confidence level (replace 0.9 in the `while` statement with 0.68), in which case we find that with 73.4% confidence, between 7 and 13 people get hit on the head each week. We could quote that, each week 10 ± 3 people get hit with coconuts.

If you recall, we had prescribed in Section 3.2.3 that the uncertainty on a counting measurement should be the square root of the number of counts, and that this would give a 68% chance that the true number is in the quoted range. We have just verified numerically that this is true, as our square root uncertainty would be $\sqrt{10} = 3.2$, close to what we have found.

6.2.1 Statistical properties of the Poisson distribution

The mean of measurements that follow the Poisson distribution for an expected number of successes, n , is simply n :

$$\bar{k} = \sum_{k=0}^{k=\infty} k P^{\text{Poisson}}(k, n) = n \quad (6.10)$$

and their variance is given by:

$$\sigma_k^2 = \sum_{k=0}^{k=\infty} (k - \bar{k})^2 P^{\text{Poisson}}(k, n) = n \quad (6.11)$$

The standard deviation (the square root of the variance) is thus given by \sqrt{n} . This is the origin of the prescription for the square root uncertainty presented in Section 3.2.3, which as we saw approaches the value of a 68% confidence when n becomes large (for $n = 10$, we saw in example 6-4 that the confidence level was approximately 73%).

We can go back to Example 6-2, where we had counted the number of students that had checked their phone during the lectures. From the data, we found a mean of 6.1 students checking their phone with a variance of 3.4. If the students checking their phones are binomially distributed, then we expected the variance to be 4.2. Now that we know the variance for a Poisson process, we can see if the students are closer to that distribution. The expected number, n , for the Poisson process is also 6.1 (it is equal to the mean of the binomial). The expected variance, if the process of student checking their phones is Poisson, is thus also 6.1. We can see that, in this case, the data (based on their variance) are more consistent with the binomial distribution than the Poisson distribution.

6.2.2 Time between Poisson events

One other interesting property of the Poisson distribution is that if you measure the time interval between two events, the distribution of those times will follow a decaying exponential. That is, you will more often have two Poisson events that follow each other close in

time, rather than having a long time between events. For example, if you look at radioactive decays and measure the time between two decays, you will find that two decays happen in rapid succession more often than with a long time interval. A more tangible example is that airplane crashes seem to appear in bunches rather than spread out regularly over time. This is because airplane crashes are rare (Poisson) events, so it is in fact more likely to have them occur closely in time rather than regularly spread out over the year. A mis-understanding of Poisson statistics also leads to people predicting impending doomsdays when “terrible” events happen in close succession: it is in fact more likely for several terrible events to happen in close succession than evenly spread apart.

This observation is rather perplexing, since the Poisson events are independent of each other and intuitively, we would expect the events not to be able to influence when other independent events happen in time (they don’t!). If we ask ourselves, what the probability is that no events ($k = 0$) will happen in a time interval Δt , we can write:

$$P(k = 0, n) = \frac{n^k e^{-n}}{k!} = e^{-n} \quad (6.12)$$

If the average rate of events per unit time is λ , then in a period Δt , we expect, $n = \lambda \Delta t$, and the probability of no events happening in that period is:

$$P(k = 0, n) = P(k = 0, \Delta t) = e^{-\lambda \Delta t} \quad (6.13)$$

That is, the probability of obtaining a period Δt with no events in it goes down exponentially! In other words, the smaller values of Δt , the time between events, are (exponentially) more likely than longer periods ³.

6.3 Statistical hypothesis testing

We can now apply our understanding of distributions to statistical hypothesis testing. The general idea is that given the result of a measurement, we want to be able to make a quantitative statement about how likely the result is based on a hypothesis. If the result is very unlikely according to the hypothesis, then we would conclude that the hypothesis is not supported by the data.

For example, a pharmaceutical company may claim that they have developed a new pill that can help people to perform very well on IQ tests. As sceptical scientists, we decide to test whether the pill is actually effective and devise the following test: we choose 16 people, give them a first IQ test, then give them the pill, and finally give them a second, different but equivalent, IQ test. Suppose that we find that 10 out of the 16 people have performed “significantly” better on the second IQ test; can we conclude that the pill works?

We need to state this problem into a hypothesis that predicts the distribution of outcomes, and then evaluate the probability of our outcome (10 out 16 people performing better). If the probability is high, then the hypothesis is supported (note that it is never proven).

³For this to give the probability density function for Δt , we would still need to normalize it, giving $P(\Delta t) = \frac{1}{\lambda} e^{-\lambda \Delta t}$

When possible, one should try to make a “null hypothesis”, which is usually easier to model. Typically, the null hypothesis is that “there is no effect”, and the results are the outcome of chance.

In our case, our null hypothesis is that the pill has no effect, or rather that the results are from pure chance. We must thus ask ourselves what is the probability that, out of pure chance, 10 out of 16 participants would perform better on a second IQ test? If we imagine that the IQ tests have many questions, it is unlikely that someone would get exactly the same score twice in a row. It seems reasonable that it is equally likely that they will perform a little better or a little worse on the second test. Let us thus assume that there is a 50% chance that they will perform a little bit better just from pure chance.

We can model our experiment as a binomial problem: we have 16 people (N trials), and we had 10 people do better on the second test (k successes), and under our null hypothesis there is a 50% chance of performing better on the second test (p). The binomial probability, $P^{\text{binom}}(k = 10, N = 16, p = 0.5)$ is easily found to be 12%. There is thus a 12% chance that exactly 10 out of 16 people would do better on the IQ test out of chance.

However, this null hypothesis does not quite cover the possibility that 11 (or 12, or more) people could have done better on the test. The better (and correct) question to ask is, what is the probability that 10 **or more** people out of 16 scored better on the test out of pure chance? We thus need to sum the binomial probabilities corresponding to the possibilities of 10 or more successes. If we sum those probabilities for $k = 10, 11, \dots, 16$, with $N = 16$ and $p = 0.5$, we find that the probability of 10 or more people doing better on the test, out of pure chance, is 23%.

Is 23% a high enough probability for us to support the null hypothesis that the results are from pure chance? Of course, that is an arbitrary question. One way to think about whether a probability is “high” is to ask yourself if you would risk your life at that probability. Would you get in a car if you knew you had a 23% chance of getting into a fatal accident? Probably not (I hope). At 23%, we would generally accept the null hypothesis as plausible, and conclude that the results are consistent with pure chance. In science, a common “bar” is that the null hypothesis must be no more than 5% plausible for it to be rejected, although depending on how “revolutionary” the result, that bar can be much higher. For example, for discovering a new particle in physics, the null hypothesis (there is no new particle) must usually be no more than 0.00006% likely. It is a somewhat accepted practice to call a result “significant” if the null hypothesis is no more than 5% probable, and “highly significant” if the null hypothesis is no more than 1% probable.

Why would we not think about this the other way around, namely that the hypothesis that the pill works is confirmed with 77% probability (100-23), well exceeding our 5% bar? Remember, our model is that there is a 50% chance of doing better on an IQ test simply due to chance. Our model is not “given this pill, there is a $x\%$ chance of doing better on the test”. The 77% probability cannot be interpreted as the probability that the pill works, which would be a completely different hypothesis. The “opposite” of “the results are from

pure chance” is *not* that “the pill works”. Perhaps there are other effects (other than chance) that lead to our result that are not the pill itself (maybe the subjects took the pill with some water, were then better hydrated, and performed better, or it was a placebo effect).

We did not quite analyse our data correctly, since we earlier claimed that 10 of the test subjects performed *significantly* better, not just marginally better. The definition of “significantly” would presumably come from the designers of the IQ test. IQ tests are statistical tests in themselves, so let us assume that if someone gets a score of X , then the test is designed such that one can attribute an uncertainty to the score, δX , such that over repeated tests, there is a 68% chance that a given participant scores in the range $X \pm \delta X$. When we claimed that 10 of the test subjects performed significantly better, we really meant that they performed with a score higher than their original score plus the uncertainty.

To take this into account, we must thus use a different probability than $p = 0.5$ in our null hypothesis that subjects would perform significantly better just out of pure chance. In this case, out of pure chance, there is a 32% (100-68) chance of a participant scoring outside of the range $X \pm \delta X$, and a 50% probability that it is on the higher side of the range. The correct binomial probability to use is thus $p = 0.5 \times 0.32 = 0.16$, and it represents the probability of scoring significantly higher out of pure chance. Evaluating the sum of the probabilities $P^{\text{binom}}(k, N = 16, p = 0.16)$ for $k = 10, 11, \dots, 16$ gives 0.0034%, which is certainly low. It is thus unlikely that the null hypothesis is correct, in other words, it is unlikely that by pure chance 10 or more out of 16 participants would score significantly higher on the IQ test. We would then lean to the conclusion that the pill likely has some effect (but as good scientists, we would want to make sure that any other possible effect have been controlled in the test; for example, that there is no placebo effect from the pill, that the water taken with the pill had no effect, etc.).

The python code below shows how to calculate the probabilities for this example.

Python Code 6.5: Computing binomial probability of null hypothesis

```
import scipy.stats as stats # to use Poisson

#Probability of 10 out of 16 people doing better on IQ test if the
#probability of doing better is 50%:
print("Prob that 10 out 16 people do a little better:{:.2f}".format(stats.
    binom.pmf(10,16,0.5)))
print("Prob that at least 10 out 16 people do a little better:{:.2f}".format(
    stats.binom.sf(9,16,0.5)))

#Probability of 10 out of 16 people doing better on IQ test if the
#probability of doing better is 16%:
print("Prob that 10 out 16 people do significantly better:{:.6f}".format(
    stats.binom.pmf(10,16,0.16)))
print("Prob that at least 10 out 16 people do significantly better:{:.6f}".
    format(stats.binom.sf(9,16,0.16)))
```

6.5: Output

Prob that 10 out 16 people do a little better:0.12

Prob that at least 10 out 16 people do a little better:0.23

Prob that 10 out 16 people do significantly better:0.000031

Prob that at least 10 out 16 people do significantly better:0.000034

Example 6-5: The 2013 Ontario Road Safety Annual Report shows the following number of fatal car collisions by month: {35,17,26,24,39,27,61,44,37,46,42,31}, which add up to 429 for the year. Upon looking at the data, a politician decides that their next campaign will be based on banning driving in July since that is a particularly deadly month for driving in Ontario (61 deaths). Can you support the politician's claim that July is a bad month for driving?

We must first develop a null hypothesis, namely that no month is particularly worse for driving. If the data are consistent with the null hypothesis, then there is little reason to support the politician's claim. We thus need to evaluate the probability of having 61 or more fatal accidents in a particular month under the hypothesis that all months are equal.

In our null hypothesis, we expect the rate of accidents each month to be approximately equal to the average rate. Since we had 429 accidents in the year, we would expect $429/12=36$ accidents per month as the average rate. Given that we expect 36 accidents in a month under the null hypothesis, what is the probability of having 61 or more accidents? This is found using the Poisson distribution, $P^{\text{Poisson}}(k, n)$. In the Poisson distribution, the value of k can go up to infinity, so it is not practical to sum the probabilities from $k = 61, 62, \dots, \infty$. It is easiest to simply use the cumulative probability distribution, which is the sum of the probabilities from $k = 0, 1, \dots, 61$, and then subtract that from 1. Evaluating the cumulative distribution and subtracting it from 1, we find that the probability of having 61 or more accidents, when we expect 36, is 0.0091%, which is certainly small. The null hypothesis is not supported by the data, so we conclude that it is highly likely that some months (July in particular) are more fatal than others on Ontario roads, based on the 2013 data.

As good scientists, we would still want to look at the data from other years to see if they also do not support the null hypothesis. In 2012, there were a total of 505 accidents in the year (an expected 42 accidents per month), and 50 accidents in July. Going through the same exercise, gives the probability for there to be 50 or more accidents in a month, when we expect 42, to be 12.5%, which is orders of magnitude higher than 2013 and well above our 5% bar (if that is the bar that we choose). As you can see, you need to be very careful when making claims based on statistical data, and some arbitrary bar!

The code below shows how to calculate the relevant probabilities in python (using the survival fraction):

Python Code 6.6: Computing Poisson probability of null hypothesis

```
import scipy.stats as stats # to use Poisson
```

```

#Probability of having 61 accidents in a month when we expect 36:
print("Prob of 61 accidents when we expect 36:{:.6f}".format(stats.
    poisson.pmf(61,36)))
print("Prob of at least 61 accidents when we expect 36:{:.6f}".format(
    stats.poisson.sf(60,36)))

#Probability of having 50 accidents in a month when we expect 42:
print("Prob of 50 accidents when we expect 42:{:.4f}".format(stats.
    poisson.pmf(50,42)))
print("Prob of 50 accidents when we expect 42:{:.4f}".format(stats.
    poisson.sf(49,42)))

```

6.6: Output

```

Prob of 61 accidents when we expect 36:0.000039
Prob of at least 61 accidents when we expect 36:0.000091
Prob of 50 accidents when we expect 42:0.0275
Prob of 50 accidents when we expect 42:0.1250

```

6.4 Summary

In this chapter, we introduced the concept of a statistical distribution to model the spread of results that arises from processes that have some randomness associated with them. We introduced the binomial distribution to model processes where we expect k successes after N independent trials, when a given success has a probability of p . The binomial distribution gives the probability of having exactly k successes, and the probability mass function is given by:

$$P^{binom}(k, N, p) = \frac{N!}{k!(N-k)!} p^k (1-p)^{N-k} \quad (6.14)$$

The mean and variance of a quantity that is binomially distributed are given by:

$$\bar{k} = \sum_{k=0}^{k=N} k P^{binom}(k, N, p) = Np \quad (6.15)$$

$$\sigma_k^2 = \sum_{k=0}^{k=N} (k - \bar{k})^2 P^{binom}(k, N, p) = Np(1-p) \quad (6.16)$$

If determining the fraction p using measurements of k and N , the mean and standard deviation of p are given by:

$$\bar{p} = \frac{k}{N} \quad (6.17)$$

$$\sigma_p = \frac{\sigma_k}{N} = \sqrt{\frac{p(1-p)}{N}} \quad (6.18)$$

We then introduced the Poisson distribution to model the probability of obtaining k successes when we expect n . We saw that the Poisson distribution is the limiting case of the binomial distribution when $n = Np$ and p is small (thus the probability of a particular

success is low, but the total number of successes is finite). The Poisson distribution probability mass function is given by:

$$P^{Poisson}(k, n) = \frac{n^k e^{-n}}{k!}$$

The mean and variance of a quantity that is Poisson distributed are given by:

$$\bar{k} = \sum_{k=0}^{k=\infty} k P^{Poisson}(k, n) = n \quad (6.19)$$

$$\sigma_k^2 = \sum_{k=0}^{k=\infty} (k - \bar{k})^2 P^{Poisson}(k, n) = n \quad (6.20)$$

The variance of the Poisson distribution is equal to its mean, n . The standard deviation is thus \sqrt{n} , which is the origin of the square root uncertainty in a counted quantity.

Finally, we introduced the concept of statistical hypothesis testing, which involved determining the probability of a particular outcome given a particular hypothesis. We argued that it is usually easier to model the null hypothesis, that is, to determine the probability of the results occurring out of pure chance. We cautioned against drawing conclusions simply based on the probability of the null hypothesis being below a given bar (5% is often used). Generally, the null hypothesis is of the form: “what is the probability of getting x or more successes out of pure chance?”.

7

Statistics - The Normal Distribution

So far, we have examined two types of statistical distributions, the binomial and the Poisson distributions. We reserved this chapter to cover one of the most important distributions in physics and science: the normal distribution (or gaussian distribution). The assumption that data follow a normal distribution is what leads to most of the rules for error analysis that we have introduced in earlier chapters (e.g. adding in quadrature, 68% confidence levels, etc.), and the normal distribution is thus fundamental in understanding many of the topics in data and error analysis.

The binomial and Poisson distributions are functions of a discrete variable (we used k in the previous chapter as the discrete variable), so we described them using a “probability mass function” (pmf). It only made sense to ask what is the probability of obtaining k heads in N coin tosses if k is an integer. $P^{binom}(k, N, p)$ thus truly represents the probability of obtaining k heads in N tosses. If we sum the probabilities for all of the possible values of k , we get 1, as expected (the probability of having some sort of outcome has to be 1).

The normal distribution is a function of a continuous variable, x , and we thus describe it using a “probability density function” (pdf). The normal distribution is given by the following pdf:

$$P^{norm}(x, \mu_x, \sigma_x) = \frac{1}{\sigma_x \sqrt{2\pi}} e^{-\frac{(x-\mu_x)^2}{2\sigma_x^2}} \quad (7.1)$$

and has two parameters: the mean, μ_x , and the standard deviation, σ_x . We cannot interpret the value of the pdf to give us the probability of obtaining the value x in a given measurement, because x is a continuous variable: we can never measure anything to infinite accuracy, so the probability of getting *exactly* x is always zero. This was not the case for the discrete binomial and Poisson distributions, where one can expect to get exactly a given number of successes. Rather, the pdf for the normal distribution gives us the probability, $P^{norm}(x, \mu_x, \sigma_x)dx$, of obtaining a value that is between x and $x + dx$, where dx is a small increment in x (small in the sense that $P^{norm}(x + dx, \mu_x, \sigma_x)$ is equal to $P^{norm}(x, \mu_x, \sigma_x)$). The probability, $P^{norm}(x, \mu, \sigma)dx$, is the area underneath the curve $P^{norm}(x, \mu_x, \sigma_x)$ between

x and $x + dx$.

Figure 7.1 shows the normal distribution as a function of x for different values of the parameters μ_x and σ_x . The normal distribution is symmetric about the mean μ_x (as evidenced by equation 7.1) and has a width that is proportional to σ_x . The factor $\frac{1}{\sigma_x\sqrt{2\pi}}$ in front of the exponential normalizes the distribution so that the total area under the curve is equal to 1. Just as in the case of the binomial and Poisson distributions, we need the sum of the probabilities of all possible outcomes to be equal to 1. For the normal distribution, we thus require:

$$\sum_{x=-\infty}^{x=+\infty} P^{norm}(x, \mu_x, \sigma_x) dx = 1$$

which of course should be written as an integral since x is a continuous variable:

$$\int_{-\infty}^{+\infty} P^{norm}(x, \mu_x, \sigma_x) dx = 1 \quad (7.2)$$

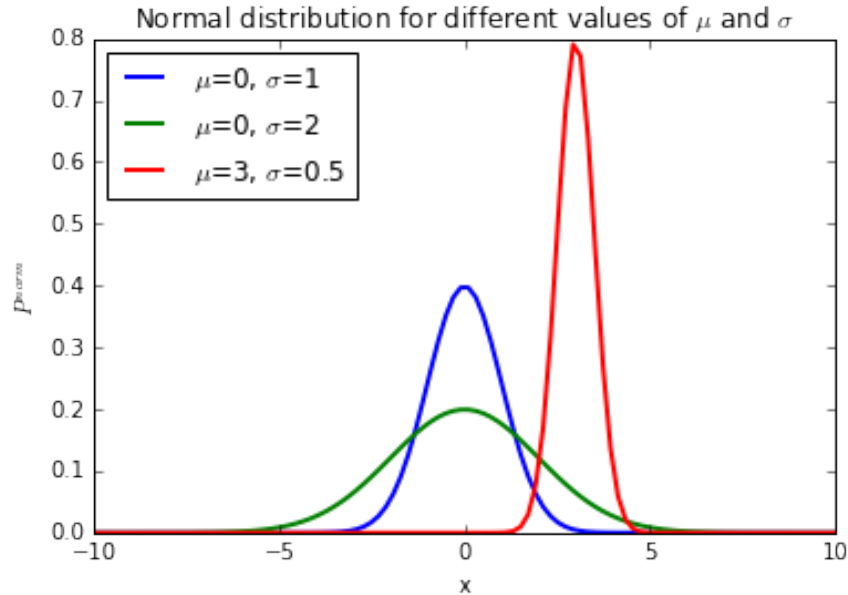


Figure 7.1: Normal distribution for different parameters.

The main reason that the normal distribution is so important is that both the binomial and the Poisson distributions approach the normal distribution when their mean is large. Figure 7.2 shows that as the mean of the binomial distribution increases, the distribution approaches the normal distribution with $\mu_x = Np$ and $\sigma_x = \sqrt{Np(1-p)}$ (given by the mean and standard deviation of the binomial distribution).

Figure 7.3 shows that as the mean of the Poisson distribution, n , increases, the distribution approaches the normal distribution with $\mu_x = n$ and $\sigma_x = \sqrt{n}$. We chose to illustrate the binomial and Poisson distributions with the same mean (6 and 22.5), although the variances are different.

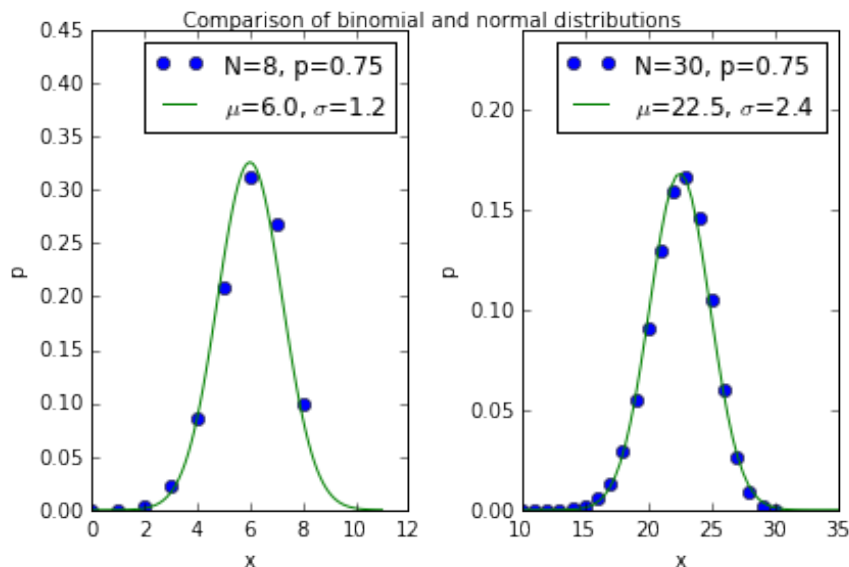


Figure 7.2: Comparison of binomial and normal distributions with different means.

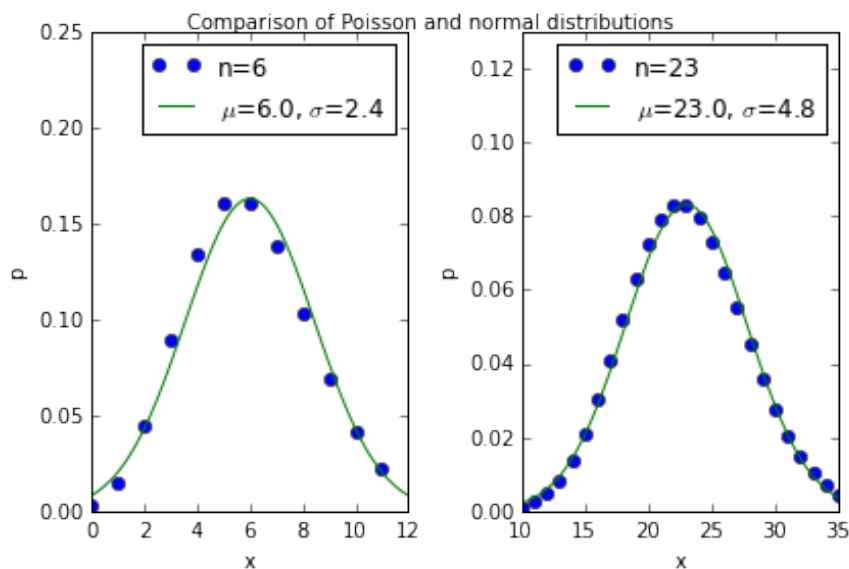


Figure 7.3: Comparison of Poisson and normal distributions with different means.

The normal distribution is much more straightforward to evaluate than the binomial or Poisson distribution, because it is a function of a continuous variable, and it does not require evaluating the factorial of a number (calculators struggle with factorials of numbers as small as 100). As we will see, many physical situations are in a regime (e.g Poisson with a large mean) where they can be well approximated by the gaussian distribution. Grades in a class, heights of people, lengths of 8 month old cats, weights of brand new pencils, the rapid fluctuations in stock prices, time spent by patients in the ER, etc. are all normally distributed.

Example 7-1: Compare the probability of obtaining 140 heads in 300 coin tosses (of a fair coin) evaluated using the binomial and normal distributions.

We need to evaluate the binomial probability given by $P^{\text{binom}}(k = 140, N = 300, p = 0.5)$:

$$P^{\text{binom}}(k = 140, N = 300, p = 0.5) = \frac{N!}{k!(N - k)!} p^k (1 - p)^{N - k} = 0.02367$$

If you try to evaluate this with your calculator, it might crash when trying to evaluate $300!$, so this is best evaluated using a computer (see below).

The parameters for the normal distribution in terms of the parameters from the binomial distribution are given by $\mu_x = Np = 140$ and $\sigma_x = \sqrt{Np(1 - p)} = 8.7$. We can evaluate the probability for the normal distribution, $P^{\text{norm}}(x, \mu_x, \sigma_x)dx$ using $x = 140$ and $dx = 1$, which we interpret as the probability of obtaining x between 140 and 141, since the probability of obtaining an exact value of x is zero. Of course, with $dx = 1$, we only really evaluate $P^{\text{norm}}(x, \mu_x, \sigma_x)$ at the given value of x , but we should always remember that we have implicitly chosen $dx = 1$, and our result is the probability of having x between x and $x + dx$.

The simple python program below evaluates the two probabilities:

Python Code 7.1: Comparison of probabilities from binomial and normal distributions

```
import scipy.stats as stats
from math import sqrt
#We need the following binomial probability:
N=300
k=140
p=0.5
print("The binomial probability of k=140, N=300, p=0.5 is: {:.5f}").
    format(stats.binom.pmf(k, N, p))

#Converting this to the parameters for the normal distribution
mu=N*p
sigma=sqrt(N*p*(1.0-p))
print("The gaussian probability of x=140 for mu={} and sigma={:.2f} is:
    {:.5f}").format(mu, sigma, stats.norm.pdf(140, mu, sigma))
```

7.1: Output

```
The binomial probability of k=140, N=300, p=0.5 is: 0.02367
The gaussian probability of x=140 for mu=150.0 and sigma=8.66 is:
0.02365
```

and the probability from the normal distribution is indeed very close to that from the binomial distribution.

7.1 Properties of the normal distribution

The normal distribution has several properties that make it straightforward to use and understand.

7.1.1 Statistical properties

The mean of measurements that follow the normal distribution with mean μ and standard deviation σ is given by

$$\bar{x} = \int_{-\infty}^{+\infty} x P^{norm}(x, \mu_x, \sigma_x) dx = \mu_x \quad (7.3)$$

and their variance is given by:

$$\sigma^2 = \int_{-\infty}^{+\infty} (x - \bar{x})^2 P^{norm}(x, \mu_x, \sigma_x) dx = \sigma_x^2 \quad (7.4)$$

That is, the mean and standard deviation of measurements that are normally distributed are precisely the mean and standard deviation of the normal distribution.

7.1.2 Area underneath the normal distribution pdf

As we stated earlier, the total area underneath the normal distribution pdf must equal 1, as it represents the probability of measuring x to be *something*. The area under the normal distribution between x_a and x_b represents the probability that x is between x_a and x_b :

$$P(x_a \leq x \leq x_b) = \int_{x_a}^{x_b} P^{norm}(x, \mu_x, \sigma_x) dx \quad (7.5)$$

Example 7-2: On average, 25 students skip any given Error Analysis class. Use the Poisson and normal distributions to evaluate the probability that 30 students will skip class on a given day, and compare the results. When using the normal distribution, compare the probabilities obtained by doing the integral from $x = 30$ to $x = 31$ and that obtained by evaluating the pdf at $x = 30$ with $dx = 1$.

This problem is well modeled by the Poisson distribution, $P^{Poisson}(k, n)$, with the expected number of students skipping class, $n = 25$. Evaluating the probability of having $k = 30$ students gives:

$$P^{Poisson}(k = 30, n = 25) = \frac{n^k e^{-n}}{k!} = 0.0454$$

The corresponding normal distribution is given using $\mu_x = n = 25$ and $\sigma_x = \sqrt{n} = 5$. Recall that the normal distribution does not give the probability of obtaining an exact value, but rather must be used to evaluate the probability of obtaining a value within a certain range. We can choose to evaluate $P^{norm}(x, \mu_x, \sigma_x) dx$ with $x = 30$ and $dx = 1$, so that we can interpret the result as giving the probability of obtaining x between 30 and 31, which is almost equivalent (in interpretation) to the result we get from the

Poisson distribution. For this case, with $dx = 1$, we get:

$$P^{norm}(x = 30, \mu_x = 25, \sigma_x = 5)dx = \frac{1}{\sigma_x \sqrt{2\pi}} e^{-\frac{(x-\mu_x)^2}{2\sigma_x^2}} = 0.0483$$

If we do the proper integral, we find:

$$\int_{x=30}^{x=31} P^{norm}(x, \mu_x = 25, \sigma_x = 5)dx = 0.0436$$

The probability from the normal distribution when evaluated correctly (with the integral) is within 1 % of the Poisson probability, and within 3% when evaluated with $dx = 1$. As the mean of the distribution becomes bigger, the approximation between the normal and Poisson distribution will improve.

