Physics Skill Manual

R. Dawson Baker

PUBLISHED BY PUBLISHER

BOOK-WEBSITE.COM

First printing, March 2013

Contents

1 — Typesetting documents in LaTeX

LaTeX(pronounced "lah-tech" or " lay-tech") is a mark-up language that allows an author to specifically, precisely and easily specify how layout and content are combined. It is based on Dr. Donald Knuth's TeXtypesetting system with two goals in mind:

1. It should be easy a require only a "minimal amount of effort" for the author to produce high-quality documents, and
2. It should run on every operating system.

Both these goals have been essentially accomplished, resulting in wide-spread use among the scientific community. At this point – while there are alternatives for typesetting scientific documents – LaTeX can easily be considered a basic tool that every physicist should get to know well. Besides making equations look good, the mark-up language is commonly used for:

1. **Making digital and hard-copy media**. This book itself was typeset in LaTeX.
2. **Making navigable documents by using hyperlinks in a pdf**. This means that you can click on certain words in the document and it will take you to another place in the document, another document, or even a website.
3. **Making high quality documents whose fonts and images can be scaled to be printed on many different kinds of printers**.

These features, combined with the fact that the language is open-source and free to use, have made LaTeX a staple in the world of publishing and scientific journals. For instance, many scientific journals prefer or exclusively accept documents in LaTeX code because it is so easy to get to a finished product, and LaTeX documents contain all the information necessary to render the product regardless of platform. If you are considering submitting something to the ArXiv (a well known open-access database for preprints) at some point, you *should* know LaTeX . While you *could* get by without it, very few people do.

To help you get to know LaTeX, this book will focus on three things: (1) explaining the basics in what is hoped is a concise way, (2) pointing you towards good online resources on the subject if you're confused and (3) (if you're up to it) helping you to get started with REVTeX, a package tailored specifically to the needs of physicists.

1.1 Getting LaTeX – compilers and editors

To use LaTeX, you need to (1) download the compiler for your operating system, then (2) download a LaTeX editor that you like. The compiler is the program which does the actual typesetting by creating a `.pdf` file, and it is different for every operating system:

1. Microsoft Windows $\Longrightarrow$ MikTeX: `http://miktex.org/`
2. Mac OS $\Longrightarrow$ MacTeX: `http://tug.org/mactex/`
3. Linux $\Longrightarrow$ look for the `texlive` package in your package manager.

There are *many* LaTeX editors, each of them has its own advantages and disadvantages. Some of the favorites at UT are [1]:

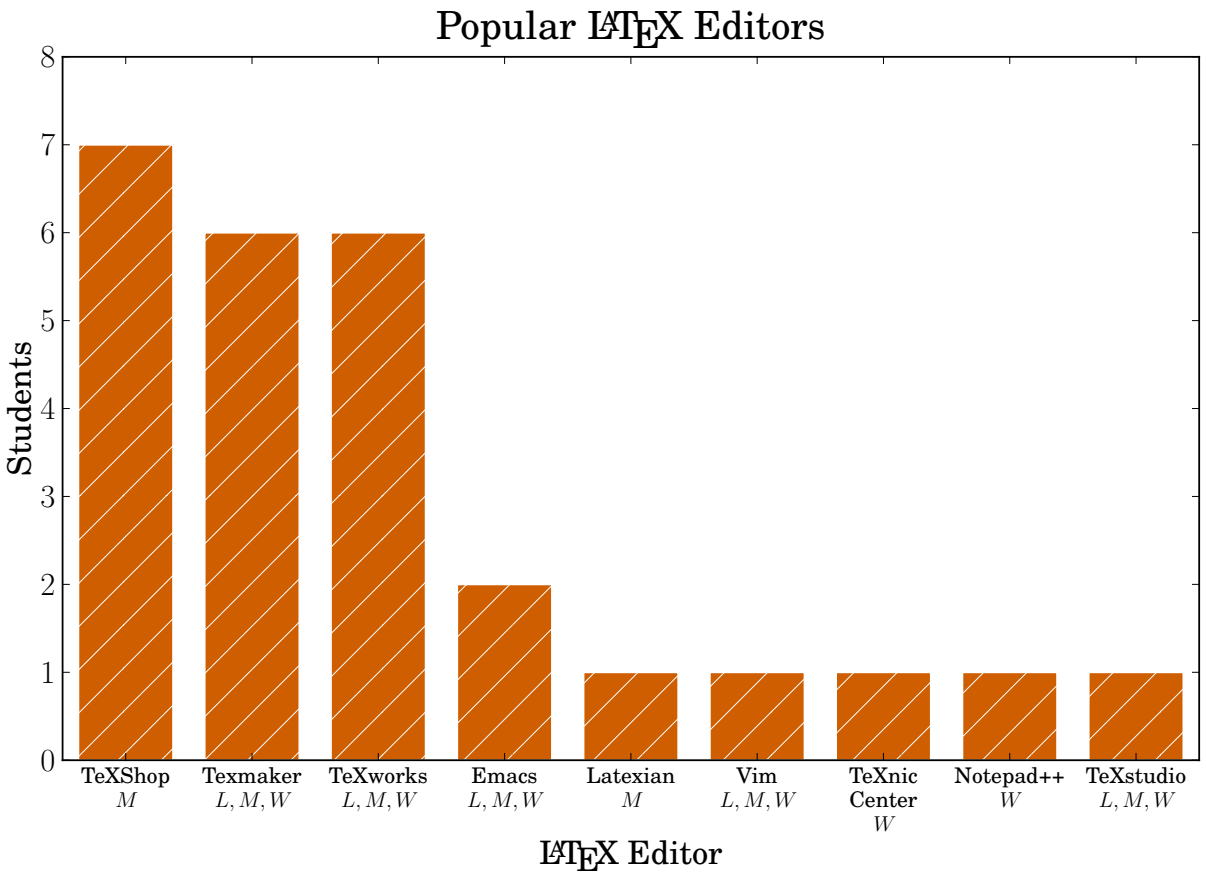


Figure 1.1: A Facebook survey was taken of other UT SPS students' LaTeX editor preferences. The plot above documents the results. Under each editor name, the italicized capital letters show which operating systems the programs are compatible with: *L* for Linux, *M* for Mac OS X, and *W* for Windows. If you choose one of these, you are most likely to find someone else who knows about the editor you're using (in case you get lost). *Note:* Emacs, Vim, and Notepad++ are text editors that can be modified to run LaTeX code. If you're a serious programmer you may already know what these are. If you're new to LaTeX , it might be best to try something else first.

A good compare and contrast of some of the different LaTeX editors out there can be found at `http://en.wikipedia.org/wiki/Comparison_of_TeX_editors`

[1] This book was typeset using Texmaker. The hotkeys and structure manager came in very handy when working through a lot of code and trying to manage multiple chapters.

1.2 Getting started immediately

This section is intended to be short and the absolute minimum to get you started with writing a paper in LaTeX. If you are more interested in a comprehensive introduction you should *certainly* check out these two much more complete guides:

The LaTeX Wikibook Found at `http://en.wikibooks.org/wiki/LaTeX`, this is probably the best organized and easiest to navigate website. It is slightly more encyclopedic and useful for troubleshooting.

The Not So Short Introduction to LaTeX2e If you're interested in having a `.pdf` or book reference, `http://tobi.oetiker.ch/lshort/lshort.pdf` is a great document that has a lot in common with The LaTeX Wikibook, but is typeset entirely in LaTeX and has a bit more of an attitude.

Both of these references are very similar, and are very enlightening. For a more full understanding of LaTeX (it would be senseless to try to compete with them) you should definitely spend a few hours reading through them. To get writing immediately refer to the document and explanation below.

1.2.1 A basic document

An example of a simple yet functional basic document that has title, abstract, some organization and an equation:

```
% This is a comment because nothing after a % sign can be displayed.
\documentclass{article}
\title{This is a title}
\author{Name}
\date{\today}

\begin{document}
\maketitle
\begin{abstract}
This is my abstract
\end{abstract}
\begin{section}{This is a section}
\begin{subsection}{This is a subsection}
Here is text and below is an equation:
\begin{equation}
\hat H = - \frac{\nabla^2}{2} - \frac{1}{x}
\end{equation}
\end{subsection}
\end{section}
\end{document}
```

This code needs to be run for the compiler to yield a document. When run in a LaTeX editor the code produces output that looks like this:

This is a title

Name