The calculations can be performed using the following code in python:

Python Code 7.2: Comparison of probabilities from Poisson and normal distributions

```
#Example 7-2
import scipy.stats as stats
from math import sqrt
#We need the following Poisson probability:
n=25
k=30
print("Poisson prob of 30 students skipping class: {:.4f}".format(stats.
    poisson.pmf(k,n)))
#This corresponds to the following gaussian:
mu=n
sigma=sqrt(n)
#Approximating the gaussian probability without taking the actual
integral:
print("Gaussian prob of 30 students skipping class evaluated at x=30:
    {:.4f}".format(stats.norm.pdf(30,mu,sigma)))
#Using the full integral, we need to subtract the cdf for x=30 and the
sf for x=31 from 1:
prob=1.0-stats.norm.cdf(30,mu,sigma)-stats.norm.sf(31,mu,sigma)
print("Gaussian prob of 30 students skipping class with integral: {:.4f}
    ".format(prob))
```

The output is:

7.2: Output

```
Poisson prob of 30 students skipping class: 0.0454
Gaussian prob of 30 students skipping class evaluated at x=30: 0.0484
Gaussian prob of 30 students skipping class with integral: 0.0436
```

As we will see, it is often useful to compute the probability of obtaining a value within a certain number of standard deviations from the mean of the distribution. For example, the

probability of obtaining x between $\mu_x - \sigma_x$ and $\mu_x + \sigma_x$ is given by:

$$P(\mu_x - \sigma_x \leq x \leq \mu_x + \sigma_x) = \int_{\mu_x - \sigma_x}^{\mu_x + \sigma_x} P^{norm}(x, \mu_x, \sigma_x) dx = 0.68 \quad (7.6)$$

which is a fixed number, regardless of the actual values of μ_x and σ_x . This is the origin of our seemingly arbitrary use of 68% as our usual confidence interval. Since many measurements are normally distributed (as we will see in the next section), the use of the standard deviation as the error (or quoted uncertainty) corresponds to a 68% chance of a value being within one standard deviation of the mean. Table 7.1 shows the probability for a measurement to lie within a certain number of standard deviations from the mean. This is illustrated in Figure 7.4.

n, number of σ_x	Probability of being in range $\mu_x \pm n\sigma_x$
1	0.682689
2	0.954500
3	0.997300
4	0.999937
5	0.999999

Table 7.1: Probability that a normally distributed measurement will fall within a certain number, n , of standard deviations from the mean.

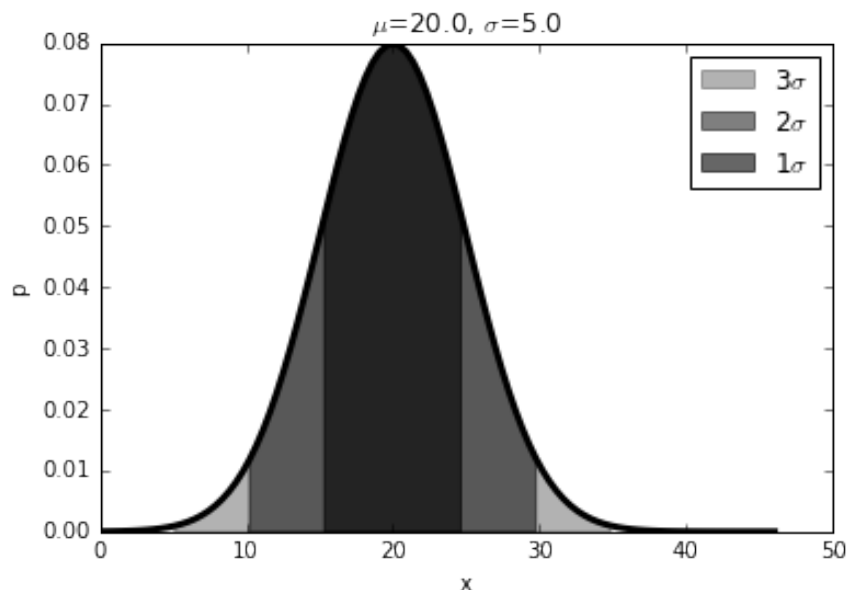


Figure 7.4: Illustration of the area under the normal distribution within 1, 2, and 3 standard deviations from the mean.

As you can see, the probability of being further than $3\sigma_x$ is already quite small, and quickly

becomes vanishingly small (as can also be seen in Figure 7.1). In particle physics, it is often required that a measurement have a validity of at least $5\sigma_x$ to be considered a discovery. For example, this would mean that in order to claim that a new particle has been discovered, the probability of the data given the null hypothesis must be at least $5\sigma_x$ from the mean expected if there were no new particle. A new particle may be discovered as a resonant peak in a spectrum; we would thus require that the peak indicating the presence of a particle be at least 5 times larger than the random fluctuations in the spectrum (under the assumption that those fluctuations are normally distributed).

Figure 7.5 shows, on the left, the normal distribution with $\mu_x=20$ and $\sigma_x=5$. Since this is not the probability of getting a given value of x (you need to take the integral to get a probability), we call this the “probability density function” (or pdf). On the right are the “cumulative probability density function” (cdf) and the “survival fraction” (sf), which is simply 1 minus the cdf. The cdf is the integral of the pdf. Thus, the cdf is a probability. The cdf evaluated at a certain value of x is the integral of the pdf from negative infinity to x , which corresponds to the **probability of obtaining a measurement x or smaller**. The survival fraction is thus the probability of getting x or bigger. There is no closed form for the cdf or sf corresponding to the normal distribution. Since the normal distribution is symmetric about the mean μ , the cdf evaluated at $x = \mu_x$ is exactly 0.5 (there is a 50% chance of measuring x smaller than the mean).

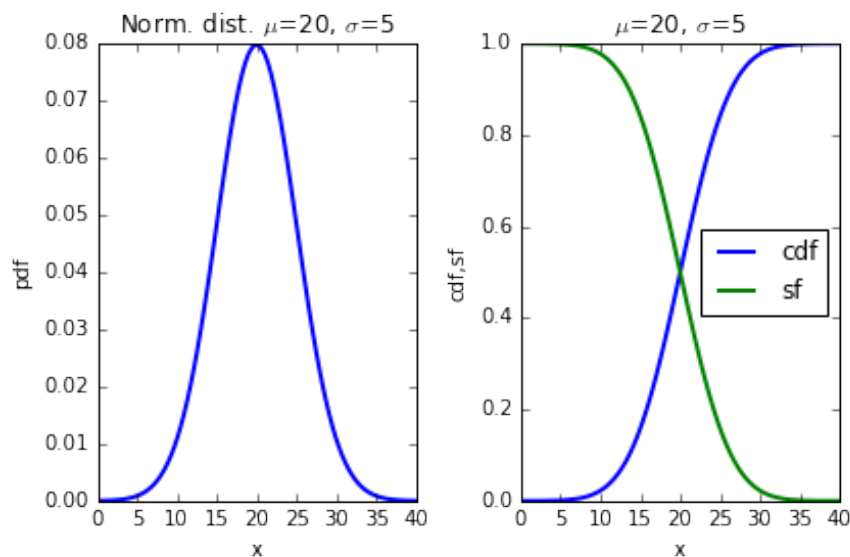


Figure 7.5: Normal probability density function (pdf), cumulative probability density function (cdf) and survival fraction (sf).

Example 7-3: An IQ test is designed such that the test results, on a random sample of the population, are normally distributed with a mean of 100 and a standard deviation of 10. What is the probability that a random person will score between 120 and 130? What is the probability that a random person will score 75 or below?

As stated in the problem, the distribution of test results follows a normal distribution with $\mu_x=100$ and $\sigma_x=10$. The probability of scoring between 120 and 130 is given by:

$$\begin{aligned} P(120 \leq x \leq 130) &= \int_{120}^{130} P^{norm}(x, \mu_x = 100, \sigma_x = 10) dx \\ &= 1 - P(x < 120) - P(x > 130) \\ &= 1 - CDF(120) - SF(130) \end{aligned}$$

where we have changed the probability to be calculated as 1 minus the probability of being outside of the range (which is easier to calculate using common functions for the cdf and sf). The probability of scoring 75 or lower is given by:

$$\begin{aligned} P(x \leq 75) &= \int_{-\infty}^{75} P^{norm}(x, \mu_x = 100, \sigma_x = 10) dx \\ &= CDF(75) \end{aligned}$$

which is the same as the cdf evaluated at $x=75$.

These probabilities are easiest to estimate with a computer program, as in the following python code:

Python Code 7.3: SF and CDF for normal distribution

```
import scipy.stats as stats

#In order to get the probability of being within a certain range,
#it easier to calculate the probability of being outside that range!

pBigger130=stats.norm.sf(130,100,10)
pSmaller120=stats.norm.cdf(120,100,10)
print("Prob of scoring between 120 and 130: {:.2f} %".format(100*(1-
    pBigger130-pSmaller120)))
print("Prob of scoring 75 or below: {:.2f} %".format(100*(stats.norm.cdf
    (75,100,10))))
```

7.3: Output

```
Prob of scoring between 120 and 130: 2.14 %
Prob of scoring 75 or below: 0.62 %
```

7.2 Why most measurements are normally distributed

In this section, we motivate why we expect most measurements to be normally distributed. Suppose that we are measuring some quantity which has a “true” value equal to X , and that our measurement is subject to a small random error ϵ . That is, each time we perform

a measurement we have equal probability ($p = 0.5$) of obtaining $X - \epsilon$ or $X + \epsilon$. With this single source of random error, over many measurements, we will obtain $X - \epsilon$, 50% of the time and $X + \epsilon$, the other 50% of the time.

If we have 2 sources of random error (each equally probable to add or subtract ϵ from the true value X), then we can obtain three possible measurements: $\{X - 2\epsilon, X, X + 2\epsilon\}$, depending on whether the errors occurred in the same “direction” or not. The possible ways of obtaining the 3 possible results are listed in Table 7.2, where we can see that there are 4 possible outcomes, 2 of which are to measure X and two of which are to measure a value that is 2ϵ away. Since the errors are random, each outcome is equally probable and we expect to measure X 50% of the time (2 out of 4 possible outcomes), $X - 2\epsilon$, 25% of the time, and $X + 2\epsilon$, 25% of the time.

Result	Combination of the 2 errors
$X - 2\epsilon$	$-\epsilon - \epsilon$
X	$-\epsilon + \epsilon$ or $+\epsilon - \epsilon$
$X + 2\epsilon$	$+\epsilon + \epsilon$

Table 7.2: Ways to obtain different results when 2 random errors contribute.

Another way to think about this is in terms of the binomial distribution. In order to evaluate the probability of obtaining $X + 2\epsilon$, we need to know the probability of both errors being positive. Each error has a probability $p = 0.5$ of being positive, and we would like to know the probability of having $k = 2$ positive errors (“successes”) when we add $N = 2$ errors (“trials”). The binomial probability for this case is:

$$P^{binom}(k = 2, N = 2, p = 0.5) = 0.25$$

just as we obtained by counting the possible outcomes in Table 7.2.

If we have N source of random errors, each error being equal to ϵ and having equal probability to push our measured value up or down, the distribution of our measurements will be binomially distributed. If a specific measured value x required that k of the errors be positive and $N - k$ of the errors be negative, then the probability of obtaining that particular measurement of x is given by:

$$P(x) = P^{binom}(k, N, p = 0.5)$$

which corresponds to the probability that the correct number of errors, k , are positive. The possible range in values of x is between $X - N\epsilon$ and $X + N\epsilon$. The distribution of our measurements will be symmetric about X and each measurement, x , will have a binomial probability of occurring (evaluated by determining how many errors, k , are needed to be positive). If we have k positive errors, then our measured value, x , is given by:

$$\begin{aligned} x &= X + k\epsilon - (N - k)\epsilon \\ &= X + 2k\epsilon - N\epsilon \end{aligned}$$

If we repeat our measurement many times, x will have a distribution of values. The average value of x will be:

$$\begin{aligned}\bar{x} &= X + 2\bar{k}\epsilon - N\epsilon \\ &= X + 2Np\epsilon - N\epsilon \\ &= X\end{aligned}$$

where we have used the fact that $p = 0.5$, and that X , ϵ , and N are constants.

The standard deviation (variance) of the distribution of values of x will depend on the standard deviation (variance) of the values that we obtain for k (X , ϵ , and N are constants, so they will not contribute to giving a distribution of values to x). A standard deviation (i.e. a spread) in the values of k of σ_k will result in a standard deviation (a spread) for values for x given by:

$$\sigma_x = 2\sigma_k\epsilon \quad (7.7)$$

k is distributed according to the binomial distribution, which, as you recall, has a standard deviation given by:

$$\sigma_k = \sqrt{Np(1-p)} = \sqrt{N\frac{1}{2}(1-\frac{1}{2})} = \frac{1}{2}\sqrt{N} \quad (7.8)$$

where we have used the fact that $p = 0.5$. The standard deviation of x is thus given by:

$$\sigma_x = 2\sigma_k\epsilon = \epsilon\sqrt{N} \quad (7.9)$$

We now argue that as ϵ becomes infinitely small and N becomes infinitely large, but in a way that $\epsilon\sqrt{N}$ is finite and equal to σ_x , the distribution of the values of x approaches a normal distribution with mean X and standard deviation σ_x . That is, in the limit of an infinite number of infinitely small random errors, we obtain a normal distribution for the values of x with $\mu = X$ and $\sigma = \sigma_x$.

We already saw that the binomial distribution approaches the normal distribution when the mean of the binomial distribution (given by Np) is large. This is true in our case since N approaches infinity while $p = 0.5$ is finite. Remember, N , is the number of small random errors, and the mean of the binomial distribution, Np , is the mean number, k , of positive errors that result in a certain measured value x . This is not the same as requiring that X (the mean of the distribution of the x) be large. Requiring that the mean of the number of positive errors be large has no impact on the mean value of the measured quantities x . We also require that the individual errors, ϵ , become very small. This means that the possible values of x that we can measure (which are spaced by ϵ) become closer and closer together, and eventually approach a continuous distribution.

Figure 7.6 shows how the distribution of the measured values, x , approaches the normal distribution when the number of random errors, N , is increased at the same time as the

random error, ϵ , is decreased in a way to keep $\epsilon\sqrt{N} = 0.1$ constant. The histograms show the results of 100,000 (simulated) measurements of x with a given N and ϵ . The histograms are normalized, and are seen to approach the normal distribution (black line) as N becomes large. The mean value of the measurements of x was chosen to be $X = 1.0$ (which is small) to illustrate that while N is large, this does not require the mean of the distribution of the x values to be large. The histograms were created by generating random values of k from a binomial distribution with N and $p = 0.5$ and then converting those to the corresponding values of x , given by $x = X + 2k\epsilon - N\epsilon$.

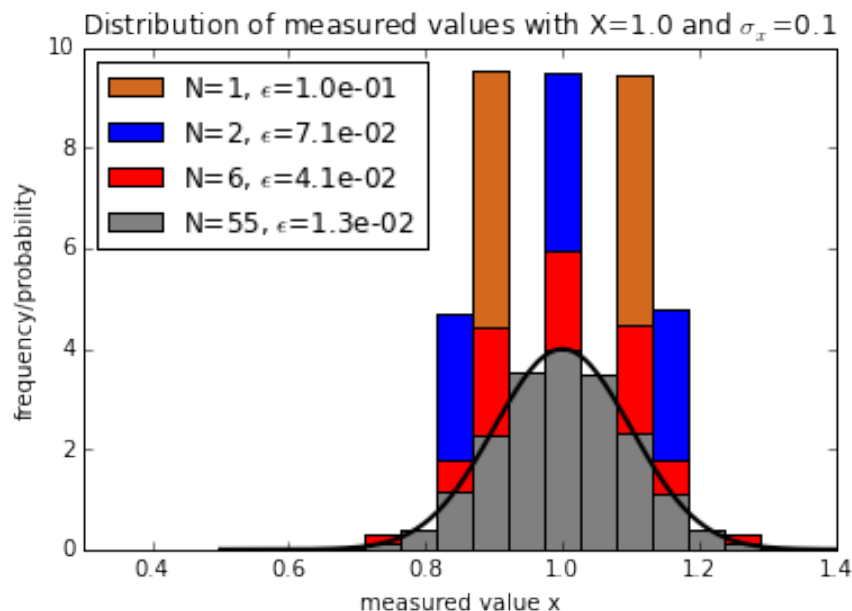


Figure 7.6: Normalized histograms for the results of 100,000 measurements of a value of x that is centred about $X = 1.0$ with a standard deviation of 0.1. The different histograms show how the distribution changes as the number of random errors, N , is increased and the random error of each measurement, ϵ , is decreased in such a way as to keep $\sigma_x = \epsilon\sqrt{N}$ constant. As N increased, the normal distribution, shown by the black line, is approached.

7.2.1 The Central Limit Theorem

Although a full development of the Central Limit Theorem (CLT) is beyond the scope of this book, it is certainly worth mentioning in this context. The CLT states that the distribution of a sum of a large number of independent identically distributed variables is normally distributed, *regardless* of the distribution that generated the independent variables. This is in fact a remarkable result, and applies to a wide range of quantities, as many things can be thought of as sums.

For example, the final marks from all of the students in a course are usually normally distributed (if there are many students and each student has multiple marks contributing to their final mark). The final mark from a single student is given by the sum of all of the marks that they obtained in the course (usually scaled by some number to get an average, but ultimately, it is still a sum). Each individual mark that the student obtained may not

be from a normal distribution (e.g. for a given assignment, the professor could have graded the marks such that they are not normally distributed). It is thus quite remarkable that even if the individual assignments marks are not normally distributed, the final marks in the course are! This is the reason that some professors will adjust their final marks with a “bell curve” (a normal distribution!). In effect, the strategy is usually to shift all of the marks so that only a fraction of the students fail (e.g. only those 1 standard deviation below the mean).

Figure 7.7 shows a simulated histogram to illustrate the CTL. In this case, a class of 100 students was simulated. Each student has 20 marks (e.g. from 20 assignments), and for each of those marks, the student has an equal probability of obtaining any mark between 0 and 10 (a flat distribution of marks, certainly not a normal distribution). As you can see, when we compute the average mark for each student (their “final mark”), and plot the distribution of those marks for all students, we get a distribution that looks like a normal distribution, remarkable!

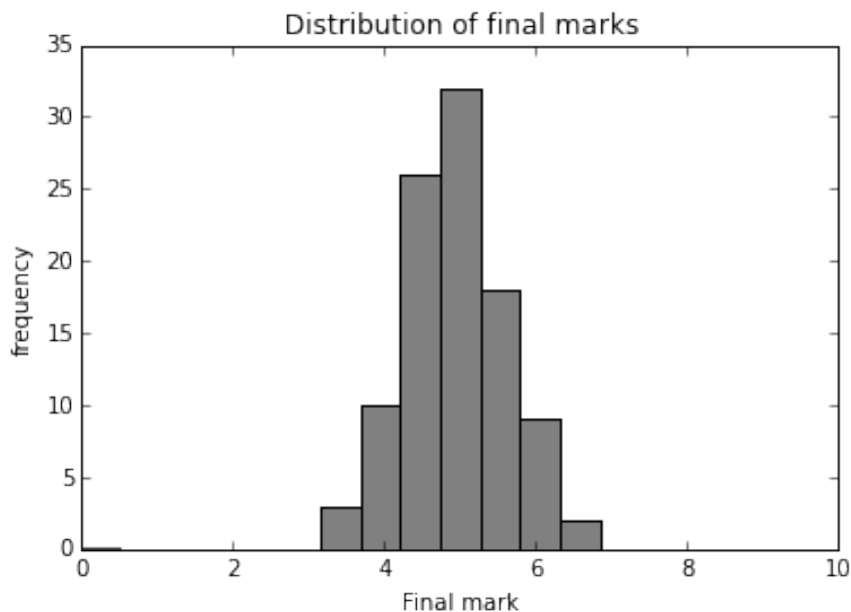


Figure 7.7: Simulated histogram of final marks from 100 students in a course.

7.3 Combining normally distributed quantities

In this section, we examine the distribution of a quantity that depends on one or more normally distributed quantities.

7.3.1 Addition of a number to a normally distributed quantity

Given a quantity, x , that is normally distributed with a probability density function, $P_x^{norm}(x)$, we wish to know the distribution, $P_F(F)$, of a quantity, F , that is given by:

$$F = a + x$$

where a is a constant. Since x is normally distributed, we have:

$$P_x(x) = \frac{1}{\sigma_x \sqrt{2\pi}} e^{-\frac{(x-\mu_x)^2}{2\sigma_x^2}}$$

and we want to know the probability density function, $P_F(F)$. Since $F = a + x$, the probability, $P_F(F)dF$, of obtaining a certain value of F between F and $F + dF$ is the same as that for obtaining the corresponding value of x , between $x = F - a$ and $x = F - a + dF = F - a + \frac{dF}{dx}dx = F - a + dx$ (since $\frac{dF}{dx} = 1$). The probability of obtaining F between F and $F + dF$ is thus equal to the probability of obtaining x between $F - a$ and $F - a + dx$:

$$P_F(F)dF = P_x(x = F - a)dx = \frac{1}{\sigma_x \sqrt{2\pi}} e^{-\frac{((F-a)-\mu_x)^2}{2\sigma_x^2}} dx \quad (7.10)$$

$$= \frac{1}{\sigma_x \sqrt{2\pi}} e^{-\frac{(F-(a+\mu_x))^2}{2\sigma_x^2}} dx \quad (7.11)$$

$$= \frac{1}{\sigma_F \sqrt{2\pi}} e^{-\frac{(F-\mu_F)^2}{2\sigma_F^2}} dF \quad (7.12)$$

$$(7.13)$$

Thus, F is also normally distributed, with the same standard deviation as x , $\sigma_F = \sigma_x$, but with the mean shifted to $\mu_F = \mu_x + a$.

7.3.2 Multiplication of a number with a normally distributed quantity

Given a quantity, x , that is normally distributed with a probability, $P_x^{norm}(x)$, we wish to know the distribution, $P_F(F)$ of a quantity, F , that is given by:

$$F = ax$$

where a is a constant. Since x is normally distributed, we have:

$$P_x(x) = \frac{1}{\sigma_x \sqrt{2\pi}} e^{-\frac{(x-\mu_x)^2}{2\sigma_x^2}}$$

and we want to know the probability density function, $P_F(F)$. Since $F = ax$, the probability, $P_F(F)dF$, of obtaining a certain value of F between F and $F + dF$ is the same as that of obtaining the corresponding value of x , between $x = \frac{F}{a}$ and $x = \frac{F+dF}{a} = \frac{F}{a} + \frac{1}{a} \frac{dF}{dx} dx = \frac{F}{a} + dx$ (since $\frac{dF}{dx} = a$). The probability of obtaining F between F and $F + dF$ is thus equal to the

probability of obtaining x between $\frac{F}{a}$ and $\frac{F}{a} + dx$:

$$\begin{aligned} P_F(F)dF &= P_x\left(x = \frac{F}{a}\right)dx = \frac{1}{\sigma_x\sqrt{2\pi}}e^{-\frac{(\frac{F}{a}-\mu_x)^2}{2\sigma_x^2}}dx \\ &= \frac{1}{\sigma_x\sqrt{2\pi}}e^{-\frac{(F-a\mu_x)^2}{2\sigma_x^2a^2}}dF \\ &= \frac{1}{\sigma_x\sqrt{2\pi}}e^{-\frac{(F-\mu_F)^2}{2\sigma_F^2}}\frac{1}{a}dF \end{aligned} \quad (7.14)$$

$$= \frac{1}{\sigma_F\sqrt{2\pi}}e^{-\frac{(F-\mu_F)^2}{2\sigma_F^2}}dF \quad (7.15)$$

$$(7.16)$$

where we have introduced $\mu_F = a\mu_x$ and $\sigma_F = a\sigma_x$. $P_F(F)$ is the pdf for a normal distribution with mean $\mu_F = a\mu_x$ and standard deviation $\sigma_F = a\sigma_x$. Thus, multiplying a normally distributed quantity by a constant results in a quantity that is still normally distributed, but with a mean and standard deviation that are scaled by that constant.

7.3.3 Sum of normally distributed quantities

Let x and y be two independent quantities that are normally distributed about μ_x and μ_y , with standard deviations σ_x and σ_y , respectively. We would like to know the expected distribution, $P_z(z)$, of their sum, $z = x + y$. We will show that z is normally distributed with a mean $\mu_z = \mu_x + \mu_y$ and a standard deviation $\sigma_z^2 = \sigma_x^2 + \sigma_y^2$. If we interpret the standard deviations as the uncertainties on the measured quantities, then we will effectively have shown that adding in quadrature is the correct method to obtain the uncertainty on the sum of two quantities.

Let x and y be distributed according to probability density functions, $P_x(x)$ and $P_y(y)$, respectively (which we will later set equal to the normal distribution). We also assume that x and y are independent (that is, the probability of a specific value of y does not depend on the value of x , and vice versa). We thus wish to find the probability density function for z , which we will call $P_z(z)$.

We start by assuming that x , y , and z can only take discrete integer values and we will later change to continuous variables. If x is equal to some value k , and $z = x + y$, we must have that $y = z - k$. For a given value $x = k$, then the probability of having a certain value of z is the product of $P_x(x = k)$ and $P_y(y = z - k)$ (i.e. the joint probability of having a specific value of x and a specific value of y):

$$P_z(z|x = k) = P_x(k)P_y(z - k)$$

where we have indicated ($P_z(z|x = k)$) that this is the probability of obtaining a certain value of z *given* that $x = k$. Of course, we do not care what value x has, since k is a completely arbitrary number. We really want to sum this probability over all of the possible values of

k to obtain $P_z(z)$ independently of the specific value of x :

$$\begin{aligned} P_z(z) &= \sum_k P_z(z|x=k) \\ P_z(z) &= \sum_k P_x(k)P_y(z-k) \end{aligned} \quad (7.17)$$

which can be applied to any discrete distributions, P_x and P_y , to obtain the distribution of their sum.

Example 7-4: When rolling two dice, what is the probability that their sum is 7?

Using our above notation, we are interested in the probability of $P_z(7)$ given the probability distribution for digits from each of the dice. If the number on the individual dice are given by x and y with probabilities $P_x(x)$ and $P_y(y)$, respectively, equation 7.17 gives $P_z(7)$ as:

$$\begin{aligned} P_z(7) &= \sum_{k=1}^{k=6} P_x(k)P_y(7-k) \\ &= P_x(1)P_y(6) + P_x(2)P_y(5) + P_x(3)P_y(4) + P_x(4)P_y(3) + P_x(5)P_y(2) + P_x(6)P_y(1) \end{aligned}$$

If both dice are fair, then $P_x = P_y = \frac{1}{6}$. The probability of $P_z(7)$ is thus $6 \times \frac{1}{6} \frac{1}{6} = \frac{1}{6}$, which can also be found by tabulating all of the possible outcomes of throwing the dice and counting how many of the outcomes sum to 7.

If we are now interested in the continuous case, the sum in equation 7.17 must become an integral:

$$P_z(z) = \int_{k=-\infty}^{k=+\infty} P_x(k)P_y(z-k)dk \quad (7.18)$$

which is valid to obtain the probability of a sum of two continuous independent variables given by probability density functions P_x and P_y . Equation 7.18 is called a “convolution” of the functions P_x and P_y , and appears in many areas of physics and mathematics not related to probabilities. We thus have our formula to compute the functional form of the distribution of the sum of x and y .

For normal distributions, P_x and P_y are given by:

$$\begin{aligned} P_x(x) &= \frac{1}{\sigma_x \sqrt{2\pi}} e^{-\frac{(x-\mu_x)^2}{2\sigma_x^2}} \\ P_y(y) &= \frac{1}{\sigma_y \sqrt{2\pi}} e^{-\frac{(y-\mu_y)^2}{2\sigma_y^2}} \end{aligned}$$

Inserting this into equation 7.18:

$$\begin{aligned} P_z(z) &= \int_{k=-\infty}^{k=+\infty} P_x(k)P_y(z-k)dk \\ &= \int_{k=-\infty}^{k=+\infty} \frac{1}{\sigma_x\sqrt{2\pi}}e^{-\frac{(k-\mu_x)^2}{2\sigma_x^2}} \frac{1}{\sigma_y\sqrt{2\pi}}e^{-\frac{((z-k)-\mu_y)^2}{2\sigma_y^2}} dk \end{aligned}$$

This can be re-arranged after a substantial amount of algebraic manipulation to:

$$P_z(z) = \int_{k=-\infty}^{k=+\infty} \frac{1}{\sqrt{\sigma_x^2 + \sigma_y^2}\sqrt{2\pi}}e^{-\frac{(z-(\mu_x+\mu_y))^2}{2(\sigma_x^2+\sigma_y^2)}} \frac{1}{\frac{\sigma_x\sigma_y}{\sqrt{\sigma_x^2+\sigma_y^2}}\sqrt{2\pi}}e^{-\frac{\left(k-\frac{\sigma_x^2(z-\mu_y)+\sigma_y^2\mu_x}{\sigma_x^2+\sigma_y^2}\right)^2}{2\left(\frac{\sigma_x\sigma_y}{\sqrt{\sigma_x^2+\sigma_y^2}}\right)^2}} dk$$

The first two multiplicative terms do not depend on k , so they can be taken out of the integral:

$$\begin{aligned} P_z(z) &= \frac{1}{\sqrt{\sigma_x^2 + \sigma_y^2}\sqrt{2\pi}}e^{-\frac{(z-(\mu_x+\mu_y))^2}{2(\sigma_x^2+\sigma_y^2)}} \int_{k=-\infty}^{k=+\infty} \frac{1}{\frac{\sigma_x\sigma_y}{\sqrt{\sigma_x^2+\sigma_y^2}}\sqrt{2\pi}}e^{-\frac{\left(k-\frac{\sigma_x^2(z-\mu_y)+\sigma_y^2\mu_x}{\sigma_x^2+\sigma_y^2}\right)^2}{2\left(\frac{\sigma_x\sigma_y}{\sqrt{\sigma_x^2+\sigma_y^2}}\right)^2}} dk \\ &= \frac{1}{\sqrt{\sigma_x^2 + \sigma_y^2}\sqrt{2\pi}}e^{-\frac{(z-(\mu_x+\mu_y))^2}{2(\sigma_x^2+\sigma_y^2)}} \end{aligned}$$

where we have recognized that the remaining integral corresponded to the integral of a normal distribution, which is equal to 1. We are thus left with $P_z(z)$ given by a normal distribution with mean $\mu_z = \mu_x + \mu_y$ and standard deviation $\sigma_z = \sqrt{\sigma_x^2 + \sigma_y^2}$, as anticipated. Thus, we have shown that the sum of two normally distributed quantities is normally distributed, with a standard deviation given by the quadrature sum of the standard deviation of the variables in the sum. This is trivially extended to a sum of more than 2 terms.

7.3.4 Function of normally distributed quantities

Although we just proved¹ that the sum of two normally distributed quantities is normally distributed, we can generalize the result for any function of two variables. If a quantity $F(x, y)$ depends on independent and normally distributed quantities x and y , and the standard deviations on x and y are “small”, then we can use a Taylor series to approximate the value of F for x and y near their mean values, μ_x and μ_y :

$$F(x, y) \approx F(\mu_x, \mu_y) + \frac{\partial F}{\partial x}(x - \mu_x) + \frac{\partial F}{\partial y}(y - \mu_y) + \dots$$

¹Minus the lengthy algebra!

Since we assumed that the errors in x and y are small, then it is reasonable to assume that x and y will generally be close to their means, μ_x and μ_y and that the Taylor series approximation is valid (i.e. that $x - \mu_x$ and $y - \mu_y$ are small). F can thus be approximated by the sum of three terms; since we know how a sum is distributed, we only need to know how the three terms that make up the sum are distributed.

The first term is constant and will only shift the mean of the distribution of the sum, as in equation 7.10, (but does not add to the standard deviation). The second term, $\frac{\partial F}{\partial x}(x - \mu_x)$, is the product of a constant, $\frac{\partial F}{\partial x}$ (evaluated at $x = \mu_x$ and $y = \mu_y$), and a quantity, $x - \mu_x$, that is normally distributed about 0 with a standard deviation of σ_x . The second term is thus normally distributed with a mean of zero and a standard deviation given by $\frac{\partial F}{\partial x}\sigma_x$, as in equation 7.14. The third term is similar, with a mean of zero and a standard deviation of $\frac{\partial F}{\partial y}\sigma_y$. F is thus the sum of a constant term, and two normal distributions with means of 0 and standard deviations of $\frac{\partial F}{\partial x}\sigma_x$ and $\frac{\partial F}{\partial y}\sigma_y$, respectively. F is thus normally distributed with a mean and standard deviations given by:

$$\begin{aligned}\mu_F &= F(\mu_x, \mu_y) \\ \sigma_F^2 &= \left(\frac{\partial F}{\partial x}\sigma_x\right)^2 + \left(\frac{\partial F}{\partial y}\sigma_y\right)^2\end{aligned}\tag{7.19}$$

just as we had prescribed in Chapter 4 and found in Chapter 5. This result is easily extended if F depends on more than two variables.

7.4 Determining parameters of the normal distribution from data

Given a set of N measurements, $\{x_1, x_2, \dots, x_N\}$, of a normally distributed quantity, we wish to estimate the “true” value of the mean and standard deviation, μ_x and σ_x , of the normal distribution that describes the data. As we will see, the sample mean and the sample standard deviation are good estimates of the mean and standard deviation of the underlying normal distribution.

7.4.1 The principle of maximum likelihood

Given a probability density function, $P(x, \vec{\beta})$, we wish to determine the set of parameters (represented as a vector, $\vec{\beta}$) that best describes a set of data that is modelled by that pdf. For example, the pdf may be that for the normal distribution, $P^{norm}(x, \mu_x, \sigma_x)$, for which we wish to estimate μ_x and σ_x , given a set of N measurements, $\{x_1, x_2, \dots, x_N\}$.

The “principle of maximum likelihood” is a procedure to determine the value of the parameters $\vec{\beta}$ for which the “likelihood” of the data is maximized. The likelihood, $L(x|\vec{\beta})$, is essentially the probability of obtaining a particular data set given a particular set of parameters. For example, the likelihood of obtaining the N data points $\{x_1, x_2, \dots, x_N\}$ would

be given by the joint probability of obtaining each individual data point:

$$\begin{aligned} L(x|\vec{\beta}) &= P(x_1, \vec{\beta})P(x_2, \vec{\beta}) \dots P(x_N, \vec{\beta}) \\ &= \prod_{i=1}^{i=N} P(x_i, \vec{\beta}) \end{aligned} \quad (7.20)$$

Given a specific value of the parameters, $\vec{\beta}$, we can calculate the likelihood of our particular set of measurements. We can then vary those parameters until the likelihood is maximized, yielding our best estimate of the parameters. Formally, we want to find the set of parameters, $\{\beta_1, \beta_2, \dots\}$, such that:

$$\frac{\partial L}{\partial \beta_i} = 0 \quad (7.21)$$

since the maximum of L is found when the derivative is zero. Often, it is easier to work with a sum of probabilities rather than a product. We can define the “log-likelihood”, to be the natural logarithm of the likelihood, and use the property that $\ln(ab) = \ln(a) + \ln(b)$. Because the logarithm is a monotonic function, the value of the parameters that maximize the log-likelihood also maximize the likelihood. Generally, the log-likelihood is:

$$\begin{aligned} \ln(L(x|\vec{\beta})) &= \ln\left(\prod_{i=1}^{i=N} P(x_i, \vec{\beta})\right) \\ &= \sum_{i=1}^{i=N} \ln(P(x_i, \vec{\beta})) \end{aligned}$$

and the maximum is given by the condition:

$$\frac{\partial \ln(L(x|\vec{\beta}))}{\partial \beta_i} = 0 \quad (7.22)$$

7.4.2 Estimating the mean and standard deviation of a normal distribution from data

In the case of the normal distribution, the likelihood of obtaining a particular set of N measurements, $\{x_1, x_2, \dots, x_N\}$, given a particular choice of mean, μ_x , and standard deviation, σ_x , is:

$$\begin{aligned} L(x|\mu_x, \sigma_x) &= \prod_{i=1}^{i=N} P^{norm}(x_i, \mu_x, \sigma_x) \\ &= \frac{1}{\sigma_x \sqrt{2\pi}} e^{-\frac{(x_1 - \mu_x)^2}{2\sigma_x^2}} \frac{1}{\sigma_x \sqrt{2\pi}} e^{-\frac{(x_2 - \mu_x)^2}{2\sigma_x^2}} \dots \frac{1}{\sigma_x \sqrt{2\pi}} e^{-\frac{(x_N - \mu_x)^2}{2\sigma_x^2}} \\ &= \left(\frac{1}{\sigma_x \sqrt{2\pi}}\right)^N e^{-\frac{1}{2} \left(\frac{(x_1 - \mu_x)^2}{\sigma_x^2} + \frac{(x_2 - \mu_x)^2}{\sigma_x^2} + \dots + \frac{(x_N - \mu_x)^2}{\sigma_x^2}\right)} \end{aligned} \quad (7.23)$$

Because we are only interested in the maximum value of the likelihood function, we can work with the logarithm, $\ln(L)$, instead of the likelihood function itself (the logarithm is a

monotonically increasing function and will thus have a maximum at the same value of the parameters as the likelihood function). For convenience, we prefer to *minimize* the negative of the logarithm of the likelihood, instead of maximizing the logarithm of the likelihood. That is, we introduce the “negative log-likelihood”, $-\ln(L)$:

$$\begin{aligned}
-\ln(L) &= -\ln \left(\left[\frac{1}{\sigma_x \sqrt{2\pi}} \right]^N e^{-\frac{1}{2} \left(\frac{(x_1 - \mu_x)^2}{\sigma_x^2} + \frac{(x_2 - \mu_x)^2}{\sigma_x^2} + \dots + \frac{(x_N - \mu_x)^2}{\sigma_x^2} \right)} \right) \\
&= -\ln \left(\left[\frac{1}{\sigma_x \sqrt{2\pi}} \right]^N \right) + \frac{1}{2} \left(\frac{(x_1 - \mu_x)^2}{\sigma_x^2} + \frac{(x_2 - \mu_x)^2}{\sigma_x^2} + \dots + \frac{(x_N - \mu_x)^2}{\sigma_x^2} \right) \\
&= N \ln(\sqrt{2\pi}) + N \ln(\sigma_x) + \frac{1}{2} \sum_{i=1}^N \frac{(x_i - \mu_x)^2}{\sigma_x^2} \\
&= N \ln(\sigma_x) + \frac{1}{2\sigma_x^2} \sum_{i=1}^N (x_i - \mu_x)^2
\end{aligned} \tag{7.24}$$

where in the last line, we dropped the constant term, $N \ln(\sqrt{2\pi})$, since it will have no influence on minimizing the negative log-likelihood in terms of μ_x or σ_x . Finding the set of parameters that maximize the likelihood function is thus equivalent to finding the parameters that minimize the negative log-likelihood function.

Thus, the best estimate of μ_x from the data, $\hat{\mu}_x$, using the principle of maximum likelihood, is given by the condition:

$$\frac{\partial}{\partial \mu_x} (-\ln(L)) = 0 \tag{7.25}$$

and the best estimate of the standard deviation, σ_x , from the data $\hat{\sigma}_x$ is given by:

$$\frac{\partial}{\partial \sigma_x} (-\ln(L)) = 0 \tag{7.26}$$

We call $\hat{\mu}_x$ and $\hat{\sigma}_x$ the “maximum likelihood estimates” of the true parameters. We used hats on the quantities to denote that these are estimated from the data as we do not know the “true” values, μ_x and σ_x . Also note that there are methods other than maximum likelihood that can be used to estimate parameters. It certainly makes intuitive sense to think of maximizing the probability of obtaining a given data set, but it is difficult to justify scientifically (e.g. what is really meant by “probability”?). For this reason, you should remember to always specify the method used to estimate your parameters from data.

Taking the derivative of the negative log-likelihood function with respect to μ_x , we have:

$$\begin{aligned}
 \frac{\partial}{\partial \mu_x} (-\ln(L)) &= \frac{\partial}{\partial \mu_x} \left(N \ln(\sigma_x) + \frac{1}{2\sigma_x^2} \sum_{i=1}^{i=N} (x_i - \mu_x)^2 \right) \\
 &= \frac{1}{2\sigma_x^2} \sum_{i=1}^{i=N} \frac{\partial}{\partial \mu_x} (x_i - \mu_x)^2 \\
 &= \frac{1}{2\sigma_x^2} \sum_{i=1}^{i=N} -2(x_i - \mu_x) \\
 &= \frac{-1}{\sigma_x^2} \left(\sum_{i=1}^{i=N} x_i - \mu_x \right) \\
 &= \frac{1}{\sigma_x^2} \left(N\mu_x - \sum_{i=1}^{i=N} x_i \right)
 \end{aligned} \tag{7.27}$$

Setting the above expression to zero, we find that:

$$\begin{aligned}
 \frac{1}{\sigma_x^2} \left(N\mu_x - \sum_{i=1}^{i=N} x_i \right) &= 0 \\
 \mu_x &= \frac{1}{N} \sum_{i=1}^{i=N} x_i \\
 \therefore \hat{\mu}_x &= \bar{x}
 \end{aligned} \tag{7.28}$$

Thus, the maximum likelihood estimate of the mean of the normal distribution, $\hat{\mu}_x$, is precisely the sample mean, \bar{x} , of the data!

We now derive the formula for the maximum likelihood estimate of the standard deviation:

$$\begin{aligned}
 \frac{\partial}{\partial \sigma_x} (-\ln(L)) &= \frac{\partial}{\partial \sigma_x} \left(N \ln(\sigma_x) + \frac{1}{2\sigma_x^2} \sum_{i=1}^{i=N} (x_i - \mu_x)^2 \right) \\
 &= \frac{N}{\sigma_x} + \frac{\partial}{\partial \sigma_x} \frac{1}{2\sigma_x^2} \sum_{i=1}^{i=N} (x_i - \mu_x)^2 \\
 &= \frac{N}{\sigma_x} - \frac{1}{\sigma_x^3} \sum_{i=1}^{i=N} (x_i - \mu_x)^2
 \end{aligned} \tag{7.29}$$

Setting the above expression to zero, we have:

$$\begin{aligned}
 \frac{N}{\sigma_x} - \frac{1}{\sigma_x^3} \sum_{i=1}^{i=N} (x_i - \mu_x)^2 &= 0 \\
 \sigma_x^2 &= \frac{1}{N} \sum_{i=1}^{i=N} (x_i - \mu_x)^2
 \end{aligned} \tag{7.30}$$

which gives us the maximum likelihood estimator for the standard deviation (or rather, the variance). You may now notice that we cannot use our data to actually calculate the estimate of the variance, σ_x^2 , because it depends on the “true” mean of the distribution, μ_x , which we do not know! We can only use our estimate of the mean, $\hat{\mu}_x$, in its place. However, when we replace the true mean, μ_x , with its estimate, we cannot be guaranteed that our estimate for the variance, $\hat{\sigma}_x^2$, is unbiased. In fact, we already pointed this out in Chapter 5 where we argued that in order to get an unbiased measure of the variance, we had to replace the N in the denominator by $N - 1$. The correct estimate of the variance of the normal distribution from data is given by:

$$\therefore \hat{\sigma}_x^2 = \frac{1}{N-1} \sum_{i=1}^{i=N} (x_i - \hat{\mu}_x)^2 = \sigma^2 \quad (7.31)$$

where σ^2 is the sample variance, as defined in Chapter 5. The maximum likelihood estimate of the standard deviation (which does not have the $N - 1$) is biased, although the effect is small even for only moderately large N .

7.4.3 The error on the mean

As we just saw, the sample mean, \bar{x} , and sample standard deviations, σ , from a set of N measurements, $\{x_1, x_2, \dots, x_N\}$ can be used as estimates of the true mean, μ_x , and standard deviation, σ_x , of a normal distribution that describes those measurements. We now determine the uncertainty, $\sigma_{\bar{x}}$, on our estimated mean of the normal distribution. In Chapter 5, we introduced the error on the mean, $\sigma_{\bar{x}}$:

$$\sigma_{\bar{x}} \equiv \frac{\sigma}{\sqrt{N}} \quad (7.32)$$

and we want to show here, that this is the correct uncertainty on our estimate of the mean $\hat{\mu}_x = \bar{x}$ of the normal distribution estimated from our measurements.

In order to “measure” the error in our estimate of the mean, we would need to estimate the mean many times and see how that is distributed. We would thus need to perform our experiment many, say, M times. In the first experiment, we would measure N values: $\{x_1^1, x_2^1, \dots, x_N^1\}$, and obtain a mean value \bar{x}^1 . We would then repeat the experiment a second time, and obtain another N measurements: $\{x_1^2, x_2^2, \dots, x_N^2\}$ with a new mean value \bar{x}^2 . We would repeat this M times, until we have obtained M mean values: $\{\bar{x}^1, \bar{x}^2, \dots, \bar{x}^M\}$. In order to understand our error on the mean, we would want to know how the mean values are distributed.

Since the quantity x that we are measuring is normally distributed, we are guaranteed that the mean values, \bar{x}^i , are also normally distributed. This is true by the Central Limit Theorem (even if the x were not normally distributed). We also showed explicitly that the sum of normally distributed quantities is normally distributed (the sample mean, \bar{x} is just a sum multiplied by a constant $\frac{1}{N}$). We thus know that the sample means, $\{\bar{x}^1, \bar{x}^2, \dots, \bar{x}^M\}$, follow a normal distribution, and we are interested in the standard deviation of that distribution. This is easy to determine using our formula (equation 7.19) for the standard deviation of a function of different quantities that are normally distributed.

The sample mean is given by a function of the various measured quantities:

$$\bar{x} = \bar{x}(x_1, x_2, \dots, x_N) = \frac{1}{N} \sum_{i=1}^{i=N} x_i = \frac{x_1 + x_2 + \dots + x_N}{N} \quad (7.33)$$

Each x_i in the sum has the same uncertainty, given by the sample standard deviation, σ . Using our equation 7.19 for the standard deviation of a function of normally distributed values, we have:

$$\sigma_{\bar{x}}^2 = \left(\frac{\partial \bar{x}}{\partial x_1} \sigma \right)^2 + \left(\frac{\partial \bar{x}}{\partial x_2} \sigma \right)^2 + \dots + \left(\frac{\partial \bar{x}}{\partial x_N} \sigma \right)^2$$

All of the derivatives are the same:

$$\frac{\partial \bar{x}}{\partial x_i} = \frac{1}{N} \frac{\partial}{\partial x_i} (x_1 + x_2 + \dots x_N) = \frac{1}{N}$$

The standard deviation of the mean is thus given by:

$$\begin{aligned} \sigma_{\bar{x}}^2 &= \left(\frac{\partial \bar{x}}{\partial x_1} \sigma \right)^2 + \left(\frac{\partial \bar{x}}{\partial x_2} \sigma \right)^2 + \dots + \left(\frac{\partial \bar{x}}{\partial x_N} \sigma \right)^2 \\ &= \left(\frac{1}{N} \sigma \right)^2 + \left(\frac{1}{N} \sigma \right)^2 + \dots + \left(\frac{1}{N} \sigma \right)^2 \\ &= N \times \left(\frac{1}{N} \sigma \right)^2 \\ &= \frac{1}{N} \sigma^2 \end{aligned} \quad (7.34)$$

which gives the expected result:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{N}} \quad (7.35)$$

Figure 7.8 shows an illustration of determining the parameters for a normal distribution using data. The histogram corresponds to 40 measurements that are normally distributed and that were sampled from a normal distribution (shown in black) with a “true” mean of 10 and a “true” standard deviation of 1. The sample mean of the measurements is 9.8 and is close to the true mean. The sample standard deviation of the data is 1.0, which is coincidentally exactly equal to the true standard deviation of the normal distribution. Finally, the error on the mean was found to be 0.2, which is representative of the error in our estimate of the true mean of the distribution. The red line shows a normal distribution drawn with the mean and standard deviation estimated from the data. The vertical lines show the range in the mean estimated from the data that comes from the error on the mean.

The error on the mean is often confused with the standard deviation. Given many data points, the standard deviation is a measure of how spread out the individual points are. The error on the mean is representative of our ability to determine the average value of those points. Even if the points are very spread out (large standard deviation), if we have many of them, we can precisely determine the mean value (small error on the mean).

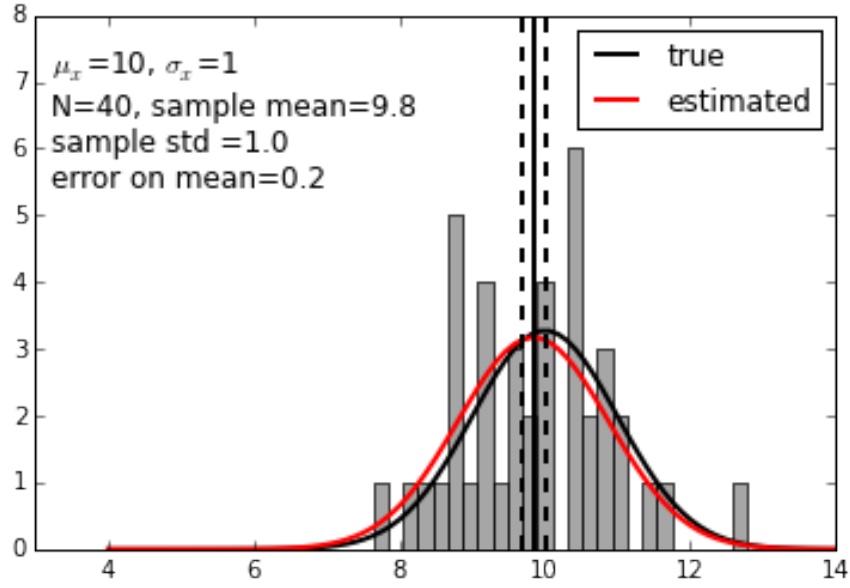


Figure 7.8: Histogram of 40 measurements that originate from a normal distribution (black curve) compared to a normal distribution (red curve) evaluated using the parameters estimated from the data. The vertical lines show the range of the estimated mean given by the error on the mean.

7.5 Averaging multiple measurements with standard errors

Suppose that a quantity, X , has been measured two different times with values $x_a \pm \sigma_a$ and $x_b \pm \sigma_b$. Perhaps the measurements were performed by two different teams, or perhaps they were performed by the same team at different times. Both measurements are quoted with uncertainties that are understood to be “standard” (corresponding to the standard deviation of a normal distribution), and both quoted results may themselves be the mean and standard deviation of multiple measurements. It should appear reasonable that there exists a way to combine the two results into a more accurate estimate, \hat{X} , of the unknown value X .

Let us assume that the two measurements, x_a and x_b , are consistent with each other; that is, the difference $|x_a - x_b|$ is not much larger than either σ_a or σ_b . If that were not the case, then we may want to reconsider combining the two measurements, since one of them (or both) is likely affected by a systematic effect. If the two measurements have similar uncertainties, $\sigma_a \approx \sigma_b$, then it would be reasonable to average the two measurements and use our rules for propagating the errors on normally distributed quantities:

$$\begin{aligned}
 \hat{X} &= \frac{1}{2}(x_a + x_b) \\
 \hat{\sigma}_X^2 &= \left(\frac{1}{2}\sigma_a\right)^2 + \left(\frac{1}{2}\sigma_b\right)^2 = \frac{1}{4}(\sigma_a^2 + \sigma_b^2) \\
 \therefore \hat{\sigma}_X &= \frac{1}{2}\sqrt{\sigma_a^2 + \sigma_b^2}
 \end{aligned} \tag{7.36}$$

If the two measurements have the same uncertainty, $\sigma_a = \sigma_b$, then the uncertainty on the average is given by:

$$\hat{\sigma}_X = \frac{1}{2}\sqrt{2\sigma_a^2} = \frac{1}{\sqrt{2}}\sigma_a \quad (7.37)$$

which is smaller than the original uncertainty. Combining these two measurements did indeed reduce the uncertainty in our measurement of X .

Let us suppose that σ_b is much bigger than σ_a ; is it still reasonable to average the measurements? First, if the uncertainty on x_b is much bigger, then the measurement x_b is much less precise and it does not make sense to give it “equal weight” as the more precise measurement, x_a , when computing the average. We would expect that the true value of X is closer to x_a than to x_b , so averaging x_a and x_b is intuitively wrong. Second, if we apply this formula, then the resulting uncertainty, $\hat{\sigma}_X$, would be bigger than the original uncertainty on x_a . Combining measurements would result in a less precise estimate of our quantity which does not make sense; we at least intuitively expect that we can only improve our estimate of a given quantity by adding more data ². We must thus find a (rigorous) way to include the fact that we should not give as much importance to measurements with large uncertainties when combining measurements.

We can use the principle of maximum likelihood to derive a method for combining measurements whose errors are normally distributed. We assume that the probability for obtaining $x_{a(b)}$ is given by a normal distribution centred at the unknown value, X , with standard deviation $\sigma_{a(b)}$:

$$\begin{aligned} P(x_a) &\propto \frac{1}{\sigma_a} e^{-\frac{(x_a - X)^2}{2\sigma_a^2}} \\ P(x_b) &\propto \frac{1}{\sigma_b} e^{-\frac{(x_b - X)^2}{2\sigma_b^2}} \end{aligned} \quad (7.38)$$

where we have neglected the normalization factor for the probability, since we will only care about maximizing the result (rather than its absolute value as a probability).

We can evaluate the joint probability of obtaining both measurements, given by the product

²If our two measurements both have small uncertainties and are inconsistent with each other, then we would expect to have a larger uncertainty on the combined measurement. This does occur when we are unable to identify any systematic effect that is leading to the discrepancy and we are effectively forced to include that discrepancy into the uncertainty

of the two probabilities:

$$\begin{aligned}
 P(x_a, x_b|X) &\propto P(x_a)P(x_b) \\
 &= \frac{1}{\sigma_a} e^{-\frac{(x_a-X)^2}{2\sigma_a^2}} \frac{1}{\sigma_b} e^{-\frac{(x_b-X)^2}{2\sigma_b^2}} \\
 &= \frac{1}{\sigma_a \sigma_b} e^{-\left(\frac{(x_a-X)^2}{2\sigma_a^2} + \frac{(x_b-X)^2}{2\sigma_b^2}\right)}
 \end{aligned} \tag{7.39}$$

and we can use this to define our estimate, \hat{X} , to be the value of X that maximizes this joint probability (also called the likelihood of our measurements). Again, instead of maximizing the likelihood, we can minimize the negative of the logarithm of the likelihood, given by:

$$\begin{aligned}
 -\ln P(x_a, x_b|X) &= -\ln \left(\frac{1}{\sigma_a \sigma_b} e^{-\left(\frac{(x_a-X)^2}{2\sigma_a^2} + \frac{(x_b-X)^2}{2\sigma_b^2}\right)} \right) \\
 &= \ln \sigma_a + \ln \sigma_b + \left(\frac{(x_a-X)^2}{2\sigma_a^2} + \frac{(x_b-X)^2}{2\sigma_b^2} \right) \\
 &= \ln \sigma_a + \ln \sigma_b + \frac{1}{2} \chi^2
 \end{aligned} \tag{7.40}$$

where we have introduced the quantity “chi-squared”, χ^2 :

$$\chi^2 \equiv \frac{(x_a - X)^2}{\sigma_a^2} + \frac{(x_b - X)^2}{\sigma_b^2} \tag{7.41}$$

Finding the value of X that maximizes the likelihood is thus equivalent to finding the value that minimizes the chi-squared (since the other terms in the log-likelihood do not depend on X and σ_a and σ_b are constants). The task of minimizing a chi-squared comes up often and is sometimes called the “method of least squares”, since one is trying to find X such that the squared distance to other quantities (x_a and x_b in our case) is minimized.

We can minimize χ^2 analytically in our case, by taking the derivative with respect to X :

$$\begin{aligned}
 \frac{d\chi^2}{dX} &= \frac{d}{dX} \left(\frac{(x_a - X)^2}{\sigma_a^2} + \frac{(x_b - X)^2}{\sigma_b^2} \right) \\
 &= -2 \frac{(x_a - X)}{\sigma_a^2} - 2 \frac{(x_b - X)}{\sigma_b^2}
 \end{aligned} \tag{7.42}$$

Setting this equal to zero, we find:

$$\begin{aligned}
 \frac{(x_a - X)}{\sigma_a^2} + \frac{(x_b - X)}{\sigma_b^2} &= 0 \\
 \hat{X} \left(\frac{1}{\sigma_a^2} + \frac{1}{\sigma_b^2} \right) &= \frac{x_a}{\sigma_a^2} + \frac{x_b}{\sigma_b^2}
 \end{aligned} \tag{7.43}$$

If we introduce “weights”, $w_a \equiv \frac{1}{\sigma_a^2}$ and $w_b \equiv \frac{1}{\sigma_b^2}$, we can write this as:

$$\hat{X} = \frac{w_a x_a + w_b x_b}{w_a + w_b} \quad (7.44)$$

and we find that our estimate, \hat{X} , obtained by combining the measurements x_a and x_b , is given by the weighted average of x_a and x_b ; the weights are given by 1 over the square of the uncertainty on the measurements. If the uncertainty on a measurement is large, then the weight of that measurement will be correspondingly smaller; this is exactly the property that we were after. This result is easily extended when averaging N measurements, $\{x_1, x_2, \dots, x_N\}$, with corresponding uncertainties, $\{\sigma_1, \sigma_2, \dots, \sigma_N\}$.

$$\begin{aligned} \hat{X} &= \left(\frac{1}{\sum_{i=1}^{i=N} w_i} \right) \sum_{i=1}^{i=N} w_i x_i \\ w_i &\equiv \frac{1}{\sigma_i^2} \end{aligned} \quad (7.45)$$

Since we now have a function for the weighted average of several quantities, we can apply our error propagation formula to obtain the resulting uncertainty in \hat{X} :

$$\hat{\sigma}^2 = \left(\frac{\partial \hat{X}}{\partial x_1} \sigma_1 \right)^2 + \left(\frac{\partial \hat{X}}{\partial x_2} \sigma_2 \right)^2 + \dots + \left(\frac{\partial \hat{X}}{\partial x_N} \sigma_N \right)^2 \quad (7.46)$$

The derivatives are all similar:

$$\frac{\partial \hat{X}}{\partial x_j} = \left(\frac{1}{\sum_{i=1}^{i=N} w_i} \right) w_j = \left(\frac{1}{\sum_{i=1}^{i=N} w_i} \right) \frac{1}{\sigma_j^2} \quad (7.47)$$

The uncertainty is thus given by:

$$\begin{aligned} \hat{\sigma}^2 &= \left(\left(\frac{1}{\sum_{i=1}^{i=N} w_i} \right) \frac{1}{\sigma_1^2} \sigma_1 \right)^2 + \left(\left(\frac{1}{\sum_{i=1}^{i=N} w_i} \right) \frac{1}{\sigma_2^2} \sigma_2 \right)^2 + \dots + \left(\left(\frac{1}{\sum_{i=1}^{i=N} w_i} \right) \frac{1}{\sigma_N^2} \sigma_N \right)^2 \\ &= \left(\frac{1}{\sum_{i=1}^{i=N} w_i} \right)^2 \left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2} + \dots + \frac{1}{\sigma_N^2} \right) \\ &= \left(\frac{1}{\sum_{i=1}^{i=N} w_i} \right)^2 \sum_{i=1}^{i=N} w_i \\ &= \frac{1}{\sum_{i=1}^{i=N} w_i} \end{aligned} \quad (7.48)$$

Therefore, the uncertainty on our estimate of the average, \hat{X} , is given by:

$$\hat{\sigma} = \frac{1}{\sqrt{\sum_{i=1}^{i=N} w_i}} \quad (7.49)$$

Example 7-5: The ATLAS and CMS experiments at the Large Hadron Collider measured the mass of the Higgs boson by looking for the decay of the Higgs into two gamma rays (amongst other channels), and obtained $m_H^{ATLAS} = 126.02 \pm 0.51$ GeV and $m_H^{CMS} = 124.70 \pm 0.34$ GeV, respectively. Both uncertainties contain a contribution from systematic and random errors. Assuming that all of the uncertainties are in fact random (and standard), and that the measurements are considered as being consistent with each other, what is the best estimate and uncertainty of the Higgs mass when combining the results from these two experiments?

First, we compute the weight for each measurement:

$$w^{ATLAS} = \frac{1}{0.51^2} = 3.84$$

$$w^{CMS} = \frac{1}{0.34^2} = 8.65$$

which we then use to compute the weighted average and uncertainty as:

$$\hat{m}_H = \frac{3.84 \times 126.020 + 8.65 \times 124.70}{3.84 + 8.65} = 125.11$$

$$\hat{\sigma} = \frac{1}{\sqrt{3.84 + 8.65}} = 0.28$$

The best estimate from combining these two measurements as if they had random errors and were completely independent, is $m_H = 125.11 \pm 0.28$, which has an uncertainty that is smaller than that of the individual measurements.

7.6 Summary

In this chapter, we introduced the probability density function for the normal distribution:

$$P^{norm}(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (7.50)$$

which has two parameters: the mean (μ) and the standard deviation (σ). The normal (or gaussian) probability density function gives the probability, $P^{norm}(x, \mu, \sigma)dx$, for a continuous variable, x , to be between x and $x + dx$.

The sample mean and sample standard deviation of a set of measurements that are normally distributed coincide with the mean and standard deviation of the normal distribution.

The probability density function (pdf) for the normal distribution is normalized such that the total area under the curve is equal to 1. The area underneath the curve between $\mu - \sigma$ and $\mu + \sigma$ is 0.68. A number drawn at random from the normal distribution thus has a 68% chance of being within 1 standard deviation of the mean.

The binomial and Poisson distribution are both approximated by the normal distribution as their mean value increases. Since many measurements are described by a binomial process

(e.g. a large number of small random errors adding up), it follows that most measurements are expected to follow a normal distribution. The Central Limit Theorem furthermore states that the sum of a large number of values drawn from any distribution is also normally distributed. This leads to the normal distribution appearing in many areas of science.

When quoting a number with an uncertainty, e.g. $x \pm \sigma_x$, it is usually reasonable to define the uncertainty, σ_x , as representing the standard deviation of a normal distribution with mean x . We thus imply that there is a 68% chance that the true value of x is within the range $x \pm \sigma_x$.

If the best estimate (x) and uncertainty (σ_x) in a quantity are the mean and standard deviation of a normal distribution, then we can recover all of the error propagation formulas that we presented in Chapter 4. In particular, we showed that the sum of two normally distributed quantities is normally distributed about the sum of the means of the two values, with a standard deviation that is found by adding in quadrature the standard deviation of the two quantities. We showed that “adding in quadrature” is really the result of combining different quantities that are normally distributed. We showed that for an arbitrary function, $F(x, y)$, of *independent*, and normally distributed values, x and y , that F is also normally distributed with mean and standard deviation given by:

$$\begin{aligned}\mu_F &= F(\mu_x, \mu_y) \\ \sigma_F^2 &= \left(\frac{\partial F}{\partial x} \sigma_x \right)^2 + \left(\frac{\partial F}{\partial y} \sigma_y \right)^2\end{aligned}\tag{7.51}$$

as long as the error in x and y given by σ_x and σ_y are small. We extended this formula for the general case where x and y are not necessarily normally distributed and independent and found that the best estimate of $F(x, y)$ and uncertainty in the function are given by:

$$\begin{aligned}\bar{F} &= F(\bar{x}, \bar{y}) \\ \sigma_F &= \sqrt{\left(\frac{\partial F}{\partial x} \sigma_x \right)^2 + \left(\frac{\partial F}{\partial y} \sigma_y \right)^2 + 2 \frac{\partial F}{\partial x} \frac{\partial F}{\partial y} \sigma_{xy}}\end{aligned}\tag{7.52}$$

where σ_x and σ_y are the standard deviations of measurements of x and y , respectively, and σ_{xy} is the covariance factor between x and y . The result is easily extended to more than two variables, by including covariance factors for each pair of variables.

We showed how to use normally distributed data to determine estimates for the mean and standard deviation of a normal distribution that describes those data. We showed that the sample mean and the square root of the sample variance give estimates of the mean and standard deviation of the normal distribution. We showed that the formula for the error on the mean gives the correct error on the estimate of the mean of the normal distribution.

Finally, we used the principle of maximum likelihood to show how to combine N measurements $\{x_1, x_2, \dots, x_N\}$, with corresponding uncertainties, $\{\sigma_1, \sigma_2, \dots, \sigma_N\}$, into an average

measurement, \hat{X} , with uncertainty $\hat{\sigma}$:

$$\begin{aligned}\hat{X} &= \left(\frac{1}{\sum_{i=1}^{i=N} w_i} \right) \sum_{i=1}^{i=N} w_i x_i \\ \hat{\sigma} &= \frac{1}{\sqrt{\sum_{i=1}^{i=N} w_i}} \\ w_i &\equiv \frac{1}{\sigma_i^2}\end{aligned}\tag{7.53}$$

which corresponds to a weighted average where each measurement is weighed by the inverse of the square of the uncertainty in the measurement.

8

Statistics - Modelling Data

In this chapter, we examine the topic of modeling data. We consider separately the aspects of parameter estimation and of model testing. In parameter estimation, we assume that we have a parametric model that can describe our data, and our goal is to determine the parameters of the model. For example, we may expect a linear relationship between two measured quantities and wish to determine the corresponding constant of proportionality (fitting the data with the model). In model testing, we assume that we have a model, and we wish to verify if the data are consistent with the model. For example, we may have data that look like they are linearly related, but we would like to quantify if that is the case.

8.1 Fitting a straight line

8.1.1 Linear problems

It is often possible to model two quantities in an experiment as being linearly related. For example, suppose that we measure the time, t , that it takes for objects to fall a certain distance, x . From Newton's Second Law, we expect to have:

$$x = \frac{1}{2}gt^2 \tag{8.1}$$

which is not a linear relationship between our measured quantities, x and t . However, if every time that we measure t , we then take the square of the value, we can turn this into a linear relationship between x and t^2 , with a slope of $\frac{1}{2}g$. This can be done with a variety of relations that do not appear to be linear at first sight.

In such an experiment, we may have a series of measurements as in Table 8.1, where we varied the drop height, x , from which an object was dropped, and measured the corresponding time, t , for it to fall. For each drop height, we measured t multiple times so that we could determine a mean drop time, \bar{t} , with a corresponding standard deviation, σ_t , and error on the mean, $\sigma_{\bar{t}}$. We repeated the measurements at each height until we obtained an error on the mean time of 0.1 s, which we propagated to t^2 . Let us suppose that we found that the error on the height, x , was negligible.

height, x (m)	time, \bar{t} , (s)	$\sigma_{\bar{t}}$ (s)	t^2 (s ²)	$\sigma_{t^2} = 2t\sigma_{\bar{t}}$ (s ²)
1.0	1.1	0.1	1.2	0.2
2.0	1.3	0.1	1.6	0.3
3.0	1.3	0.1	1.7	0.3
4.0	1.5	0.1	2.2	0.3
5.0	1.5	0.1	2.3	0.3
6.0	1.6	0.1	2.4	0.3
7.0	1.8	0.1	3.1	0.4
8.0	1.8	0.1	3.3	0.4
9.0	1.8	0.1	3.1	0.4
10.0	1.9	0.1	3.7	0.4

Table 8.1: Example measurements of the time, \bar{t} , that it takes to fall a certain distance, x .

Since we varied x while measuring t , and we assumed that the uncertainty in x is small, it makes more sense to plot t^2 vs x , rather than x vs t^2 , and so we model the data as:

$$t^2 = \frac{2}{g}x \quad (8.2)$$

The data from Table 8.1 are plotted in Figure 8.1. The slope of the data in this graph is expected to be $\frac{2}{g}$. We will show in this chapter how to determine the slope and uncertainty on the slope from these data.

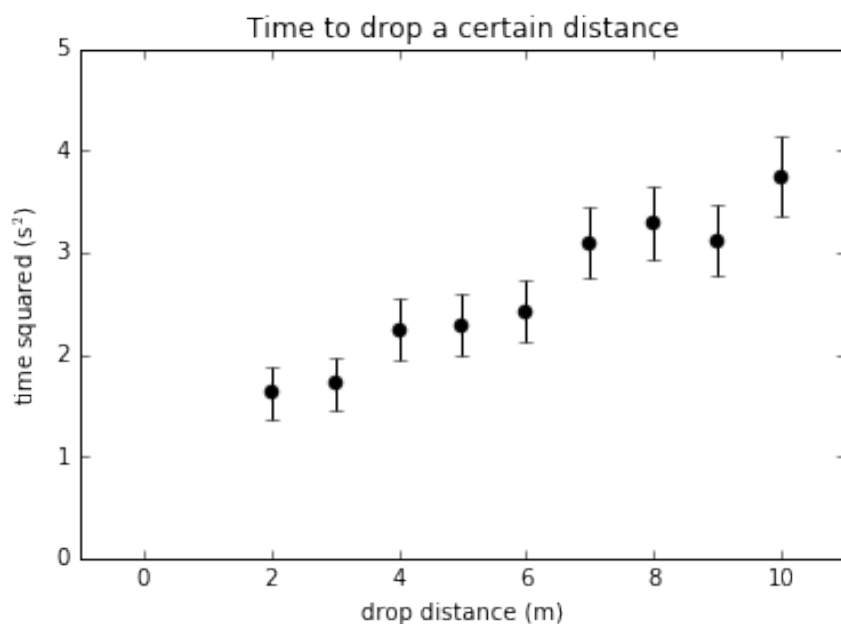


Figure 8.1: Measured time (squared) to fall a certain distance, using data from Table 8.1.

8.1.2 The Method of Least Squares

Given a set of N pairs of measurements $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$, we wish to determine the best estimates (and uncertainties) for the slope, m , and offset, b , of a straight line:

$$y = mx + b \quad (8.3)$$

that best matches the data. For simplicity, we assume that x can be measured with negligible uncertainty compared to the uncertainties on y . Furthermore, we assume that each value of y , y_i , have an uncertainty, σ_{yi} .

Equation 8.3 is our model for the data. At a given value of $x = x_i$, we expect that we should measure $y^{exp}(x_i) = mx_i + b$. However, since we cannot measure y exactly (due to random errors in our measurement), we will measure a value y_i that is close to the expected value. If the errors in our measurement are truly random, and we measure y many times for a fixed value of $x = x_i$, then we expect a normal distribution of y_i with a mean of $y^{exp}(x_i)$ and a standard deviation, σ_{yi} . Given a value of $x = x_i$ and our model, the probability¹ of measuring a specific value, y_i , is given by the normal distribution:

$$P(x_i, y_i) = \frac{1}{\sigma_{yi}\sqrt{2\pi}} e^{-\frac{(y_i - y^{exp}(x_i))^2}{2\sigma_{yi}^2}} \quad (8.4)$$

where the mean of the distribution is given by $y^{exp}(x_i) = mx_i + b$ for the model in equation 8.3.

In order to estimate the parameters of our model, m and b , we use the principle of maximum likelihood to find the value of m and b that yield the highest joint probability (i.e. likelihood) of obtaining our set of measurements. The joint probability of obtaining our data set, given values of m and b , is given by:

$$L(x, y|m, b) = P(x_1, x_2, \dots, x_N, y_1, y_2, \dots, y_N|m, b) \propto \prod_{i=1}^{i=N} e^{-\frac{(y_i - y^{exp}(x_i))^2}{2\sigma_{yi}^2}}$$

where we have ignored the normalization constant $\frac{1}{\sigma_{yi}\sqrt{2\pi}}$ since it does not depend on m and b and we are only interested in the maximal value in terms of m and b . As we did in the previous chapter, we can take the negative of the logarithm of the likelihood (which we will then want to minimize):

$$-\ln L(x, y|m, b) = \frac{1}{2} \sum_{i=1}^{i=N} \frac{(y_i - y^{exp}(x_i))^2}{\sigma_{yi}^2} \quad (8.5)$$

¹Although this is technically not a probability (it is a probability density), the ultimate goal will be to maximize the probability, which is also maximized when the probability density is maximized.

We can focus on minimizing the chi-squared, χ^2 , given by:

$$\chi^2 = \sum_{i=1}^{i=N} \frac{(y_i - y^{exp}(x_i))^2}{\sigma_{yi}^2} \quad (8.6)$$

The chi-squared is related to the original likelihood by:

$$L(x, y|m, b) \propto e^{-\frac{\chi^2}{2}} \quad (8.7)$$

Before proceeding, we can pause to think of what it means to minimize χ^2 . The formula for chi-squared is a sum over all data points of the square distance between the point, y_i , and the model prediction, $y^{exp}(x_i)$. That squared distance is then divided (or scaled) by the uncertainty on the point. Thus, if the points are on average 1 standard deviation away from the model prediction, then they will add approximately 1 unit to the chi-squared. For a good model, we expect the chi-squared to be about equal to the number of points. It should be apparent why the method is called the “method of least squares”, since we are effectively minimizing the squared distance between the model and the data when minimizing the chi-squared.

We assume that each point y_i have the same uncertainty, $\sigma_{yi} = \sigma_y$. Inserting our model explicitly for $y^{exp}(x_i)$:

$$\chi^2 = \sum_{i=1}^{i=N} \frac{(y_i - mx_i - b)^2}{\sigma_y^2} \quad (8.8)$$

By taking the derivative of χ^2 with respect to m and b , we can find the values that lead to a minimum:

$$\frac{d\chi^2}{dm} = \frac{d}{dm} \sum_{i=1}^{i=N} \frac{(y_i - mx_i - b)^2}{\sigma_y^2} = \frac{-2}{\sigma_y^2} \sum_{i=1}^{i=N} x_i(y_i - mx_i - b) \quad (8.9)$$

$$\frac{d\chi^2}{db} = \frac{d}{db} \sum_{i=1}^{i=N} \frac{(y_i - mx_i - b)^2}{\sigma_y^2} = \frac{-2}{\sigma_y^2} \sum_{i=1}^{i=N} (y_i - mx_i - b) \quad (8.10)$$

Setting these two equations equal to zero:

$$\sum_{i=1}^{i=N} x_i(y_i - mx_i - b) = \sum_{i=1}^{i=N} x_i y_i - m \sum_{i=1}^{i=N} x_i^2 - b \sum_{i=1}^{i=N} x_i = 0 \quad (8.11)$$

$$\sum_{i=1}^{i=N} (y_i - mx_i - b) = \sum_{i=1}^{i=N} y_i - m \sum_{i=1}^{i=N} x_i - Nb = 0 \quad (8.12)$$

and rearranging:

$$m \sum_{i=1}^{i=N} x_i^2 + b \sum_{i=1}^{i=N} x_i = \sum_{i=1}^{i=N} x_i y_i \quad (8.13)$$

$$m \sum_{i=1}^{i=N} x_i + Nb = \sum_{i=1}^{i=N} y_i \quad (8.14)$$

which is a system of linear equations that is easily solved for m and b :

$$\begin{aligned} m &= \frac{N \sum xy - \sum x \sum y}{N \sum x^2 - (\sum x)^2} \\ b &= \frac{\sum x^2 \sum y - \sum x \sum xy}{N \sum x^2 - (\sum x)^2} \end{aligned} \quad (8.15)$$

where we have omitted the indices on the x_i and y_i and we have implied that the sums are from $i = 1$ to $i = N$. Now that we have formulas for the m and b , we can propagate the errors to obtain:

$$\begin{aligned} \sigma_m &= \sigma_y \sqrt{\frac{N}{N \sum x^2 - (\sum x)^2}} \\ \sigma_b &= \sigma_y \sqrt{\frac{\sum x^2}{N \sum x^2 - (\sum x)^2}} \end{aligned} \quad (8.16)$$

Example 8-1: Use equations 8.15 and 8.16 to determine the slope and offset for a line that best fits the data in Figure 8.1, and thus determine a value of the gravitational constant g with its uncertainty from the data.

We want to plot the data of t^2 as a function of x , and determine the slope and offset of those data. The slope will be equal to $\frac{2}{g}$ (equation 8.2), and can thus be used to determine our measured value of g . We can use equations 8.15 and 8.16 to determine the slope and offset, and then use error propagation to convert the error in the slope into an error in g . If the error in the slope is σ_m , the error in g , σ_g , will be given by:

$$\begin{aligned} g &= \frac{2}{m} \\ \sigma_g &= \sqrt{\left(\frac{dg}{dm} \sigma_m\right)^2} = \frac{2}{m^2} \sigma_m \end{aligned}$$

The rest of the calculations are done easily in python:

Python Code 8.1: Linear fit with least squares

```
import numpy as np
from math import *
import pylab as pl
import scipy.stats as stats
%matplotlib inline

#Copy and paste data from the table in the notes, add commas:
datalist=[
1.0, 1.1, 0.1, 1.2, 0.2,
2.0, 1.3, 0.1, 1.6, 0.3,
3.0, 1.3, 0.1, 1.7, 0.3,
4.0, 1.5, 0.1, 2.2, 0.3,
```

```

5.0, 1.5, 0.1, 2.3, 0.3,
6.0, 1.6, 0.1, 2.4, 0.3,
7.0, 1.8, 0.1, 3.1, 0.4,
8.0, 1.8, 0.1, 3.3, 0.4,
9.0, 1.8, 0.1, 3.1, 0.4,
10.0, 1.9, 0.1, 3.7, 0.4,
]
#convert this to a numpy array, and change the shape
data=np.array(datalist)
data=data.reshape(10,5) #changes the shape to get the data in the same
    format as table
#Get the columns out of the table:
x=data[:,0]#This is the first column of data (x)
t2=data[:,3]#The 4th column (t^2)
sigma_t2=data[:,4] #The 5th column, sigma_t^2

#calculate the slope and offset
#start by calculating all of the sums
N=x.size
x2sum=(x*x).sum()
xt2sum=(x*t2).sum()
xsum=x.sum()
t2sum=t2.sum()

#We will use the mean value of the sigma_t2 as the "error in y"

denominator=N*x2sum-xsum**2
#slope:
m=(N*xt2sum-xsum*t2sum)/denominator
sigma_m=sigma_t2.mean()*sqrt(N/denominator)# using mean error in t^2
#offset:
b=(x2sum*t2sum-xsum*xt2sum)/denominator
sigma_b=sigma_t2.mean()*sqrt(x2sum/denominator)# using mean error in t^2

gmeas=2.0/m
sigma_g=2/m/m*sigma_m

#The fit results
text=''Fit results: t^2$=mx+b
m= {:.3f} +/- {:.3f}
b= {:.3f} +/- {:.3f}
g= {:.1f} +/- {:.1f} ''.format(m, sigma_m, b, sigma_b, gmeas, sigma_g)

#calculate probability of obtaining a value this far away from the
    accepted value
dsigma=(9.81-gmeas)/sigma_g
prob=2.0*stats.norm.sf(dsigma)
print("Probability to get g {:.1f} sigma or further from accepted value:
    {:.3f}".format(dsigma, prob))

#plot:
#The data:
pl.errorbar(x, t2, yerr=sigma_t2, fmt='o', color='black')

```

```

#The fit:
pl.plot(x,m*x+b,color='red')
#The fit with m+/-sigma_c
pl.plot(x,(m+sigma_m)*x+b,'--',color='red')
pl.plot(x,(m-sigma_m)*x+b,'--',color='red')

pl.text(0,3,text,fontsize=12)#fit results onto the plot
pl.xlabel("x, drop distance (m)")
pl.ylabel("t^2, time squared (s^2)")
pl.title("Time to drop a certain distance")
pl.axis([-1,11,0,5])
pl.show()

```

8.1: Output

Probability to get g 2.2 sigma or further from accepted value: 0.027

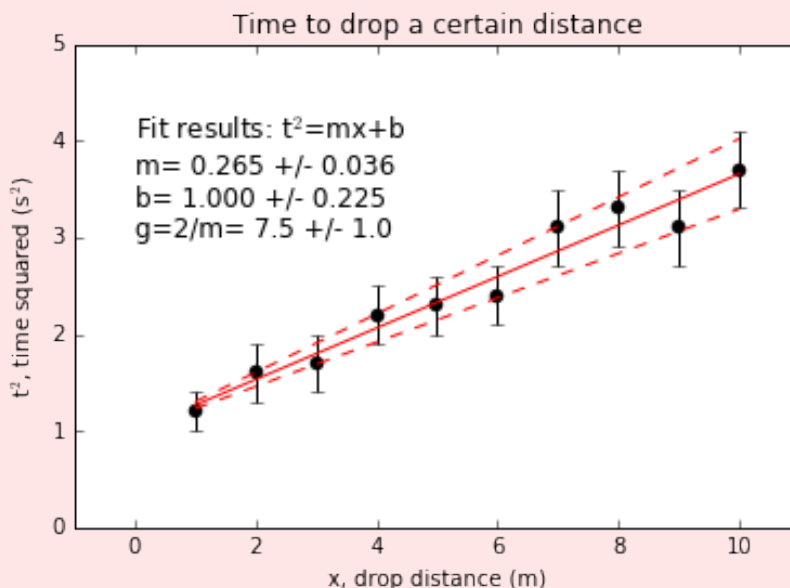


Figure 8.2: Measured time (squared) to fall a certain distance, using data from Table 8.1 and fit (solid red line) using equation 8.15. The slope of the line is also shown for the values of the slope 1 standard deviation away from the mean value.

The results of this code, are shown in Figure 8.2. Since our formulas in equations 8.15 and 8.16 assumed that all points had the same size vertical error bar, we used the mean of the values in the last column of table 8.1 as the error. We find that the measurement is $g = 7.5 \pm 1.0 \text{ m/s}^2$ which is more than one standard deviation away from the accepted value of g (9.81 m/s^2). The value is $\frac{9.81-7.5}{1.0}\sigma = 2.2\sigma$ away from the expected value, which has a 2.7% change of occurring. It is not clear if this is due to pure chance or if there is a systematic effect. Given the low probability of obtaining such a shifted result, one should suspect a systematic effect.

We should also note that the fit returned a non-zero value for the offset of the fit.

Although our model expects the offset to be zero, we have found a value of $b = 1.0 \pm 0.2 \text{ s}^2$ that is inconsistent with zero by 5σ . It is thus very likely that there is some systematic offset in the data leading to the time always being measured to be slightly high. Perhaps the person doing the timing systematically stopped the timer late, maybe due to their reaction time. This offset in the timing may also result in a systematic shift of the slope, leading to a systematic shift in the measured value of g .

We see that even if we expect there to be no offset in the model, it is always best to include it. If there is no real systematic effect, then the offset should fit to a value that is consistent with zero.

The analysis in example 8-1 highlights some interesting aspects of “fitting” a straight line to data using the least squares method (also called “linear regression”). In our physical model, 8.2, we would expect the data to fit to a straight line with a slope and no offset. It is generally true that even if the model does not have an offset, one should always fit a line with an offset to the data. If the data really are consistent with no offset, then the fitted value of the offset will be zero within its uncertainty. If the fitted offset is inconsistent with zero (as it was in the example by 5 standard deviations), then this is a useful tool to uncover a systematic effect. You should thus always include an offset in the model.

8.1.3 Residuals

In order to get an idea of how well our model “fits” the data, one can look at the “residuals” of the fit. For a data point, (x_i, y_i) , with a model $y^{exp}(x)$, the residuals, $R(x_i)$, are defined as:

$$R(x_i) \equiv y_i - y^{exp}(x_i) \quad (8.17)$$

and correspond to the difference between the data and the model prediction. Again, if we assume that the errors in the x_i are negligible, then the error in the residuals are given by:

$$\sigma_{R(x_i)} = \sigma_{y_i} \quad (8.18)$$

That is, the error on the residuals are given by the error in the data points. We do not propagate the error in the fit parameters to the residuals, as we are interested in the trend of the residuals and their distance away from zero in terms of the error in the data points. If the model is a good representation of the data, then the residuals are expected to be symmetrically distributed about a mean of 0. If all of the residuals were positive or negative, then one would question whether the model is a good fit to the data. Similarly, if the residuals show a trend, then one would also suspect that there is an issue with the model. The residuals for the fit from example 8-1 are shown in Figure 8.3 and do not show any obvious trend. Within the uncertainties in the data points, we can conclude that a linear model is a good representation of the data. Note that in this particular experiment, we would expect that friction from drag would likely play an effect for high drop heights, and the residual plot may have helped us to identify this effect. The code to plot the residuals from example 8-1 in python is given by:

Python Code 8.2: Plotting residuals

```
#The residuals have the same error bars:
pl.errorbar(x,t2-(m*x+b),yerr=sigma_t2,fmt='o',color='black')
pl.xlabel("drop distance (m)")
pl.ylabel("residual from fit (s2)")
pl.title("Residuals from linear fit")
pl.axis([-1,11,-1,1])
pl.show()
```

8.2: Output

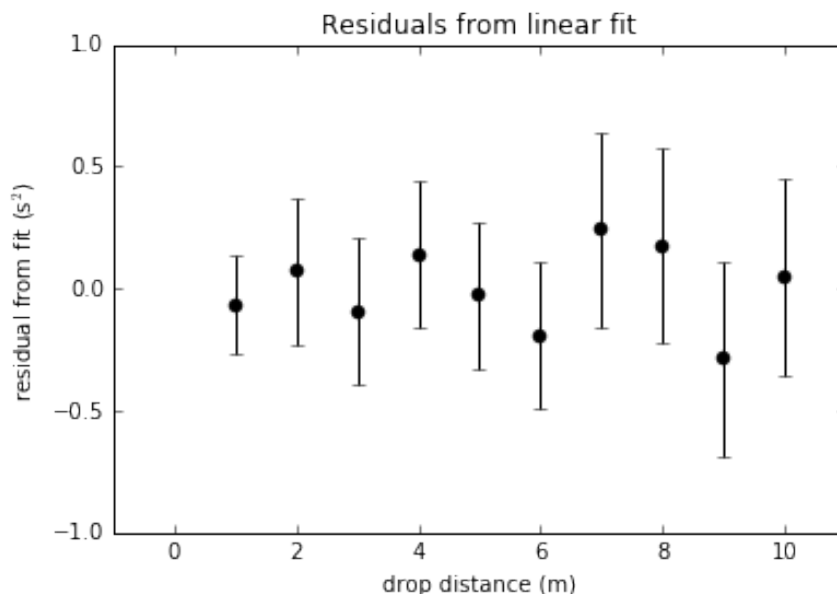


Figure 8.3: Residuals from the fit in Figure 8.2. There is no obvious trend in the residuals and we thus conclude the linear model is a reasonable representation of the data.

8.1.4 Correlated parameters

In the least squares fitting for example 8-1, we treated the slope, m , and offset, b , as independent parameters. While the minimum in the chi-squared can be found for each parameter independently, it is not clear that the parameters are actually independent from each other. That is, it is likely the case that as m changes, the corresponding value of b that minimizes the chi-squared changes as well. In other words, it is usually the case that the parameters of the fit are correlated, which is important in considering their errors, especially when they are propagated to a quantity, $f(m, b)$, that depends on both m and b . As you recall, we can only add errors in quadrature when propagating them if the various quantities with errors are independent of each other. If the value of m that minimizes the chi-squared depends on b , then m and b are not independent and we would need to know their covariance to propagate their errors (equation 5.20).

This is illustrated for the example 8-1 in Figure 8.4 which shows the value of chi-squared as a function of m for different values of b (at the best fit value of b and at the best fit value of b plus 1 standard deviation). When b is also at its best fit value ($b = 1.0$), we see that the minimum of the chi-squared indeed occurs when m is at its best fit value of $m = 0.265$;

however, this is no longer the case when $b = 1.2$. We thus say the parameters m and b are correlated (rather, in this case, they are anti-correlated, as a larger value of one parameter leads to a lower optimal value of the other parameter). The χ^2 is also shown in Figure 8.5 as a function of both m and b , and the tilt in the plot is a sign that the parameters are correlated.

The code to make the two plots in python is:

Python Code 8.3: Plot chi-squared as a function of the fit parameters

```
#A function to calculate chi-squared for a given set of data, errors on data,
    and model predictions
def chis(ydata, sigma_y, ymodel):
    return (((ydata-ymodel)/sigma_y)**2).sum()

ydata=t2
sigma_y=sigma_t2

#an array of values of b and m centered around the best fit value:
b_vals=np.linspace(b-2*sigma_b, b+2*sigma_b, 100)
m_vals=np.linspace(m-2*sigma_m, m+2*sigma_m, 100)

#Plot the chi-squared vs m, for 2 different values of b
pl.plot(m_vals, [chis(ydata, sigma_y, m_vals[i]*x+b) for i in range(m_vals.size)
], label="b={:.1f}".format(b))
pl.plot(m_vals, [chis(ydata, sigma_y, m_vals[i]*x+(b+sigma_b)) for i in range(
    m_vals.size)], label="b={:.1f}".format(b+sigma_b))
pl.legend(loc='best')
pl.xlabel('m')
pl.ylabel('$\chi^2$')
pl.title('Minimum of $\chi^2$ as a function of m for different values of b')
pl.show()

#Make a 2D plot of chi-squared vs m and b:
mm,bb=np.meshgrid(m_vals, b_vals)
chi2d=pl.zeros(mm.shape)
for i in range(m_vals.size):
    for j in range(b_vals.size):
        chi2d[i,j]=chis(ydata, sigma_y, m_vals[i]*x+b_vals[j])
pl.pcolormesh(mm,bb,chi2d, cmap='RdBu')
pl.axis([m_vals.min(), m_vals.max(), b_vals.min(), b_vals.max()])
pl.xlabel('m')
pl.ylabel('b')
pl.title("$\chi^2$ as a function of m and b")
pl.colorbar()
pl.show()
```

8.3: Output

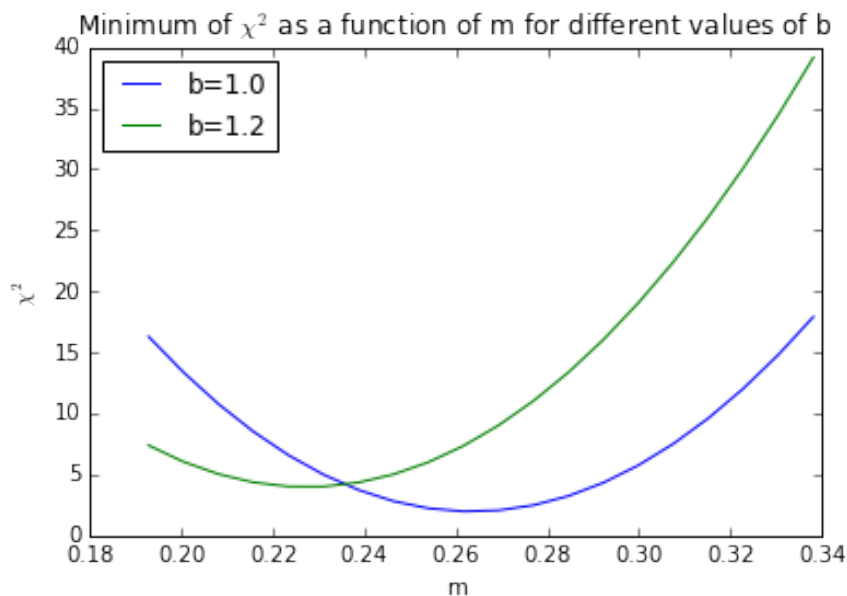


Figure 8.4: Chi-squared as a function of the slope from the fit in Figure 8.2 for different values of the offset. Since the minimum in chi-squared as a function of the slope, m , decreases for a higher value of the offset, b , we can see that the parameters from the fit are anti-correlated.

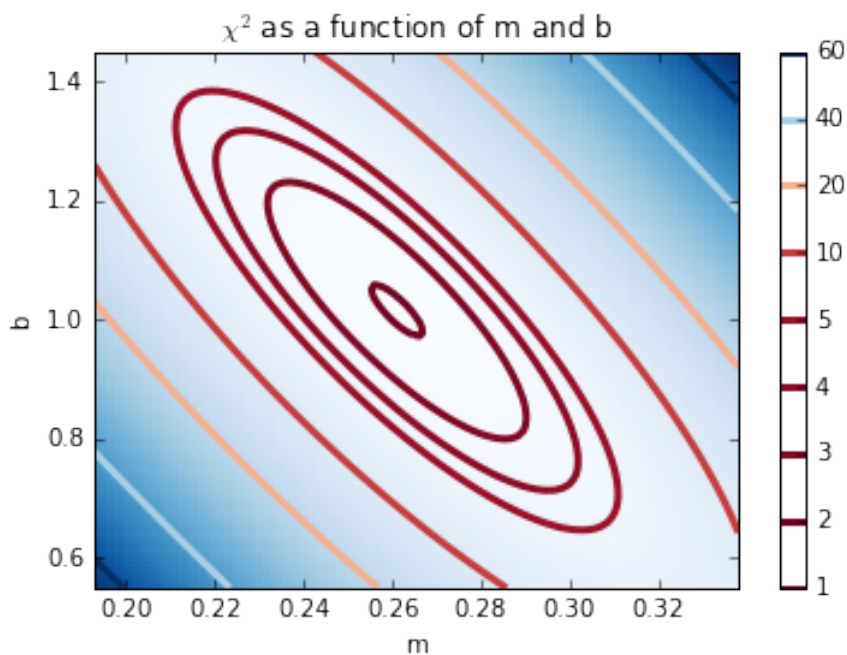


Figure 8.5: Chi-squared as a function of the slope and offset from the fit in Figure 8.2. The “tilt” in the chi-squared map is the result of the anti-correlation between the two parameters.

8.2 Non-linear fits

In the previous section, we used the principle of maximum likelihood and showed that minimizing the chi-squared, χ^2 , given by:

$$\chi^2 = \sum_{i=1}^{i=N} \frac{(y_i - y^{exp}(x_i))^2}{\sigma_{y_i}^2} \quad (8.19)$$

was equivalent to finding the maximum value of the likelihood. For the specific case of a linear model, $y^{exp}(x) = mx + b$, we found that the slope and offset could be obtained analytically by differentiating the formula for the chi-squared. It can be shown that for any polynomial model, and indeed any model that is linear in the parameters that we want to determine, a closed form solution can be obtained from a system of linear equations. However, for model functions that are not linear in the parameters, one must determine the best fit parameter values numerically. In general, the problem is thus to find the values for a set of parameters, $\vec{\beta} = \{\beta_1, \beta_2, \dots\}$, such that the chi-squared:

$$\chi^2 = \sum_{i=1}^{i=N} \frac{(y_i - y^{exp}(x_i, \vec{\beta}))^2}{\sigma_{y_i}^2} \quad (8.20)$$

between the data and the model is a minimum. Various clever minimization algorithms exist to do this efficiently rather than systematically exploring all of parameter space. These algorithms are usually based on estimating the derivatives of the chi-squared as a function of the different parameters and searching in the direction with negative derivatives.

As an example, let us suppose that we have measured the energy of many collisions in an accelerator, and made a histogram of the energy of the collisions, between, say, 0 MeV and 20 MeV, in 1 MeV bins (Figure 8.6). This is a typical situation in particle physics, where we would want to look for a resonance at a certain energy that would correspond to a specific particle being created. We expect the resonance to be a gaussian peak on top of a sloped background, and thus model the number of counts, n , as a function of energy, E , as:

$$n(E) = \beta_0 + \beta_1 E + \beta_4 \left(\frac{1}{\beta_3 \sqrt{2\pi}} e^{-\frac{(E-\beta_2)^2}{2\beta_3^2}} \right) \quad (8.21)$$

where we have used β_i for the 5 parameters, and labeled the first one with an index of 0 instead of 1, to be consistent with the code in python. Parameters 0 and 1 are thus the slope and offset of the background. Parameter 2 is the mean of the gaussian (the center of the resonance), parameter 3 is the width of the resonance (the standard deviation of the gaussian), and parameter 4 is the normalization of the gaussian (the number of events above background that are from the resonance).

Since we can view each bin as a separate measurement, we can assign each bin an uncertainty equal to the square root of the number of counts in that bin (as you recall, the uncertainty in a counted quantity is the square root of the quantity). We thus wish to fit our model to the histogram of the number of counts as a function of energy to determine the slope and offset of the background, as well as the mean, standard deviation and normalization of the gaussian. An example of such data is shown in Table 8.2 and plotted in Figure 8.6.

Energy, E	Counts in 1 MeV, n	Uncertainty, \sqrt{n}
1.0	4.0	2.0
2.0	3.0	1.7
3.0	2.0	1.4
4.0	3.0	1.7
5.0	5.0	2.2
6.0	2.0	1.4
7.0	4.0	2.0
8.0	2.0	1.4
9.0	10.0	3.2
10.0	16.0	4.0
11.0	12.0	3.5
12.0	6.0	2.4
13.0	3.0	1.7
14.0	3.0	1.7
15.0	3.0	1.7
16.0	3.0	1.7
17.0	5.0	2.2
18.0	5.0	2.2
19.0	2.0	1.4
20.0	5.0	2.2

Table 8.2: Simulated data for a histogram of energy observed in an accelerator.

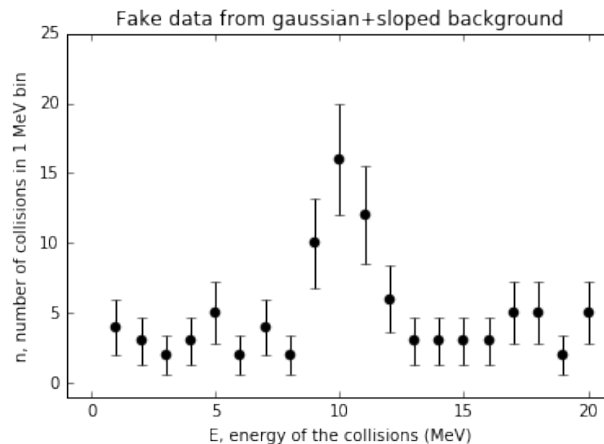


Figure 8.6: Simulated data from an accelerator experiment, showing the number of counts in 1 MeV bins as a function of energy, from table 8.2.

We will use the `scipy.optimize` module to fit the model to the data and to determine the value of the unknown parameters. The first step is specify the model in python as a function of the dependent variable (in our case the energy) and the unknown parameters. We can make the definition of our function slightly more general by passing the unknown parameters as a tuple:

Python Code 8.4: Defining a function for a non-linear fit

```
#Define a model for a gaussian + linear background, with 5 parameters:
def model(x, *pars):
    offset=pars[0]
    slope=pars[1]
    mu=pars[2] #mean
    sigma=pars[3] #sigma
    norm=pars[4] #normalization
    return offset+slope*x+norm*(1.0/sqrt(2.*pi*sigma**2)*np.exp(-((x-mu)**2)
    /2./sigma**2))
```

We can load the data into arrays, `xdata`, `ydata`, and `ysigma`, corresponding to the 3 columns in Table 8.2. The `curve_fit` function can then be used to fit the data for the unknown parameters:

Python Code 8.5: Non-linear fit in python

```
from scipy.optimize import curve_fit
#Try a guess for our parameters
#The fit does not converge if we give a crazy guess for the mean
guess_pars=[1,1,12,1,1]

#Run the fit to the data:
fit_pars, fit_cov = curve_fit(model,xdata,ydata,sigma=ysigma,p0=guess_pars)
```

where we have to “help” the algorithm a little by specifying reasonable guesses for our unknown parameters (in this case, the algorithm can have trouble if the guess for the mean

of the gaussian is unreasonable). Note that we can specify a different error bar for each data point (which we did by passing the y errors on each point using the array `ysigma`).

The `curve_fit` function returns a tuple with two components. The first component, `fit_pars`, is an array with the best fit values of the parameters (e.g. `fit_pars[0]` corresponds to the offset of the sloped background). `fit_cov` is the “covariance matrix” of the fit parameters. Each diagonal element in the covariance matrix corresponds to the variance of that parameter. Thus the square roots of the diagonal elements correspond to the standard deviations of each parameter estimate and can be used as the uncertainty in the corresponding parameter. The off-diagonal elements correspond to the covariance factors between pairs of parameters (for example, the element in the second row and third column of the matrix corresponds to the covariance between the second and third parameter in the fit). The covariance matrix is thus symmetric. If two parameters have a large covariance, then the determination of one of those parameters has a strong influence on the other.

The following python code performs a fit to the data in Table 8.2 using our model, produces a plot showing the data with the fit and the residuals (Figure 8.7), and calculates the chi-squared for the fit.

Python Code 8.6: Complete non-linear fit with residuals

```
import numpy as np
from math import *
import pylab as pl
from math import *
from scipy.optimize import curve_fit
import matplotlib.gridspec as gridspec # for unequal plot boxes

#Define a model for a gaussian + linear background, with 5 parameters:
def model(x, *pars):
    offset=pars[0]
    slope=pars[1]
    mu=pars[2] #mean
    sigma=pars[3] #sigma
    norm=pars[4] #normalization
    return offset+slope*x+norm*(1.0/sqrt(2.*pi*sigma**2)*np.exp(-((x-mu)**2)/2./sigma**2))

#Load the table data
data=np.array([1.0,4.0,2.0,
2.0,3.0,1.7,
3.0,2.0,1.4,
4.0,3.0,1.7,
5.0,5.0,2.2,
6.0,2.0,1.4,
7.0,4.0,2.0,
8.0,2.0,1.4,
9.0,10.0,3.2,
10.0,16.0,4.0,
11.0,12.0,3.5,
12.0,6.0,2.4,
```

```

13.0,3.0,1.7,
14.0,3.0,1.7,
15.0,3.0,1.7,
16.0,3.0,1.7,
17.0,5.0,2.2,
18.0,5.0,2.2,
19.0,2.0,1.4,
20.0,5.0,2.2])
#extract the data by column
data=data.reshape(20,3)
xdata=data[:,0]
ydata=data[:,1]
ysigma=data[:,2]

#Try a guess for our parameters
#The fit does not converge if we give a crazy guess for the mean
guess_pars=[1,1,12,1,1]

#Run the fit to the data:
fit_pars, fit_cov = curve_fit(model,xdata,ydata,sigma=ysigma,p0=guess_pars)

#Get the parameter errors out of the covariance matrix:
fit_err = np.sqrt(np.diag(fit_cov))

#Calculate value of y from the model using the fit parameters:
yfit=model(xdata,*fit_pars)

#Calculate the residuals:
yres=ydata-model(xdata,*fit_pars)

#Calculate the chi-squared between data and the fit
chisq=((ydata-yfit)/ysigma)**2).sum()
ndof=ydata.size-fit_pars.size-1

#These are the "true" parameters used to make the data (for comparison)
true_pars=[3.,0.1,10.,0.8,20.]

#Put the results into some text:
textfit="Fit (true) parameters,  $\chi^2 = {:.1f}$ , ndof={}\n".format(chisq,
    ndof)
textfit=textfit+"n(E)=p[0]+p[1]+p[4]*Gauss(E,p[2],p[3])\n"
for i in range(len(true_pars)):
    textfit = textfit + "Par {}: {:.2f} +/- {:.2f} ({:.2f})\n".format(i,
        fit_pars[i], fit_err[i], true_pars[i])

#Plot the fit and the residuals
pl.figure(figsize=(8,8))
gs = gridspec.GridSpec(2, 1, height_ratios=[3, 1])
pl.subplot(gs[0])
pl.plot(xi,model(xi,*fit_pars),lw=2,color='red')
pl.errorbar(xdata,ydata,yerr=ysigma,fmt='o',color='black')#data
pl.ylabel('n, number of collisions in 1 MeV bin')
pl.title('Fit to the fake data')
pl.text(-0.5,13,textfit,fontsize=12)

```

```

pl.axis([-1,22,-2,25])

pl.subplot(gs[1])
pl.errorbar(xdata,yres,yerr=ysigma,fmt='o',color='black')#residuals
pl.ylabel('residuals')
pl.xlabel('E, energy of the collisions (MeV)')
pl.axis([-1,22,-6,6])

pl.show()

```

8.6: Output

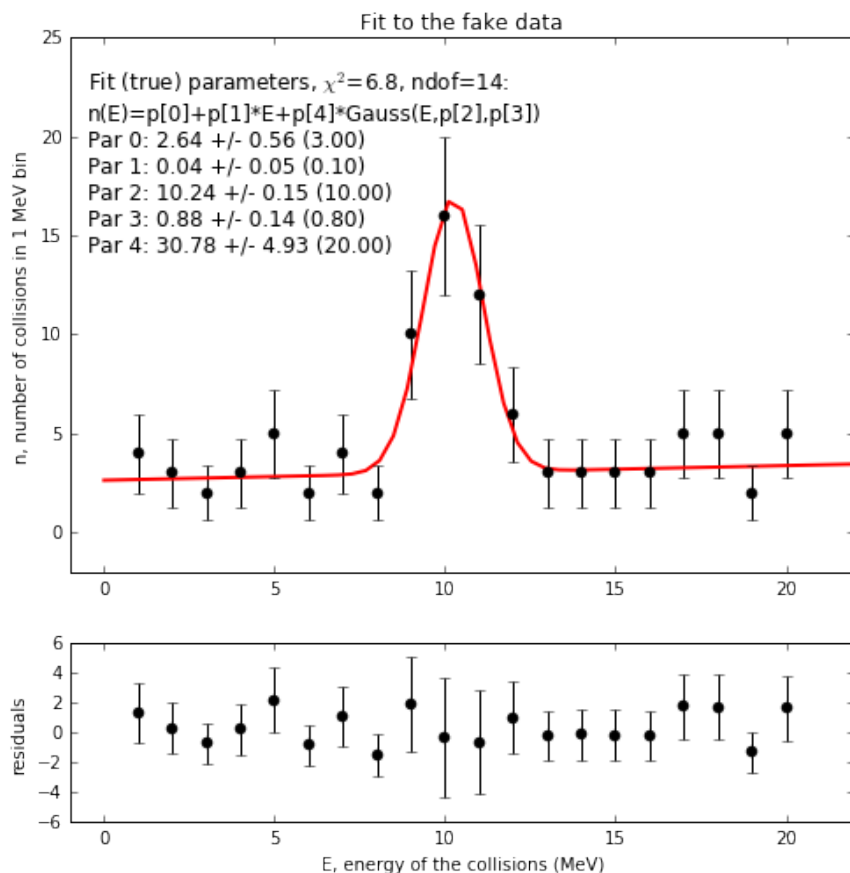


Figure 8.7: Fit to the data from Table 8.2 and residuals using the model in equation 8.21. The “true” value of the parameters used to make the data are shown in parentheses next to their fitted values.

By looking at the residuals of the fit, we see no evidence of a trend in the residuals that would indicate that this model is not a good fit for the data. We can conclude from the fit results that the data are consistent with a resonance near 10 MeV with a confidence level of 6σ , as evidenced by the uncertainty in the normalization of the gaussian (the fit indicates that the normalization of the gaussian is inconsistent with zero by 6 standard deviations).

8.3 Goodness of fit

So far, we have concerned ourselves with determining the parameters of a model that best fit a set of data. We have qualitatively looked at the residuals between the fit and the data to look for a trend in the residuals that would indicate that the model is inadequate. However, we have not determined quantitatively if the model is a *good* fit to the data. Since this is a vast subject, we will only limit ourselves to one method to evaluate the quality of a fit: the value of the χ^2 .

We know that we can determine the parameters of the model, $\vec{\beta}$, by minimizing the chi-squared between the data and the model. It turns out that we can use the value of the chi-squared evaluated using the best-fit parameters, to tell us about the goodness of the fit. Recall the definition of the chi-squared:

$$\chi^2 = \sum_{i=1}^{i=N} \frac{(y_i - y^{exp}(x_i, \vec{\beta}))^2}{\sigma_{y_i}^2} \quad (8.22)$$

As we discussed previously, we expect that for a good model, the chi-squared will increase by approximately 1 per data point. On average, we expect the data points to differ from the model prediction by approximately 1 standard deviation; thus each term in the sum contributes approximately 1 to the chi-squared.

We now introduce the “number of degrees of freedom”, ν , for the fit as:

$$\nu \equiv N - n - 1 \quad (8.23)$$

where N is the number of data points and n is the number of parameters that we are fitting, and the minus 1 is related to the $N - 1$ that we have in the definition of the variance. The number of degrees of freedom tells us something about how much we can vary the parameters in the model based on the number of data points. For example, if we have 2 data points we can always find a straight line (with two parameters, slope and offset) that will exactly go through those two points, and we really have no freedom to vary those parameters. In this situation, we have no “degrees of freedom” (in fact we have $\nu = -1$ because of the “minus 1” in the definition of ν). In a case with no (or few) degrees of freedom, we cannot really say much about how our model fits the data, since there is very little liberty in our model. Conversely, if our model appears to fit the data well when we have a large number of degrees of freedom, we can be more confident in our model.

Although it is beyond the scope of this text, one can show that the chi-squared should scale roughly as the number of degrees of freedom, rather than the actual number of data points. We can define the “reduced chi-squared”, as the chi-squared divided by the number of degrees of freedom. In the fit from Figure 8.7, we found a chi-squared of 6.8 for 14 degrees of freedom. The reduced chi-squared was thus $6.8/14=0.5$. If the reduced chi-squared was substantially bigger than 1, we would conclude that the model is not likely a good fit to the data. One should however be careful: a small reduced chi-squared does not necessarily imply that the model is a good fit! If you think about it, the denominator in the chi-squared

is the error on each data point. Thus, if there errors are very large, the chi-squared will be small, and we cannot conclude anything about the model since the error in the data are too large.

We can be quantitative about the expected value of chi-squared for a given number of degrees of freedom. Let us suppose that our 5 parameter model from equation 8.21 is in fact the correct model for the data from our experiment. That is, if we repeat the experiment a number of times, we will be able to fit the data to our model and obtain a “reasonable” chi-squared. However, each time that we repeat the experiment, due to statistical variations in the data, we will not obtain exactly the same chi-squared; rather, we will obtain a distribution of different values of chi-squared. The distribution of chi-squared that we obtain is well-understood and governed by the “chi-squared distribution”, given by the following probability density function:

$$P^{\chi^2}(x, \nu) = \frac{x^{\frac{\nu}{2}-1} e^{-\frac{x}{2}}}{2^{\frac{\nu}{2}} \Gamma\left(\frac{\nu}{2}\right)} \quad (8.24)$$

where $\Gamma()$ is called the “gamma function”. Figure 8.8 shows the chi-squared distribution for different number of degrees of freedom.

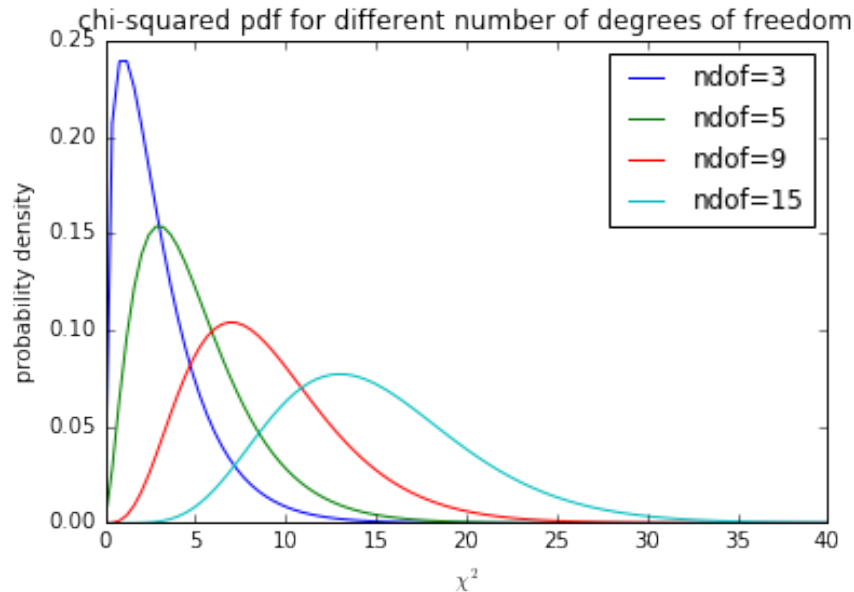


Figure 8.8: Probability density functions for the chi-squared distribution for different numbers of degrees of freedom.

The chi-squared distribution for 14 degrees of freedom is shown in Figure 8.9. For our example from Figure 8.7, we obtained a chi-squared of 6.8. By looking at the cumulative chi-square distribution in Figure 8.9, we can see that the probability of obtaining a $\chi^2 = 6.8$ or less is 6%. Conversely, there is a 94% chance of obtaining a chi-squared of 6.8 or bigger simply from statistical variations in the data if the model is correct. This effectively informs us about the quality of our fit. If the model is correct, we got lucky and obtained a very good

fit, and 94% of the time, we would get a worse chi-squared. We can define the “p-value” to be 1 minus the cdf for the chi-squared distribution (which we also called the “survival fraction” in general), which in this case is 94%.

The p-value is the probability of obtaining a chi-squared equal or worse (bigger) than we did, if the model were correct. We have to make our decision on whether we have a good fit based on a statistical criterion. Typically, we could say that if the p-value is less than 5%, then the model is not likely to be a good fit. But again, we should take this with a grain of salt and generally make sure that we understand everything about the fit (such as the residuals and the covariance matrix) rather than making a conclusion based on a “hard” threshold on the p-value.

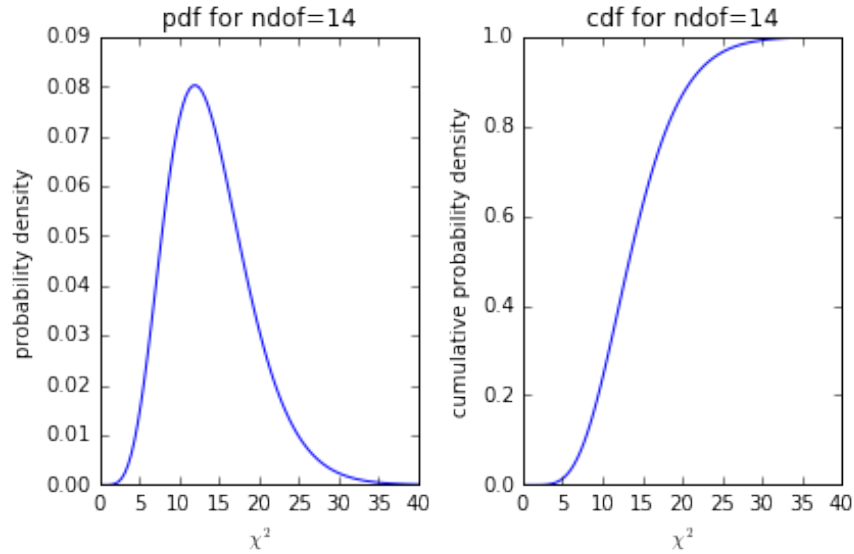


Figure 8.9: Probability density and cumulative density functions for the chi-squared distribution for 14 degrees of freedom (as in the case for the fit in Figure 8.7).

8.4 Summary

In this chapter, we used the principle of maximum likelihood to evaluate the parameters, $\vec{\beta}$, of a model, $y^{exp}(x, \vec{\beta})$, to fit a set of N pairs of data points, $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$. We found that this led to the requirement that we minimize the chi-squared, χ^2 , given by:

$$\chi^2 = \sum_{i=1}^{i=N} \frac{(y_i - y^{exp}(x_i, \vec{\beta}))^2}{\sigma_{y_i}^2} \quad (8.25)$$

which is a sum of the squared distance between the points and the model, divided by the error in the data points. For this reason, the method is also called the “Method of Least Squares”. We note that this **method is only valid when we expect the errors on the y points to be standard errors** (i.e. that the uncertainties on the y values correspond to the standard deviation of values that are normally distributed about the model prediction, y^{exp}).

In the case where the model is linear in the unknown parameters, one can minimize the chi-squared analytically (by taking the derivatives with respect to the parameters and setting them equal to zero) and obtain a set of coupled linear equations for the parameters. We showed this explicitly for the model:

$$y^{exp}(x, m, b) = mx + b \quad (8.26)$$

where we found that the least squares estimates of m and b , if all of the errors in the y_i are equal to σ_y , are given by:

$$\begin{aligned} m &= \frac{N \sum xy - \sum x \sum y}{N \sum x^2 - (\sum x)^2} \\ b &= \frac{\sum x^2 \sum y - \sum x \sum xy}{N \sum x^2 - (\sum x)^2} \end{aligned} \quad (8.27)$$

Using the formalism to propagate errors, we found that the errors in the parameters are given by:

$$\begin{aligned} \sigma_m &= \sigma_y \sqrt{\frac{N}{N \sum x^2 - (\sum x)^2}} \\ \sigma_b &= \sigma_y \sqrt{\frac{\sum x^2}{N \sum x^2 - (\sum x)^2}} \end{aligned} \quad (8.28)$$

We found that we could qualitatively discuss the goodness of a model by considering the residuals between the model estimates of the y_i and the measured values. If the residuals show no trend and are symmetrically distributed about 0, then the model is likely a good representation of the data.

We also argued that there may be correlations between fitted parameters, and that one must take this into account (through their measured covariance), if propagating the errors from the parameters.

We also considered the general case of models that are not linear in the parameters, and showed that one can use numerical techniques to minimize the chi-squared in these cases, and to calculate the covariance matrix.

We finished by discussing one method to evaluate the goodness of the fit of the model to the data. We saw that the value of the chi-squared itself is a good indication of the fit. In particular, one expects that the chi-squared should be equal to approximately the number of the degrees of freedom in the model, ν :

$$\nu \equiv N - n - 1 \quad (8.29)$$

where N is the number of data points and n is the number of parameters. We saw that if a model is correct, we will obtain a well-defined distribution of chi-squared values when the

model is fit to various data sets with statistical fluctuations. We introduced the p-value as the probability that if the model is correct, one would obtain an equal or worse value of chi-squared (the p-value is one minus the cdf of the chi-squared distribution). We cautiously recommended that in general, if the p-value is less than 5%, then the model is unlikely to be a good representation of the data.

9

Statistics - Probabilities and the Bayesian approach

So far, we have used probability density functions to describe the distribution of the outcomes that we expect in a given situation, and we were able to specify a probability for a given outcome. For example, we could assign that the probability of rolling the number 5 with a dice was $\frac{1}{6}$. But what do we really mean by “probability”? How do we determine that the probability is $\frac{1}{6}$? In this chapter, we take a closer look at the meaning of the word “probability”, how to define it, and how to interpret a probability.

9.1 Kolmogorov’s axioms

We begin by using Kolmogorov’s three axioms to define probabilities. Given a set, S (for example, a list of possible outcomes), and a subset A of the set, we define the probability $P(A)$ as a real number with the following properties:

1. $P(A) \geq 0$ is a non-negative real number for any element A
2. The sum of the probabilities of all mutually exclusive subsets in S is equal to 1, $P(S) = 1$
3. For any two mutually exclusive subsets, A and B , of S , the probability assigned to their union, $A \cup B$, is given by the sum of their individual probabilities: $P(A \cup B) = P(A) + P(B)$ (an element of the subset A must not be an element of the subset B for them to be mutually exclusive).

For example, if $S = \{1, 2, 3, 4, 5, 6\}$ is the set of possible outcomes of dice rolls, the probability of rolling a 5 is defined $P(5)$. The probability of obtaining a roll that is a 5 or a 6 is defined as $P(5 \cup 6) = P(5) + P(6)$. Finally, the sum of all the $P(i)$ is 1. Intuitively, we feel that neither of these outcomes is special compared to the others, so we can assign them all the same value, which by axiom 2, means that $P(i) = 1/6$. We can also define E as the subset of even numbers, $E = 2 \cup 4 \cup 6$, so that $P(E) = P(2) + P(4) + P(6) = \frac{1}{2}$.

We introduce the “complement of A ”, $!A$, to correspond to all elements in the set S that are mutually exclusive with A . We also introduce the “intersection of two subsets”, $A \cap B$, to correspond to the subset of elements that are elements of A and of B (shown in Figure

9.1). Note that the intersection of two mutually exclusive subsets is empty. From these definitions and using the axioms, one can derive additional properties of probabilities:

1. $P(!A) = 1 - P(A)$
2. $P(A \cup !A) = 1$
3. $0 \leq P(A) \leq 1$
4. $P(A \cup B) = P(A) + P(B) - P(A \cap B)$

The last property holds for the case where A and B are not necessarily mutually exclusive subsets.

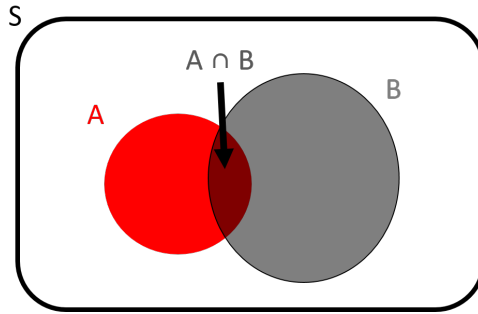


Figure 9.1: Illustration of the intersection of two subsets.

We can introduce the “conditional property”, $P(A|B)$, which is read as “the probability of A given B ”, which is given by:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \quad (9.1)$$

One way to think of the conditional probability, is to first think of taking an element from B , and then asking what is the probability that it is also a part of A . Only the fraction of the elements of B that are in the intersection of A and B satisfy the conditional probability.

A and B are said to be “independent” if:

$$P(A \cap B) = P(A)P(B) \quad (9.2)$$

If A and B are independent, then the conditional probability simplifies to:

$$\begin{aligned} P(A|B) &= \frac{P(A \cap B)}{P(B)} \\ &= \frac{P(A)P(B)}{P(B)} \\ &= P(A) \end{aligned} \quad (9.3)$$

That is, if A and B are independent, the probability of A given B is just the probability of A and does not depend on B .

Example 9-1: What is the probability of the sum of two dice being 7 given that one of the dice is a 5?

We need to calculate the conditional probability, $P(\text{sum} = 7 | \text{rolled a } 5)$.

$$P(\text{sum} = 7 | \text{rolled a } 5) = \frac{P(\text{sum} = 7 \cap \text{rolled a } 5)}{P(\text{rolled a } 5)} \quad (9.4)$$

The probability of the sum equal to 7 and rolling a 5, $P(\text{sum} = 7 \cap \text{rolled a } 5)$, is given by the probability of rolling a 2 and 5 together (the only way to get a sum of 7 if one of the dice is a 5). Out of 36 ways to roll a pair of dice, only two of them correspond to one dice being 2 and the other being 5:

$$P(\text{sum} = 7 \cap \text{rolled a } 5) = \frac{2}{36} \quad (9.5)$$

The probability of rolling a 5, when rolling two dice is given by counting the number of combinations that have a 5 in them. Out of the 36 possible outcomes, 11 of them contain at least one 5:

$$P(\text{rolled a } 5) = \frac{11}{36} \quad (9.6)$$

The conditional probability is thus given by:

$$P(\text{sum} = 7 | \text{rolled a } 5) = \frac{P(\text{sum} = 7 \cap \text{rolled a } 5)}{P(\text{rolled a } 5)} = \frac{\frac{2}{36}}{\frac{11}{36}} = \frac{2}{11} \quad (9.7)$$

We can verify the result by tabulating all of the outcomes that contain a 5, these are:

$\{15, 25, 35, 45, 55, 65, 51, 52, 53, 54, 56\}$

there are 11 such outcomes, two of which sum to seven.

Example 9-2: If we rolled two dice and the sum was 7, what is the probability that one of the dice is a 5?

This is the inverse of the conditional probability that we had in the previous example. We now need to calculate the conditional probability, $P(\text{rolled a } 5 | \text{sum} = 7)$.

$$P(\text{rolled a } 5 | \text{sum} = 7) = \frac{P(\text{rolled a } 5 \cap \text{sum} = 7)}{P(\text{sum} = 7)} \quad (9.8)$$

The probability of the sum equal to 7 and rolling a 5, $P(\text{rolled a } 5 \cap \text{sum} = 7)$, is the

same as in Example 9-1:

$$P(\text{rolled a } 5 \cap \text{sum} = 7) = \frac{2}{36} \quad (9.9)$$

The probability of rolling two dice and having a sum equal to 7 is given by

$$P(\text{sum} = 7) = \frac{6}{36} \quad (9.10)$$

as there are 6 combinations of 2 dice that give a sum equal to 7. The conditional probability is thus given by:

$$P(\text{rolled a } 5 | \text{sum} = 7) = \frac{P(\text{rolled a } 5 \cap \text{sum} = 7)}{P(\text{sum} = 7)} = \frac{2}{36} \frac{36}{6} = \frac{1}{3} \quad (9.11)$$

We can verify the result by tabulating all of the outcomes that sum to 7, these are:

$\{16, 25, 34, 43, 52, 61\}$

there are 6 such outcomes, two of which contain the number 5.

9.2 Bayes' Theorem

In general, the conditional probability $P(A|B)$ is not equal to $P(B|A)$, however, the two are related. The relationship is easily found using equation 9.1 to write the probability of the intersection, $P(A \cap B)$ in terms of the conditional probability:

$$\begin{aligned} P(A|B) &= \frac{P(A \cap B)}{P(B)} \\ \therefore P(A \cap B) &= P(A|B)P(B) \end{aligned} \quad (9.12)$$

and since $P(A \cap B) = P(B \cap A)$, we can write:

$$P(A|B)P(B) = P(B|A)P(A) \quad (9.13)$$

This is often re-arranged in the form:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (9.14)$$

and is called “Bayes’ Theorem”. Bayes’ theorem not only gives us a way to invert the conditional probability, but as we will see, it gives us an entirely new way to interpret probabilities. The ability to invert the conditional probability also highlights some properties of probabilities that are counter-intuitive.

Example 9-3: Suppose that there is a rare disease, that only 1% of the population carries, and a test that is 99% accurate when someone has the disease (for someone with the disease, the test will be positive 99% of the time). Furthermore, the test has a false positive rate of 2% (for someone without the disease, there is a 2% chance that the

test will be positive). If someone tests positive for the disease, what is the probability of actually having the disease?

We need to calculate the probability of having the disease given that we have tested positive. Bayes' theorem tells us:

$$P(\text{have disease}|\text{tested positive}) = \frac{P(\text{tested positive}|\text{have disease})P(\text{have disease})}{P(\text{tested positive})}$$

We know the following:

- $P(\text{tested positive}|\text{have disease}) = 0.99$
- $P(\text{have disease}) = 0.01$

The quantity $P(\text{tested positive})$ is not given to us directly, and corresponds to the probability of having a positive test, regardless of having the disease. It is thus the sum of the probability of testing positive if we have the disease, and the probability of testing positive if we don't have the disease. It is given by:

$$\begin{aligned} P(\text{tested positive}) &= P(\text{have disease}) \times P(\text{tested positive}|\text{have disease}) \\ &\quad + P(\text{not have disease}) \times P(\text{tested positive}|\text{not have disease}) \\ &= 0.01 \times 0.99 + 0.99 \times 0.02 \\ &= 0.0297 \end{aligned}$$

We now have all of the quantities that we need for Bayes' theorem to evaluate the probability of having the disease, given that we have tested positive:

$$\begin{aligned} P(\text{have disease}|\text{tested positive}) &= \frac{0.99 \times 0.01}{0.0297} \\ &= 0.33 \end{aligned}$$

That is, there is only a 33% chance of having the disease even if we tested positive. This seems counter-intuitive, as the test sounds like it should be pretty accurate as it only has a 2% rate of false positive, yet, testing positive only gives us a 33% chance of actually having the disease! The reason the test is not actually that good is because the probability of anyone having the disease is 1% which is very small. For the test to be accurate, it must have a false positive rate that is well below the probability of a random person having the disease.

Example 9-3 illustrates a common mistake that can be made when trying to invert a conditional probability incorrectly. In the case above, one would be mistaken to assume that a large value of $P(\text{tested positive}|\text{have disease})$ should result in a large value of $P(\text{have disease}|\text{tested positive})$.

In the legal system, this is often called the “prosecutor’s fallacy”, and has led to people being

wrongly convicted of crimes as the result of a DNA test. For example, a DNA test may have a very high probability, $P(DNA\ match|guilty)$, of being a match for someone who is guilty. The prosecutor's fallacy would be to attribute a high probability of being guilty to someone with a DNA match, $P(guilty|DNA\ match)$, without considering, $P(guilty)$, the probability that the person is guilty regardless of the DNA test. From Bayes' theorem, we have:

$$P(guilty|DNA\ match) = \frac{P(DNA\ match|guilty)P(guilty)}{P(DNA\ match)}$$

If the individual is a random person from the population, the probability that they committed the crime and are guilty is $P(guilty) = \frac{1}{N}$, where N is the number of individuals in the population that could commit the crime, and is thus typically very small.

Generally, a DNA test should only be used when $P(guilty)$, the probability that the individual is guilty, is already "high" based on other factors, for example, if the individual was seen near the scene of the crime, etc. As we will see in the next section, $P(guilty)$ is called the "prior probability", and corresponds to the probability that we can assign before integrating new information (such as a DNA test).

It is now hopefully understood in the legal system, that trying to get a DNA match on a cold hit from a database is not a scientifically valid approach. Although the probability to have a DNA match is low, if the database is large, the probability that someone in the database will match is actually high. For example, if the probability of a random DNA match is 1 in 10,000, and the database contains 10,000 individuals, the probability that at least one person will match is 63% (from the binomial probability). Since that person is a random individual, it is certainly not 63% probable that they committed the crime, even if the DNA test only has a 0.01% chance of giving a false positive. One has to remember that the probability that a random person committed the crime is also very small.

A famous case is that of Sally Clark, a British woman accused of killing her two infants. The probability of having a Sudden Infant Death Syndrome (SIDS) in one child is small, and if it is purely uncorrelated, the probability of having two SIDS death in the same family is even much smaller. This was used as the basis for convicting her. However, it was not considered that there could be genetic factors that could highly increase the probability of two SIDS in one family. Furthermore, one should not consider the probability of having two SIDS in one family as the only meaningful probability. This probability needs to be compared to the probability of having two murders of infants in one family, which is also very low, potentially even lower than having two SIDS. If the conclusion that she is guilty is based purely on probabilities, then it is not scientifically valid. One would need further supporting evidence to suspect that she murdered her infants, especially if the probability of a random person murdering her two infants is smaller than the probability of two infants dying from SIDS in the same family.

9.3 The Bayesian interpretation of probability

The interpretation of probabilities is a subtle issue that can cause much confusion and un-necessary debates (e.g. frequentists versus Bayesians).

The frequentist approach (also called the "classical definition") defines probability as the frequency with which you expect a certain outcome if you perform the experiment many times. That is, if you roll the dice 600 times and you got the number five 100 times, then the probability of obtaining a five is 100/600. The frequentist definition implicitly suggests that the measurement can be performed many times and that the probability is precisely a statement on the outcome of many experiments.

When we introduced the binomial probability:

$$P^{binom}(k, N, p) = \frac{N!}{k!(N-k)!} p^k (1-p)^{N-k} \quad (9.15)$$

we specifically defined it to be "the probability to obtain k successes in N trials when we expect an average of Np successes". That is, we did not actually impose an interpretation on the meaning of p . We can in fact use this definition of the binomial probability to define p as the probability of getting an individual success. For the case of rolling a dice, the probability of obtaining the number five would be defined as the average number of times that we rolled the number five (Np) divided by the total number of times that we rolled the dice (N). This would be a frequentist interpretation and definition of p , the probability to roll the number five.

But what if we ask about the probability of it raining tomorrow? The frequentist approach cannot be applied to define that probability, as there is only one tomorrow. We do not have a well-prescribed method to determine the frequency of how many times "it rains tomorrow". At best, we could look at how many days it rains on average and say that the probability of raining on any given day is on average a certain number. But that does not really answer the question about it specifically raining tomorrow.

The Bayesian definition of probability (also called the "subjective definition") is given as a "degree of belief for something to happen". Specifically, one defines a hypothesis space, S , and the Bayesian probability, $P(A)$, is a statement in the confidence of a particular hypothesis, A , to be true. If all hypotheses are mutually exclusive and cover all possibilities, then their set will satisfy the axioms for defining probabilities.

For example, "I am 85% sure that it will rain tomorrow" is a Bayesian statement on the probability of the hypothesis that it will rain tomorrow. That probability can be estimated from prior information (such as the weather patterns in the last few days), but it will remain subjective. Some would argue that the Bayesian approach includes the frequentist approach, as a frequentist statement such as "performing an experiment N times will yield k successes" can be regarded as a Bayesian hypothesis.

The Bayesian interpretation of probabilities is based on Bayes' theorem, which we can write as:

$$P(\text{hypothesis}|\text{data}) = \frac{P(\text{data}|\text{hypothesis})P(\text{hypothesis})}{P(\text{data})}$$

This gives us a tool to understand the probability of a hypothesis, given some (new) data and our “prior” belief in the hypothesis, $P(\text{hypothesis})$. It is a unique approach that allows us to formalize how we include our prior beliefs and new data into a degree of belief in a hypothesis.

The Bayesian interpretation is particularly well-suited for analysing data from a physics experiment, where we want to test a hypothesis using data that we have collected. In the context of a physics experiment, we would have:

- $P(\text{hypothesis}|\text{data})$ the probability that given the data the hypothesis is correct, or rather, our new degree of belief in the hypothesis after analysing the data. This is called the “posterior probability”.
- $P(\text{data}|\text{hypothesis})$ the probability that given our hypothesis we would get the measured data, also called the “likelihood” of the data.
- $P(\text{hypothesis})$ our previous degree of belief in the hypothesis, before collecting the new data, also called “the prior (probability)”.
- $P(\text{data})$ is the probability of obtaining the data, regardless of our hypothesis; this is generally not possible to calculate.

Since the probability of the data, $P(\text{data})$, is not usually possible to calculate, it is often omitted, and we write:

$$P(\text{hypothesis}|\text{data}) \propto P(\text{data}|\text{hypothesis})P(\text{hypothesis})$$

$P(\text{data})$ only really serves to normalize the probability of the hypothesis given the data. In a typical application, we would compare the ratios of the probabilities of different hypotheses, so the $P(\text{data})$ would cancel out. For example, we may calculate the ratio of the probability of our hypothesis in a new theory of gravity over the probability in the (null) hypothesis that Newton's (or Einstein's) theory of gravity is correct.

It is also worth noting that the likelihood, $P(\text{data}|\text{hypothesis})$ is the same likelihood that we used in previous chapters, when we calculated the probability of getting a certain set of measurements given a model.

The “prior probability”, $P(\text{hypothesis})$, is our degree of belief in the hypothesis before collecting the data. This can be a subjective term to define and may be a number that we subjectively “feel” is correct, or it could be based on the results of previous experiments.

The Bayesian interpretation is that we start with a prior probability (a degree of belief about the hypothesis), we then perform an experiment which modifies our prior probability

to gives us the posterior probability (i.e. our new degree of belief in the hypothesis). It formalizes how our belief should change, based on the result of an experiment. It is critical to note that the posterior probability (how much we now believe the hypothesis) depends on how much we previously believed the hypothesis.

Consider the following example to determine our degree of belief that aliens probed your brain while you were sleeping. If aliens probe your head while you are asleep, then you will wake up with a headache; there is no question about that! The data is “you having a headache in the morning”, and the hypothesis is that “aliens probed your head”. If we define $P(data|hypothesis)$ as the probability that you have a headache in the morning if aliens probed your head, then we could assign it a very high number, say 1, as it would be very likely that you would have a headache if aliens probed your head.

The question is whether you waking up with a headache means that aliens probed your head, or in other words whether you should now believe that $P(hypothesis|data)$ is now also big. The answer is clearly no! The hypothesis that aliens probed your head had a very small probability before you woke up with a headache (a small prior probability, $P(hypothesis)$). There are many reasons why you would have a headache, so even though the data are consistent with the hypothesis, that does not make the hypothesis significantly more likely!

Surely you can find less ridiculous examples in real life where someone has tried to convince you that some data makes some hypothesis sound plausible. That is because they forgot that the prior probability for that theory was infinitesimal to begin with! The less likely a hypothesis is, the more significant the data must be to convince one of the hypothesis (and you can never “prove” a hypothesis). Assigning a numerical value to the prior probabilities is the tricky part. You can however agree that Bayes’ theorem gives a useful way to think about probabilities and statistics, and this is why it is often adopted as a modern framework for analysis.

9.3.1 The probability that a coin is fair

We can use a Bayesian analysis to estimate, based on a result, whether a coin is fair. Let us suppose that we flipped a coin 10 times and obtained heads 7 times, and we wish to use this result to determine the probability, p , that the coin will land on heads. We need to formulate this problem into a Bayesian hypothesis. Since p can take a infinite number of values between 0 and 1, we could formulate the hypothesis that p lies between 0.49 and 0.51 to be equivalent to the hypothesis that the coin is fair. Very generally, we can form a continuous number of hypotheses corresponding to p being in different ranges. Let us call $P(p|result)$ the probability density function for p , so that $P(p|result)dp$ is the posterior probability that p lies between p and $p + dp$.

From Bayes’ theorem, we have:

$$P(p|result) = \frac{P(result|p)P(p)}{P(result)}$$

$P(result|p)$ is simply the binomial probability of obtaining 7 heads out of 10 tosses for a

given value of p . $P(result)$ is the probability of obtaining 7 out of 10 results, regardless of the value of p . Since it does not depend on p , we can ignore it, but it is given by:

$$P(result) = \int_0^1 P(result|p)P(p)dp$$

Finally, $P(p)$, is our Bayesian prior probability for our value of p . If we have no prior knowledge of whether the coin is fair, we can just make this equal to a constant, independent of p . In the case of no prior knowledge (a so-called “flat prior”):

$$P(p|result) \propto P(result|p) = P^{binomial}(k = 7, N = 10, p)$$

That is, we can treat the binomial probability as the probability density for p , given a fixed value of k and N . Using this flat prior, the posterior probability for p can be calculated with the following simple python code, and is plotted in Figure 9.2.

Python Code 9.1: Posterior probability for a coin toss

```
import scipy.stats as stats
import numpy as np
import pylab as pl

k=7
N=10
#An array of many values of p
p = np.linspace(0,1,100000)
#The corresponding likelihood for different values
pbinom = stats.binom.pmf(k,N,p)
prior = stats.uniform.pdf(p)
#multiply by the prior (not very useful here!)
posterior = pbinom*prior
#normalize the posterior and prior when plotting
pl.plot(p, posterior/posterior.sum(), color='black', label='posterior')
pl.plot(p, prior/prior.sum(), '--', color='black', label='prior')
pl.xlabel('p')
pl.ylabel('pdf for p to get k={:.0f} and N={:.0f}'.format(k,N))
pl.legend(loc='best')
pl.title("Prior with no knowledge about p")
pl.show()
```

9.1: Output

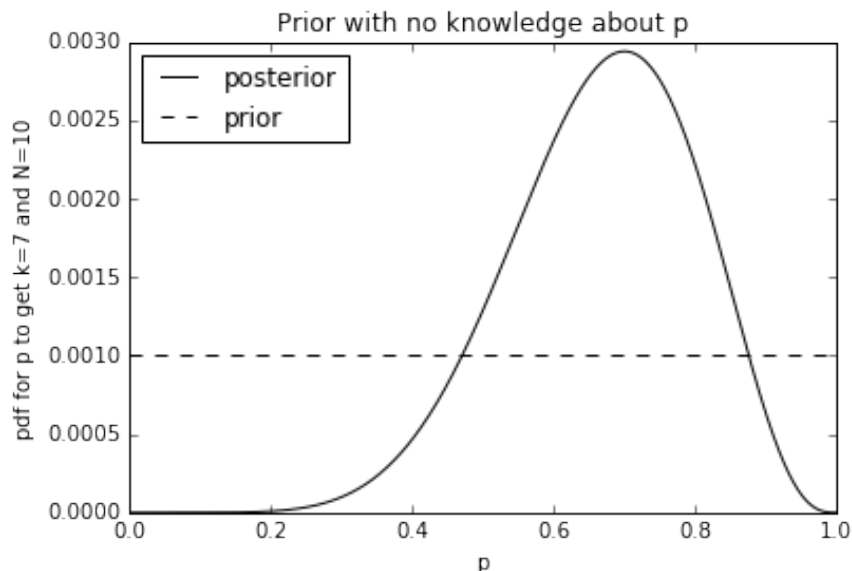


Figure 9.2: Posterior probability for the probability that a coin will land on heads, based on having had the result of 7 heads in 10 coin tosses and a flat prior.

The posterior probability plotted in Figure 9.2 tells us the probability that p is within a certain range based on our measurements of k and N . In Chapters 3 and 6 we introduced the binomial error that one would obtain for the measurement of the fraction (or proportion), $p = \frac{k}{N}$ as:

$$\sigma p = \frac{\sigma_k}{N} = \sqrt{\frac{p(1-p)}{N}}$$

Using the data in Figure 9.2, we can determine which fraction of the posterior probability lies in the range $p \pm \sigma_p$ using a simple python code:

Python Code 9.2: Coverage of the binomial error on a ratio

```
p_mean = k/N
sigma_p = np.sqrt(p_mean*(1.0-p_mean)/N)
#Calculate the probability that p is in the range p+/-sigma_p by summing all
#of the probabilities in the range
#(and then dividing by the normalization given by the sum in the range of p=0
#to p=1):
#Find the indices of the array p that correspond to the min and max value of
#p:
pmin = p_mean-sigma_p
pmax = p_mean+sigma_p
ipmin = (np.abs(p-pmin)).argmin()
ipmax = (np.abs(p-pmax)).argmin()
#the probability
prob = pbinom[ipmin:ipmax].sum()/pbinom.sum()
print("The probability that p is between {:.2f} and {:.2f} is {:.1f}%".format
      (p[ipmin], p[ipmax], 100*prob))
```

9.2: Output

The probability that p is between 0.56 and 0.84 is 72.3%.

which is close to the canonical 68% that one would obtain if the posterior for p were normally distributed. It is easy to verify that as k and N increase (while maintaining their ratio constant), the posterior distribution for p (under the assumption of a flat prior) converges to a normal distribution with standard deviation equal to σ_p . We have already observed that the binomial distribution for k approaches the normal distribution when the mean $\bar{k} = Np$ is large, so the same condition will lead to a normal distribution for p , as long as the Bayesian prior is flat. The Bayesian approach thus gives us a more general description of our measurement of p , and we see that the binomial error gives a good approximation in the case when the posterior for p becomes gaussian.

We can examine the effect of different prior probabilities. For example, if we believe that the coin is fair (perhaps it was provided by your trusted physics professor), we may have a prior probability that is a gaussian centered at $\mu_p = 0.5$ with a standard deviation of, say, $\sigma_p = 0.1$.

On the other hand, we may believe that the coin is biased (perhaps it was provided by a con artist), so that the prior probability is peaked at values of $p = 0$ and $p = 1$. The python code below shows how the posterior probability is affected by these different priors, which are plotted in Figure 9.3.

Python Code 9.3: Posterior probability for a coin toss with different priors

```
#Let's start with an assumption that the coin is likely fair, with p normally
    distributed about 0.5 with std=0.1
prior_fair = stats.norm.pdf(p,0.5,0.1)
posterior_fair = pbinom*prior_fair

pl.figure(figsize=(10,4))
pl.subplot(121)

#We normalize the prior and posterior to have the same area
pl.plot(p,posterior_fair/posterior_fair.sum(),color='black',label='posterior'
)
pl.plot(p,prior_fair/prior_fair.sum(),color='black',label='prior')
pl.xlabel('p')
pl.ylabel('pdf for p to get k={:.0f} and N={:.0f}'.format(k,N))
pl.legend(loc='best')
pl.title("Prior that the coin is fair")

#Let's compare with the assumption that the coin is likely unfair (p=0 or 1
    are more likely)
prior_unfair = (p-0.5)**6
posterior_unfair = pbinom*prior_unfair

#We normalize the prior and posterior to have the same area
pl.subplot(122)
pl.plot(p,posterior_unfair/posterior_unfair.sum(),color='black',label='
    posterior')
pl.plot(p,prior_unfair/prior_unfair.sum(),color='black',label='prior')
pl.xlabel('p')
pl.ylabel('pdf for p to get k={:.0f} and N={:.0f}'.format(k,N))
```



```

pl.legend(loc='upper center')
pl.title("Prior that the coin is biased")

pl.tight_layout()
pl.show()

```

9.3: Output

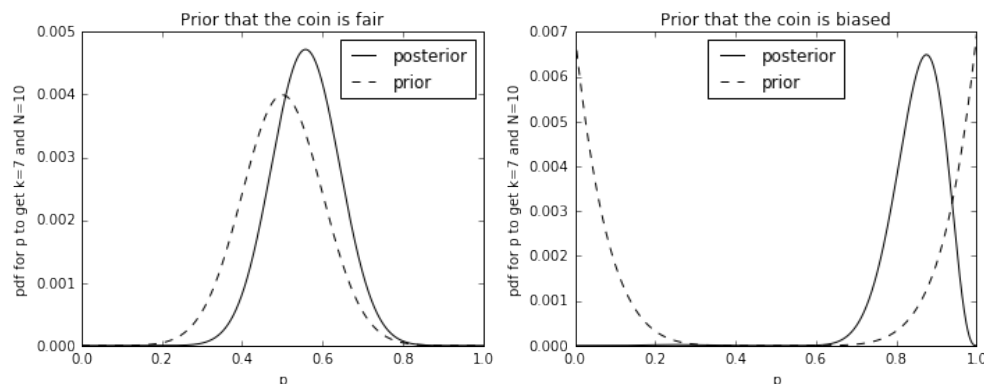


Figure 9.3: Posterior probabilities for the probability that a coin will land on heads, based on having had the result of 7 heads in 10 coin tosses and a prior that the coin is likely fair or likely biased.

In these cases, we see that our choice of prior has an effect on the posterior probability and our interpretation of the data. For the prior that the coin is fair, the posterior is slightly shifted to higher values of p , but since the data are not that significant, the posterior probability is still consistent with a value $p = 0.5$. For the case of the prior that the coin is unfair, the data has resulted in us completely rejecting the possibility that p is close to zero, and also results in a low posterior probability that the coin is fair ($p = 0.5$). The data combined with our prior probability thus support the conclusion that the coin is unfair. As you can see, in a Bayesian analysis, the effect of the prior probability can be quite large, which is both a good and a bad thing. It is good because it gives us a framework to incorporate previous knowledge, and bad because defining the prior can be subjective.

We can use this posterior probability for p to give our degree of belief that p lies within a certain range. Note that the posterior probability is not normalized, so we must normalize it by dividing by the sum of all possible outcomes. The python code below gives the probability that p is between 0.45 and 0.55 for the 3 different priors:

Python Code 9.4: Defining a confidence range from the posterior probability

```

#Define a range for p
pmin = 0.45
pmax = 0.55
#Find the indices of the array p that correspond to those values
ipmin = (np.abs(p-pmin)).argmin()
ipmax = (np.abs(p-pmax)).argmin()
#To get the probability, we sum the posterior for all values in the range
#and then divide by the sum of all elements to ensure that the probability is

```

```

normalized
prob = posterior[ipmin:ipmax].sum()/posterior.sum()
print("With flat prior, the probability that p is between {:.2f} and {:.2f}
      is {:.2f}%".format(p[ipmin],p[ipmax],100*prob))
prob = posterior_fair[ipmin:ipmax].sum()/posterior_fair.sum()
print("With fair prior, the probability that p is between {:.2f} and {:.2f}
      is {:.2f}%".format(p[ipmin],p[ipmax],100*prob))
prob = posterior_unfair[ipmin:ipmax].sum()/posterior_unfair.sum()
print("With unfair prior, the probability that p is between {:.2f} and {:.2f}
      is {:.7f}%".format(p[ipmin],p[ipmax],100*prob))

```

9.4: Output

With flat prior, the probability that p is between 0.45 and 0.55 is 13.01%
 With fair prior, the probability that p is between 0.45 and 0.55 is 36.28%
 With unfair prior, the probability that p is between 0.45 and 0.55 is
 0.000068%

In Figure 9.4, we show the posterior probabilities for p after having obtained 70 heads in 100 coin tosses. We show the result for the same three priors as above, namely a flat prior, a prior that the coin is fair, and a prior that the coin is biased. In this case, all three posterior distributions are similar, and we see that the choice of prior had little effect on our analysis. Regardless of our prior, we would conclude that it is unlikely that $p = 0.5$. This is because the result of obtaining 70 heads in 100 coin tosses is much more statistically significant than that of obtaining 7 heads in 10 coin tosses. In this case, the data carried much more weight than the prior, which is intuitively satisfying. Thus if we have significant data, the choice of prior matters less, and we can be less concerned about the potentially subjective nature of the prior.

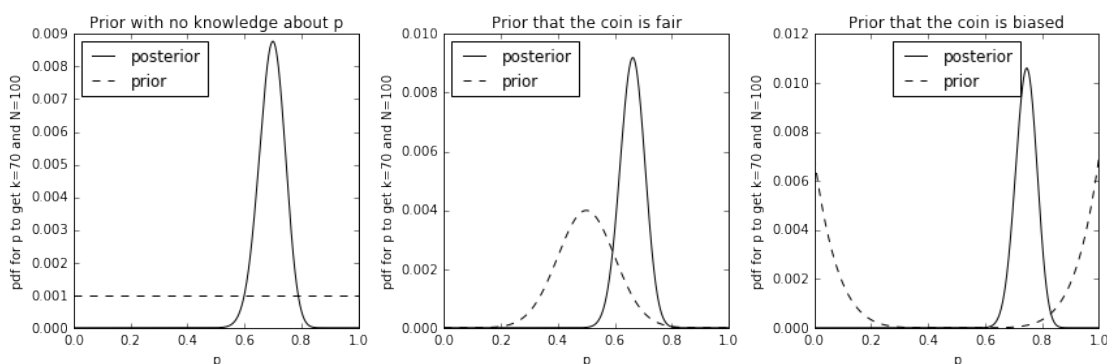


Figure 9.4: Posterior probabilities that a coin will land on heads, based on having had the result of 70 heads in 100 coin tosses and a flat prior, as well as priors that the coin is likely fair or likely biased.

We can again compare the different probabilities that the value of p is between 0.45 and 0.55 for the case of obtaining 70 heads in 100 coin tosses. In all cases, the probability is small and we see that the choice of prior has little impact on our qualitative conclusion that the coin is likely unfair. The output of the code to calculate the probability of p between 0.45 and 0.55 is shown below:

With flat prior, the probability that p is between 0.45 and 0.55 is 0.12%

With fair prior, the probability that p is between 0.45 and 0.55 is 0.60%

With unfair prior, the probability that p is between 0.45 and 0.55 is 0.0000058%

9.3.2 Fitting a straight line the Bayesian way

A complete development of Bayesian data analysis is beyond the scope of this book, although it is worth illustrating the process. Consider the case where we have N pairs of data points, (x_i, y_i) , and we wish to determine the best values for the parameters, a and b , of a linear model that describes the data:

$$y^{model}(x) = a + bx$$

The Bayesian approach is to determine the posterior probabilities for the parameters a and b , given the data and our prior probabilities for the two parameters. Let us assume that we have no prior knowledge of a , and so we assign a , a “flat” prior probability that is constant for all values of a . For the slope, b , let us assume that it has been measured to be $\mu_b \pm \sigma_b$ by a previous experiment. We assume that the previous experiment quotes a standard error, so that we can express our prior probability of b as being normally distributed about a mean of μ_b with a standard deviation of σ_b : $P^{prior}(b) = P^{norm}(b, \mu_b, \sigma_b)$.

We now regard the posterior probability as the probability that a given choice of parameters is correct given the data (the hypothesis is thus that a given choice of parameters is correct). We will then use this posterior probability to guide us in choosing the best estimates for a and b . We write the posterior probability as $P(a, b|data)$, and we will calculate this for many values of a and b and compare the different probabilities that we get. As you can see, we do not need the normalizing term, $P(data)$, since it does not depend on a particular hypothesis (i.e. a particular choice of a and b) and we will only care about the relative probabilities for different choices of the parameters.

The posterior probability is proportional to the likelihood multiplied by the prior probabilities:

$$P(a, b|data) \propto P(data|a, b)P(a, b)$$

The likelihood, $P(data|a, b)$, is the probability of obtaining a given data set given a particular choice of a and b . If we assume that the data points have standard errors only in y , then we know that the likelihood is related to the chi-squared that we can calculate between the data and the model for a particular choice of a and b (recall equation 8.7):

$$P(data|a, b) = e^{-\frac{\chi^2}{2}}$$

where χ^2 depends on a and b as well as the data. The prior probability is given by:

$$\begin{aligned} P(a, b) &= P(a)P(b) \\ &\propto e^{-\frac{(b-\mu_b)^2}{2\sigma_b^2}} \end{aligned} \tag{9.16}$$

where, again, we do not care about the parts of the normalization that are constant and do not depend explicitly on a or b . The posterior probability for given choice of a and b is thus given by:

$$P(a, b|data) \propto e^{-\frac{\chi^2}{2}} e^{-\frac{(b-\mu_b)^2}{2\sigma_b^2}} \quad (9.17)$$

where you might notice that the (gaussian) prior probability on b is very similar to adding an extra “penalty term” to the chi-squared (treating b as a measured value with expected values μ_b and error σ_b). The penalty term will give a low value to the posterior probability if b is far from μ_b (far in terms of the size of σ_b).

There is a clever algorithm called the Markov Chain Monte Carlo (MCMC) that, given a function for the posterior probability, can draw random values of the parameters a and b such that those parameters are distributed according to the posterior probability. That is, the clever algorithm knows how to numerically evaluate the posterior probability. It is beyond the scope of this book to go into the details of the algorithm, but it is worth seeing its results.

Figure 9.5 shows a simulated data set with a true offset, $a = 3$, and a true slope $b = 8$.

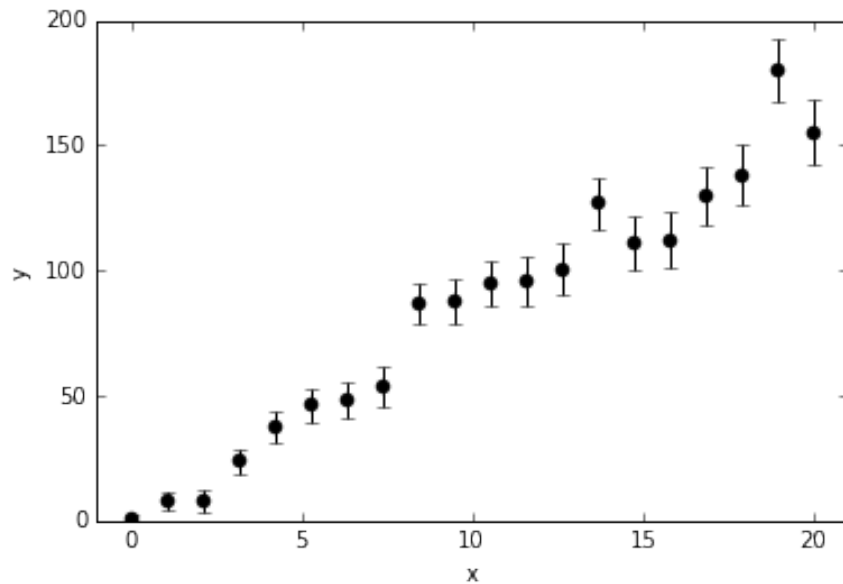


Figure 9.5: Simulated data generated with an offset of 3 and a slope of 8.

Figure 9.6 shows a projection of the posterior probability $P(a, b|data)$ onto a 2-dimensional plane corresponding to a and b . This was done for the case where the prior probability for b was chosen with $\mu_b = 10$ and $\sigma_b = 1$. The mean value of b from a previous measurement is thus not consistent with the “true” value of $b = 8$ that we chose to simulate the data.

We can see that the posterior probability is maximal in the regions of $b = 8.5$ and $a = -1$. The posterior is similar to the chi-squared as a function of a and b , except that it also contains

the prior probability for b . The MCMC algorithm actually allows us to draw a histogram of data that are proportional to the posterior probability, which we could normalize to obtain an actual probability.

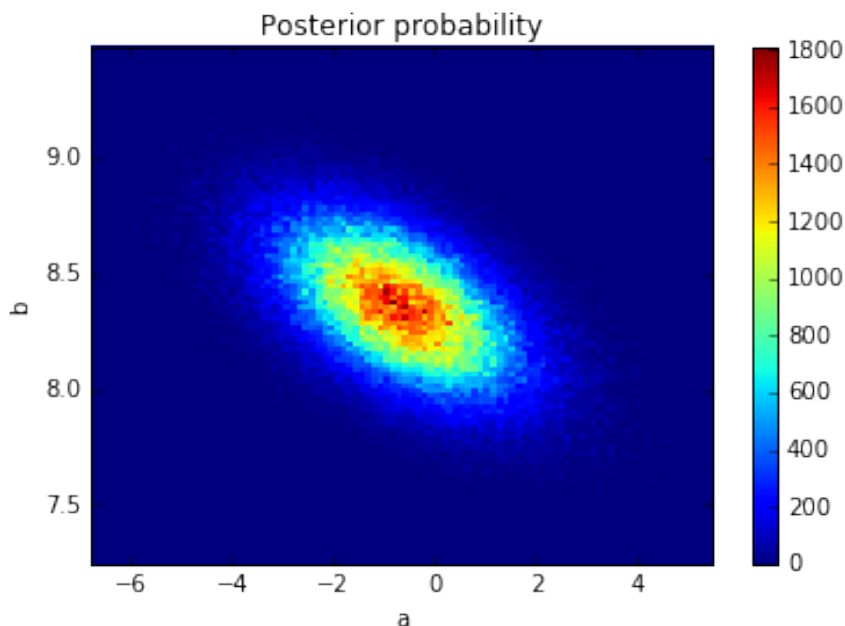


Figure 9.6: 2-dimensional posterior distribution, for the case with $\mu_b = 10$ and $\sigma_b = 1$.

In order to estimate our best values for a and b , we can project the posterior probability into a 1-dimensional form for a and b . This leads to 1-dimensional histograms that are proportional to the prior probability, which we can normalize, as shown in Figure 9.7. From this, we can see that the posterior probabilities for both parameters are reasonably close to gaussian. It is thus reasonable to define the best fit values of the parameters as the mean and standard deviation of the gaussian posterior distribution, although one is free to define the best estimate as they please. The true power of the Bayesian method is that one can visualize the actual posterior probability for the parameters.

If we use the mean and standard deviations to define our best estimate of the parameters, we obtain $a = -0.71 \pm 1.41$ and $b = 8.36 \pm 0.24$. As we have seen previously, the uncertainty in the offset, a , is quite large, and the offset is usually difficult to fit precisely. Since the prior for b pulled b away from 8 (towards 10), the offset a was pulled down from the true value of 3, since the offset and slope in a linear fit are anti-correlated. We find that the value of the slope is much closer to the true value, 8, that we used to generate the data than it is to its prior probability value of 10. This is because the uncertainty in the prior value, was rather large, $\sigma_b = 1$, and had a small influence on our “updated” value of b . This illustrates how we used the data to “update” our knowledge of the parameter b that we had from a prior experiment.

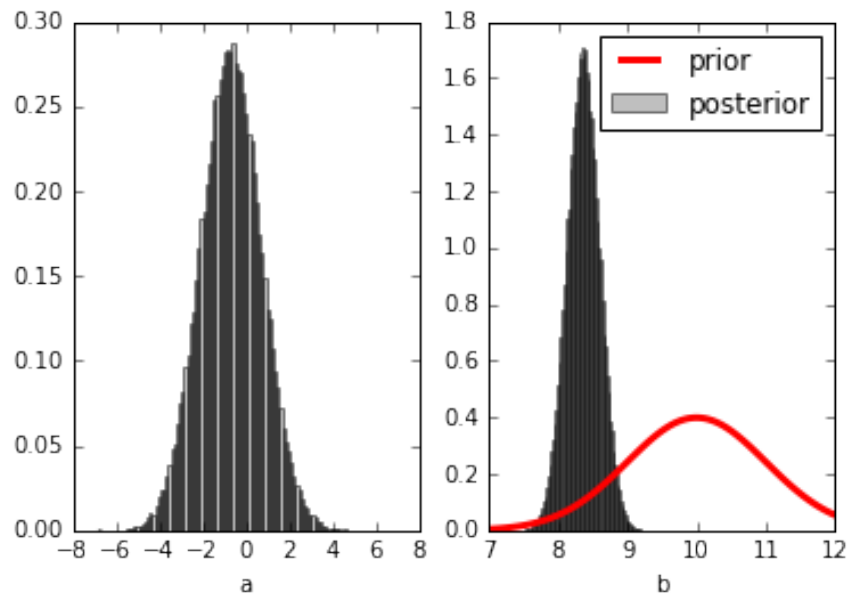


Figure 9.7: 1-dimensional posterior distributions for a and b , for the case with $\mu_b = 10$ and $\sigma_b = 1$. The prior probability for b is also shown.

Finally, Figure 9.8 shows the 1-d posterior probabilities for the case where $\mu_b = 10$ and $\sigma_b = 0.1$; that is, when the prior knowledge of b is given a much smaller uncertainty. In this case, we see that posterior for b has a mean that is much closer to 10, and that the distribution is quite narrow, because the prior is given a lot of importance (a small σ_b). If we use the mean and standard deviations to define our best estimate of the parameters, we obtain $a = -5.141 \pm 1.22$ and $b = 9.75 \pm 0.09$. We also see that the mean value of the parameter a has shifted down; this is to compensate for the higher value of b , since the slope and offset in a linear fit are usually anti-correlated. This again illustrates how the Bayesian approach allows us to combine prior information with data to update our knowledge. In this case, our prior knowledge was very “significant” and the data only had a small effect.

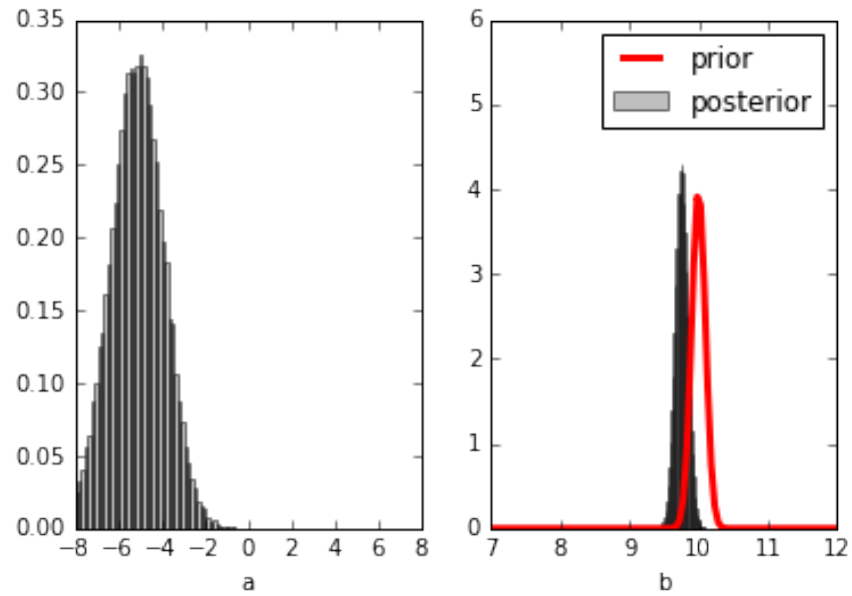


Figure 9.8: 1-dimensional posterior distributions for a and b , for the case with $\mu_b = 10$ and $\sigma_b = 0.1$. The prior probability for b is also shown.

10

Statistics - The Monte Carlo Method

The Monte Carlo (MC) method was developed to approximate difficult calculations by using random numbers to try and sample the solution. Originally, it was developed at the Los Alamos National Laboratory to simulate the propagation of neutrons in the design of nuclear weapons without having to solve the complex transport equations for neutrons. It has now gained widespread use in a variety of fields beyond physics, including economics, biology, and others. The Monte Carlo method is named after the Monte Carlo casino in Monaco (where James Bond is a regular), because the use of random numbers can remind one of using dice (or other instruments of chance) to perform calculations.

The basic idea behind the Monte Carlo method is to randomly sample the solution space in a clever way that does not require us to calculate the actual distribution of the solutions. For example, you may be interested in the distribution of profits that your poutine restaurant will make based on the number of customers that you get. Since every day you will get a random number of customers, ordering a random number of items, it is very difficult to calculate the distribution of the expected returns per day. With a Monte Carlo simulation, you would generate a random number of customers each day, generate random items to buy, calculate the profit, and then repeat many times. If you know the average distribution of the number of customers and their purchases, your simulation will yield the correct distribution for your expected returns.

10.1 Random numbers from a computer

Monte Carlo simulations require computers to generate random numbers. However, it is not possible to write a computer program that will generate truly random numbers, since computers will always execute a program in a predictable way. For that reason, the random numbers generated by a computer are called “pseudo random numbers”. Generally, the pseudo random number generating (PRNG) algorithms are based on taking a “seed” number and then generating a sequence of numbers that is seemingly random (although completely predictable if the seed number and the algorithm are known).

The sequence will ultimately repeat itself and has a “period” that depends both on the

seed and the algorithm. A good PRNG will have a large period and will generate random numbers that are “uniform”. By convention, most random number generators return a floating point value between 0 and 1, that is uniformly distributed between 0 and 1 (that is, no number is more likely than any other). Finally, a good PRNG should produce numbers that are “uncorrelated”, so that without knowing the algorithm, one would not be able to predict the next number in the sequence. For example, a PRNG could have a large period, produce numbers that are uniformly distributed, but alternately produce large and small numbers, so that the sequence is correlated and predictable. If one is relying on a PRNG to produce numbers for a Monte Carlo simulation, and the PRNG is not reliable, the results of the calculation will be incorrect.

Apart from their use in Monte Carlo simulations, PRNGs are used widely in encryption, where very large random numbers are needed to produce encryption keys that are difficult to break. They are also used in a variety of other applications, including electronic lottery machines.

A very simple class of random number generators are the “Linear Congruential Generators” (LCG). They are given by the simple recursive relation:

$$N_i = (aN_{i-1} + b) \bmod M$$

where N_0 is the seed of the sequence, and “mod” is the modulo operation ¹. The integer constants a , b , M , determine the properties of the random numbers. a is called the “multiplier”, b the “increment”, and M the “modulus”. In order to obtain numbers, u_i , between 0 and 1, one divides N_i , by M :

$$u_i = \frac{N_i}{M}$$

The period of this class of PRNGs depends on the choice of constants and is at most M (N_i will be between 0 and $M - 1$, so there are only M possible values that N_i can take).

A reasonable choice is Park and Miller’s “Minimal Standard”, with $a = 16807$, $b = 0$, and $M = 2^{31} - 1 = 2147483647$, which passes most stringent tests for a PRNG. A particularly terrible choice is the so-called RANDU generator, $a = 65539$, $b = 0$, $M = 2^{31}$. This last choice was implemented on IBM machines for many years and has led to a plethora of inaccurate results (and probably still undiscovered ones).

The following python code is a simple implementation of the LCG, which uses the Park Miller minimal standard by default. Note the use of a “function attribute”, `LCG.seed`, to store a number that the `LCG()` function updates (it starts with the seed of the sequence, and then gets updated with the latest value of N_i to generate a new number the next time that it is called). A histogram of the values generated is shown in Figure 10.1.

¹The modulo operation gives the remainder of a division. For example, $11 \bmod 3$ is 2; the remainder of dividing 11 by 3 is 2.

Python Code 10.1: Park-Miller minimal standard LCG

```

import numpy as np
import pylab as pl

#Define a function that returns LCG (pseudo) random numbers
def LCG (a=16807,b=0,M=2147483647):
    N1 = (a*LCG.seed+b) % M
    LCG.seed = N1
    return N1/M

#Initialize the seed:
LCG.seed=10135

#Generate an array of random numbers from the the LCG
nr=10000
r=np.zeros(nr)
for i in range(nr):
    r[i] = LCG()

#Plot a histogram of the values:
pl.hist(r,bins=100,color='gray')
pl.xlabel('random number')
pl.ylabel('frequency')
pl.title('10,000 numbers for the Park-Miller LCG')
pl.show()

```

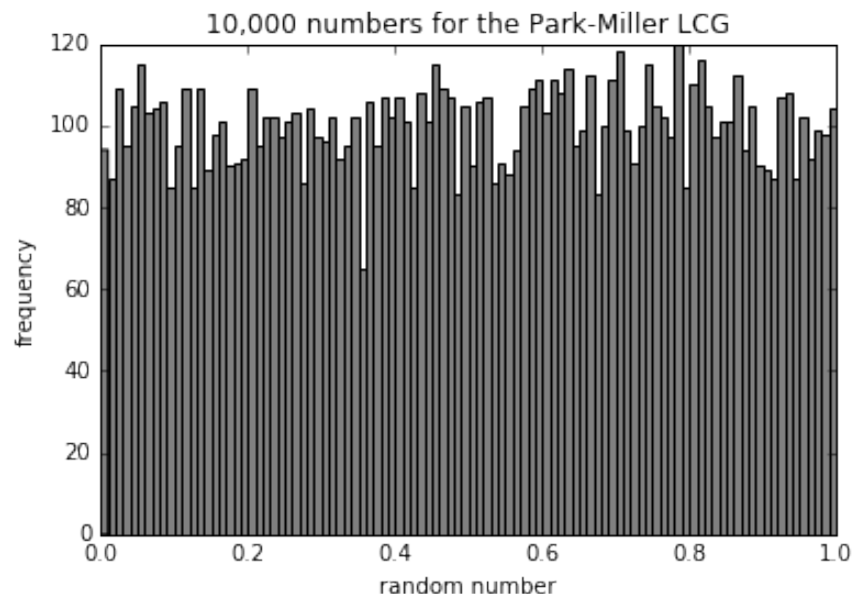
10.1: Output

Figure 10.1: Distribution of 10,000 numbers generated with the Park Miller Minimal Standard LCG.

Although LCG PRNGs are easy to implement, there are many much more advanced algo-

rithms that should be used when one needs random numbers. The LCG algorithm was only given here as an example of “tricking” a computer to output seemingly random numbers.

10.1.1 Random numbers that follow a given distribution

Although a good PRNG should be uniform, it is often desirable to generate random numbers according to a specific distribution. For example, one may want to generate numbers that are normally distributed rather than uniformly distributed. This can be done straightforwardly for any distribution using uniformly distributed random numbers.

If we have uniform random numbers, x , we wish to find a function, $y(x)$, such that the numbers, y , are distributed according to some chosen distribution, $g(y)$. Consider the cumulative distribution function (cdf), $G(y)$, given by:

$$G(y) = \int_{-\infty}^y g(y)dy$$

If we now choose the values of $y(x)$ such that $G(y)$ are uniform, then the values of y will be distributed according to $g(y)$. If x is a uniform random number, we can choose the y such that:

$$x(y) = G(y)$$

and invert the functions, such that:

$$y(x) = G^{-1}(x)$$

The function inversion cannot usually be done analytically, although one can easily program a function inversion using interpolation from tabulated values. Also note that for distributions, $g(y)$ that are not monotonous, it may not be possible to use this method if the function inverse is not one-to-one. For example, if this method were used for the normal distribution, then it would only generate values on one side of the mean (but the algorithm could easily be modified to generate numbers that are symmetric about the mean).

As an example that can be done analytically, let us suppose that we want to generate random numbers that are distributed according to an exponential probability density function, $g(y) = \frac{1}{\tau}e^{-\frac{y}{\tau}}$. The cdf, $G(y)$, is easily found:

$$\begin{aligned} g(y) &= \frac{1}{\tau}e^{-\frac{y}{\tau}} \\ G(y) &= \int_0^y g(y) = 1 - e^{-\frac{y}{\tau}} \end{aligned}$$

Setting $G(y)$ equal to x and inverting:

$$\begin{aligned} x(y) &= G(y) = 1 - e^{-\frac{y}{\tau}} \\ y(x) &= G^{-1}(x) = -\tau \ln(1 - x) \end{aligned}$$

Since the x are uniformly distributed between 0 and 1, $1 - x$ is also uniformly distributed between 0 and 1, and we can simplify this to:

$$y(x) = G^{-1}(x) = -\tau \ln x$$

where the “new” x ($= 1 - x$) are uniformly distributed between 0 and 1, as were the original x . That is, if we generate uniform random numbers, x , take their natural logarithm and multiply by $-\tau$, the resulting numbers will have a negative exponential distribution with decay constant τ . This is illustrated in the following python code, which uses the uniform random numbers, r , generated in the code above with the LCG and converts them to numbers, y , that are exponentially distributed. The resulting histogram is shown in Figure 10.2.

Python Code 10.2: Converting uniform PRNs to exponential PRNs

```
#using the array of values , r , we transform these into values y:
tau = 10
y = -tau * np.log(r) #numpy log is really ln
#Plot a histogram of the values:
n,bins ,patches=pl.hist(y,bins=100,color='gray')
pl.xlabel('exponential random number')
pl.ylabel('frequency')
pl.title('10,000 exponentially distributed numbers')
#plot the corresponding exponential (normalized by the number of y values)
xi=np.linspace(0,y.max(),100)
norm = y.size/(bins[1]-bins[0])/tau
pl.plot(xi,norm*np.exp(-xi/tau),color='red',lw=2,label="exponential")
pl.legend()
pl.show()
```

10.2: Output

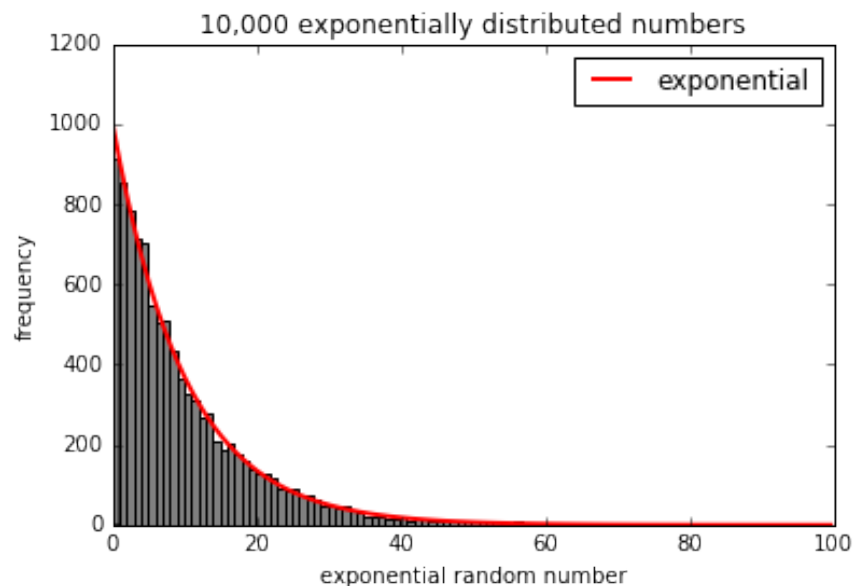


Figure 10.2: Distribution of 10,000 exponentially distributed random numbers obtained from the numbers in Figure 10.1.

In practice, we generally do not need to write our own pseudo-random number generators, and we can usually use built-in functions to generate random numbers according to common distributions. For example, the `numpy.random` module in python has many distributions already available, as illustrated in the following code, with histograms of the various random numbers shown in Figure 10.3.

Python Code 10.3: Standard distributions of PRNs

```
nr = 10000 # number of random numbers to generate
#Uniform distribution:
xmin=0
xmax=10
xunif = np.random.uniform(xmin,xmax,nr)
#Normal distribution:
mu = 5
sigma = 2
xnorm = np.random.normal(mu,sigma,nr)
#Binomial distribution:
N=10
p=0.5
xbinom = np.random.binomial(N,p,nr)
#Poisson distribution:
n=3
xpoiss = np.random.poisson(n,nr)
#Plot them all:
bins = np.linspace(0,10,50)
pl.hist(xunif,bins=bins,color='red',alpha=0.5,label='uniform')
pl.hist(xnorm,bins=bins,color='blue',alpha=0.5,label='normal')
pl.hist(xbinom,bins=bins,color='black',alpha=0.5,label='binomial')
pl.hist(xpoiss,bins=bins,color='orange',alpha=0.5,label='poisson')
pl.legend()
pl.show()
```

10.3: Output

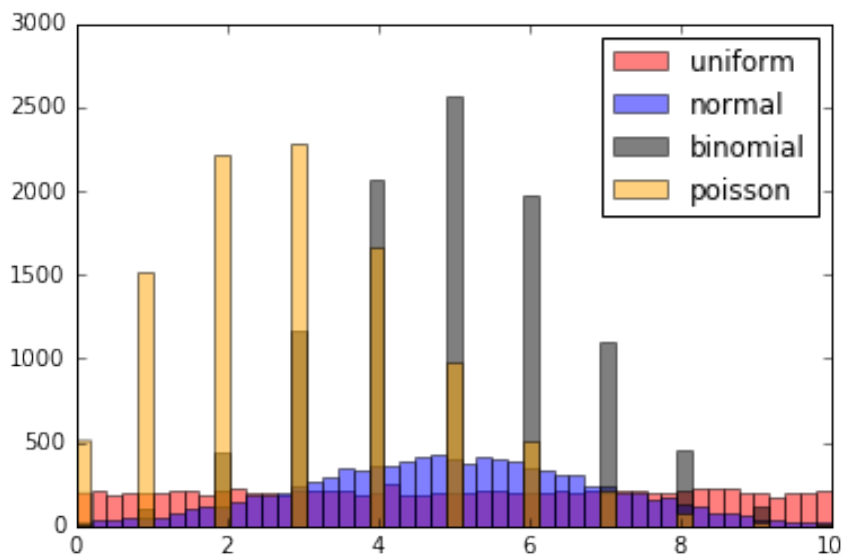


Figure 10.3: Histogram of 10,000 random numbers generated in python with uniform, normal, binomial, and poisson distributions.

10.2 Monte Carlo simulations

In this section, we examine a few example of Monte Carlo simulations that use random numbers to model a scenario.

10.2.1 Simulating coin tosses

Let us perform a Monte Carlo simulation of tossing a coin ten times. One “experiment” is to throw the coin ten times, and to count the number of heads that we obtained. Over many experiments, we know that the number of heads that we obtain will be binomially distributed. Let us suppose that our coin is unfair and has a probability $p = 0.7$ of landing on heads. We can use Monte Carlo simulation to obtain the distribution of the number of times that we get heads in ten coin tosses and, hopefully, recover the binomial distribution.

To simulate a single coin toss, we can generate a random number, r , that is uniformly distributed between 0 and 1. We can define the coin toss to be heads if $r < p = 0.7$ and tails if $r \geq p = 0.7$. Since r is uniform, on average we will get 70% heads and 30% tails, as desired. To simulate counting the number of heads in ten coin tosses (a single experiment), we generate ten values of r and count how many of them are smaller than 0.7, as in the python code below:

Python Code 10.4: Simulating 10 coin tosses

```
#Simulation of a set of 10 coin tosses:
#Make 10 random numbers between 0 and 1:
N=10
r = np.random.uniform(0.0,1.0,N)
#Count how many are smaller than 0.7
nheads = r[r<0.7].size
print(nheads," heads in 10 tosses")
```

10.4: Output

```
6 heads in 10 tosses
```

We can now simulate this experiment many times (say 10,000), and obtain the distribution of the number of heads that we get, as in the code below:

Python Code 10.5: Many simulations of throwing a coin 10 times

```
import scipy.stats as stats
Nexp=10000 # repeat the experiment 10,000 times
nheads = np.zeros(Nexp) # array to hold the result of each experiment
N=10
#Conduct the Nexp experiments:
for i in range(Nexp):
    #A single experiment
    r = np.random.uniform(0.0,1.0,N)
    #Count how many are smaller than 0.7, add to array of results
    nheads[i] = r[r<0.7].size
```

```

#Plot the result:
bins=np.linspace(-0.5,N+0.5,N+2)
pl.hist(nheads,bins=bins, color='gray')
pl.xlabel("number of heads in 10 tosses")
pl.ylabel("frequency")
#Plot the binomial distribution
norm = Nexp/(bins[1]-bins[0]) #normalization, since the histogram is not
    normalized
xi=np.arange(N+1)
pl.plot(xi,norm*stats.binom.pmf(xi,N,0.7),color='black',lw=3,label='binomial
    pmf')
pl.legend(loc='best')
pl.show()

```

10.5: Output

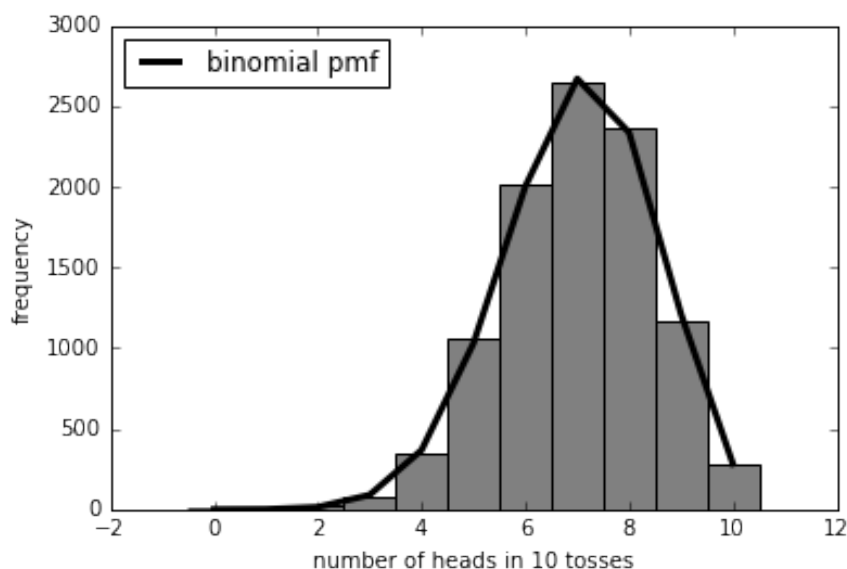


Figure 10.4: Histogram of the outcome (the number of heads) of 10,000 coin tosses of an unfair coin with $p = 0.7$ of landing heads. Overlaid, is the corresponding binomial distribution, which we expect the results to follow.

Figure 10.4 shows a histogram of the number of heads obtained in the 10,000 experiments. The corresponding binomial distribution, with $p = 0.7$ and $N = 10$ is shown and seen to describe the results of the Monte Carlo simulation, as expected. It is worth noting that we have also effectively implemented an algorithm that can use uniform random numbers to generate binomially-distributed random numbers!

In the previous example, we could have easily calculated the result, since we know from first principles that the distribution of results is binomially distributed. Let us consider a more complicated example, where the distribution of outcomes is not obvious.

10.2.2 Simulating a poutine restaurant

Suppose that we are operating a poutine restaurant, and need to decide if it makes sense to stay open one hour later each day. By using Monte Carlo simulation, we can simulate the expected profits of staying open one extra hour to see if they would on average offset the cost of paying to staff the restaurant. We sell 3 types of poutine (traditional, pork, and vegetarian), and each type has a different popularity and results in different amounts of profit (excluding cost of personnel), as tabulated in Table 10.1.

Poutine type	Percent ordered	Profit in \$
Traditional	70%	\$2.00
Pork	20%	\$2.50
Vegetarian	10%	\$0.50

Table 10.1: Frequency that each type of poutine is ordered and profit on each type.

Let us suppose that during a trial period, we have measured that, on average, 15 customers buy poutines in the restaurant during this extra hour. On average, customers buy 1 poutine, but sometimes they buy more (e.g. for their friends). We only count people that enter the store as a customer if they bought at least 1 poutine. For our Monte Carlo simulation, we will consider as a single “experiment” determining the profits in one night of staying open for the extra hour. We will then repeat the experiment many times to obtain the distribution of profits that we expect when having the store open for an extra hour for many nights. Our strategy for a single experiment (one night) is as follows:

1. Draw a random number, `ncustomers`, from a Poisson distribution with a mean 15 to be the number of customers that night.
2. For each customer, draw a random number, `npoutines`, from a Poisson distribution with a mean of 1 to be the number of poutines ordered by that customer. If the number is 0, draw again.
3. For each poutine, draw a uniform random number between 0 and 1, `choice`, to represent which type of poutine it is. If `choice` is between 0 and 0.7, it is a traditional poutine, if it is between 0.7 and 0.9, it is a pork poutine, and if it is between 0.9 and 1, it is a vegetarian poutine.
4. Add up the profits for all of the poutines, for all of the customers.

The following python code performs this Monte Carlo simulation, giving the distribution of expected profits in a histogram shown in Figure 10.5.

Python Code 10.6: Monte Carlo of the poutine restaurant

```
Nexp = 10000 # the number of experiments (nights)
profits = np.zeros(Nexp) # array to hold the profits for each night
#Perform Nexp experiments
for iexp in range(Nexp):
    #For each night:
```



```

#Generate a random number of customers with (Poisson) mean 15
ncustomers = np.random.poisson(15)
#For each customer, simulate their order:
for icustomer in range(ncustomers):
    #Generate a random number of poutines with Poisson mean of 1
    #Repeat until the number is bigger than 0
    npoutines=0
    while(npoutines<1):
        npoutines = np.random.poisson(1)
    #For each poutine, decide what type it is
    for ipoutine in range(npoutines):
        #Draw a uniform random number between 0 and 1 for the choice
        choice = np.random.uniform()
        #based on the type, add in the profits for that night:
        if choice<0.7: #traditional
            profits[iexp] = profits[iexp]+2.00
        elif choice>=0.7 and choice <0.9: #pork
            profits[iexp] = profits[iexp]+2.50
        else: #vegetarian
            profits[iexp] = profits[iexp]+0.50

print("Mean profits: ${:.2f}".format(profits.mean()))
#Plot it:
pl.hist(profits ,bins=20,color='gray')
pl.xlabel('profits ($)')
pl.title('Profits simulated for 10,000 nights')
pl.show()

```

10.6: Output

Mean profits: \ \$46.08

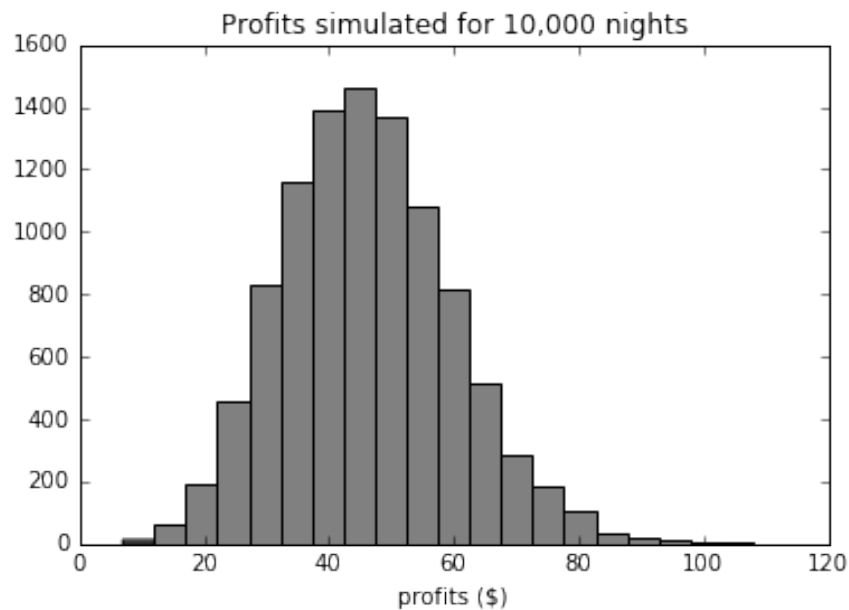


Figure 10.5: Histogram of the distribution of profits in the last hour of the poutine store.

We can conclude from our simulation that on average, we will generate \$46 of profit by staying open an extra hour, which will be worthwhile if that is enough to cover the cost of the staff. The histogram, if normalized, is truly the probability density function for the expected profits of the store. We can use this pdf to estimate other quantities that may be of interest, such as the profit that we expect to make at least 90% of the time (by finding the value where the cumulative distribution function is 0.1).

10.2.3 Monte Carlo simulation for error propagation

So far, we have used the general formula to propagate uncertainties in measured quantities x and y to the uncertainty in a function $F(x, y)$:

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

$$\sigma_F = \sqrt{\left(\frac{\partial F}{\partial x} \sigma_x\right)^2 + \left(\frac{\partial F}{\partial y} \sigma_y\right)^2 + 2 \frac{\partial F}{\partial x} \frac{\partial F}{\partial y} \sigma_{xy}} \quad (10.1)$$

As you recall from Chapter 7, this formula was derived from a Taylor series expansion of $F(x, y)$ near the best estimates for the measured quantities x and y . The Taylor series is only valid as long as the error on x and y are small, which is not necessarily always the case. So how do we handle error propagation when the errors are not small? One obvious, albeit mathematically inconvenient, answer is to add more terms into the Taylor series. We could also use the Min-Max method. However, as we will show here, it turns out the the Monte Carlo method is by far the easiest to apply, and in fact will work in all cases.

Let us suppose that we are interested in measuring Coulomb's force from two charges, q_1 and q_2 , that are a distance r apart. The force is given by:

$$F = k \frac{q_1 q_2}{r^2}$$

Let us suppose that we have measured all of the quantities with 1% relative uncertainty, and obtained the following values: $q_1 = 10^{-6} \text{ C}$, $q_2 = 2 \times 10^{-5} \text{ C}$, and $r = 0.1 \text{ m}$ (all measurements are $\pm 1\%$). The following python code calculates the uncertainty using standard error propagation:

Python Code 10.7: Standard error propagation

```
import numpy as np
import pylab as pl
import sympy as sym

#Measured (or numerical) values:
k_meas = 9e9
e = 1.6e-19
q1_meas = 1e-6
sigma_q1_meas = 0.01*q1_meas
q2_meas = 2e-5
sigma_q2_meas = 0.01*q2_meas
r_meas = 0.1
sigma_r_meas = 0.01*r_meas
```

```

#Define symbols
k,q1,q2,r,sigma_q1,sigma_q2,sigma_r= sym.symbols('k, q-1 q-2 r sigma-{q1}
sigma-{q2} sigma-{r}')
#F and its uncertainty
F = k*q1*q2/r**2
sigma_F =sym.sqrt((sym.diff(F,q1)*sigma_q1)**2+ (sym.diff(F,q2)*sigma_q2)**2+
(sym.diff(F,r)*sigma_r)**2)
values = [(k,k_meas),(q1,q1_meas),(q2,q2_meas),(r,r_meas),
(sigma_q1,sigma_q1_meas),(sigma_q2,sigma_q2_meas),(sigma_r,
sigma_r_meas)]
F_val = F.subs(values)
sigma_F_val = sigma_F.subs(values)
print("Using error propagation, the force is:")
print("F = {:.2f} +/- {:.2f} N".format(F_val,sigma_F_val))

```

10.7: Output

```

Using error propagation, the force is:
F = 18.00 +/- 0.44 N

```

Now let us compare this with a Monte Carlo simulation. We will assume that the errors in q_1 , q_2 , and r are standard, although it is easy to relax this assumption by using different distributions. The strategy is very simple:

1. generate N randomly normally distributed² samples for q_1 , q_2 , and r
2. for each sample, calculate a corresponding value of F
3. histogram F to see what the distribution looks like
4. usually, pick the mean and the standard deviation of the simulated values of F as the best estimate and uncertainty on F

This is accomplished in the following python code:

Python Code 10.8: Monte Carlo error propagation

```

import numpy as np
import pylab as pl

#Measured (or numerical) values:
k_meas = 9e9
e = 1.6e-19
q1_meas = 1e-6
sigma_q1_meas = 0.01*q1_meas
q2_meas = 2e-5
sigma_q2_meas = 0.01*q2_meas
r_meas = 0.1
sigma_r_meas = 0.01*r_meas

#Generate normally distributed values of r, q1, q2 about their measured
values:
N = 1000000 # number of MC samples

```

²If the errors were not standard, but corresponded to some different distribution, we could generate the random numbers from that distribution.

```

#Normally distributed values of q1, q2, and r:
r_sim = np.random.normal(r_meas, sigma_r_meas, N)
q1_sim = np.random.normal(q1_meas, sigma_q1_meas, N)
q2_sim = np.random.normal(q2_meas, sigma_q2_meas, N)

#Generate the corresponding values for the force:
F_sim = k_meas*q1_sim*q2_sim/r_sim**2

#Plot a histogram of the simulated values of F
pl.hist(F_sim, bins=50, color='gray')
pl.xlabel("Simulated Coulomb Force/N")
pl.show()

print("Using Monte Carlo simulation mean and standard deviation, the force is
:")
print("F = {:.2f} +/- {:.2f} N".format(F_sim.mean(), F_sim.std(ddof=1)))

```

10.8: Output

Using Monte Carlo simulation mean and standard deviation, the force is:
 $F = 18.00 \pm 0.44 \text{ N}$

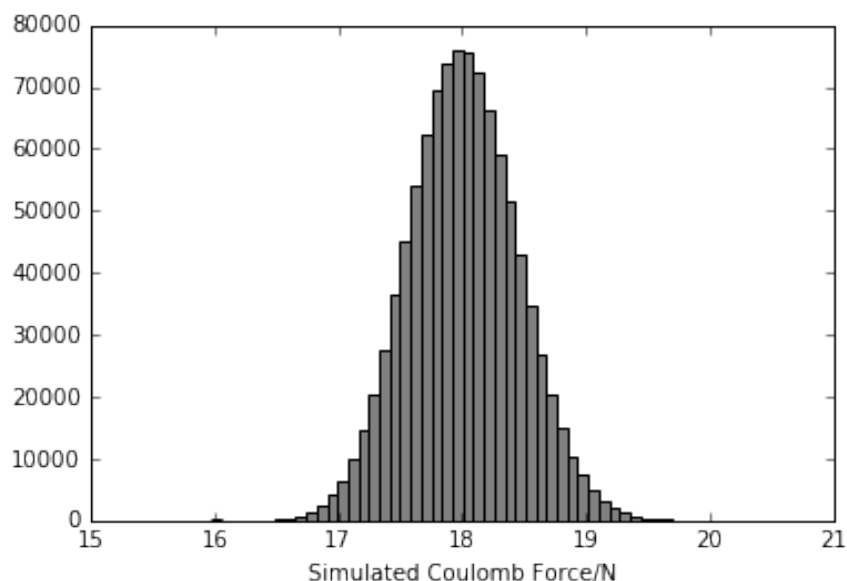


Figure 10.6: Histogram of the simulated values of the Coulomb force when the relative errors on the measured charges and the separation between them is 1%.

Figure 10.6 shows the histogram of the simulated values of the Coulomb force, and we note that it is very close to being normally distributed, thus supporting the use of the Taylor series expansion to propagate the uncertainty. When we take the mean and standard deviations of the values in the histogram, we get an estimate of F that is consistent with the obtained from the error propagation formula, as expected. It is worth noting that the Monte Carlo method is quite a bit easier to implement, since it does not require evaluating the partial derivatives.

We now repeat the same exercise, but using 10% relative uncertainties on all measurements. Modifying the relevant lines in the python code, we get the following output:

Using error propagation, the force is:

$F = 18.00 \pm 4.41 \text{ N}$

Using Monte Carlo simulation mean and standard deviation, the force is:

$F = 18.56 \pm 4.75 \text{ N}$

which now disagree in both the best estimate of F and on the uncertainty. This means that **the values from the error propagation formula are wrong**. The histogram of the simulated values is shown in Figure 10.7, where we can observe that the distribution of simulated values of F is no longer normally distributed (which we no longer expect if the Taylor series approximation no longer holds).

If we quote our MC estimate of F as the mean and standard deviation of the simulated values, we need to be careful to specify that this is not a standard error, that is, one cannot assume that F is normally distributed about our quoted value. It is ok to use the mean and standard deviation when reporting the result, but one should be very clear in specifying what is reported.

We can also choose to report the estimate of F in a different way, for example, by choosing the mode of the distribution (the “most probable value”) and choosing an uncertainty that covers 68% of the area around the mode, as long as we precisely report what was done. The following python code calculates the range of values symmetrically about the mode of the histogram that give approximately 68% coverage, which is shown by the lines on Figure 10.7.

Python Code 10.9: Selecting 68% coverage errors

```
#Histogram the values and keep the counts in each bin
n,bins,patches = pl.hist(F_sim,bins=50,color='gray')
#Find the index of the bin with the highest content (The mode)
n_max = n.max()
index = (np.abs(n-n_max)).argmin()
#The corresponding value of the force
best_F = bins[index]
#Iterate to find the 68% coverage
d=0 # add or subtract d iteratively to index to cover larger ranges of the
    histogram
area = n[index-d:index+d].sum()/n.sum() #area between index-d and index+d

while area < 0.68: #normalize area each time it's calculated
    d = d+1
    area = n[index-d:index+d].sum()/n.sum()

sigma_F = best_F-bins[index-d]

print("Using Monte Carlo mode and 68% confidence:")
print("The force is {:.2f} +/- {:.2f} N, with {:.1f}% confidence".format(
```

```

best_F , sigma_F , 100*area))

pl.plot([best_F , best_F] , [0 , n_max] , color='red' , lw=3 , label='mode = {:.2f} N'.
        format(best_F))
pl.plot([best_F-sigma_F , best_F-sigma_F] , [0 , n_max] , '—' , color='red' , lw=3 , label
        ='lower bound = {:.2f} N'.format(best_F-sigma_F))
pl.plot([best_F+sigma_F , best_F+sigma_F] , [0 , n_max] , '—' , color='red' , lw=3 , label
        ='upper bound = {:.2f} N'.format(best_F+sigma_F))
pl.xlabel("Simulated Coulomb Force/N")
pl.title("F = {:.2f} +/- {:.2f} N, with {:.1f}% confidence".format(best_F ,
        sigma_F , 100*area))
pl.legend()
pl.show()

```

10.9: Output

Using Monte Carlo mode and 68% confidence:
The force is 16.21 +/- 4.69 N, with 70.2% confidence

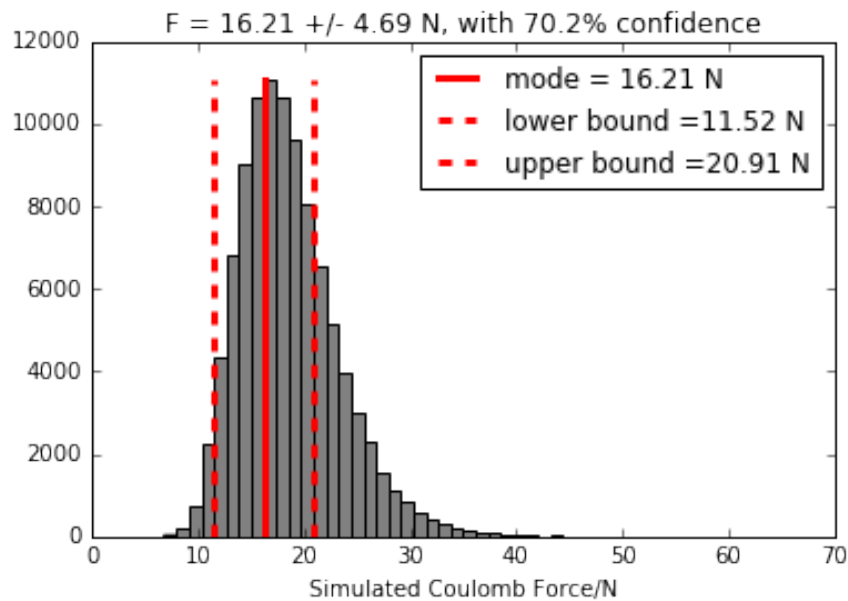


Figure 10.7: Histogram of the simulated values of the Coulomb force when the relative errors on the measured charges and the separation between them is 10%. The distribution is no longer gaussian. The red lines show the mode of the distribution and a symmetric range about the mode that cover approximately 68% of the values.

It is worth noting that the Monte Carlo method always gives the correct estimate of the best estimate and uncertainty, unlike the error propagation formula which relies on a Taylor series being valid. However, for the Monte Carlo method to be correct, one must also include any potential correlation between the parameters as well as the correct distribution for the measured values (which may not be normally distributed), although this caveat applies to the error propagation formula as well. As a minimum, one should use the Monte Carlo method to evaluate whether the error propagation formula is being used in a valid regime. It is also worth noting that, in general, the Monte Carlo method is easier to implement. It

will also likely impress your physics professors if you use the MC method to evaluate the uncertainties in an undergraduate laboratory!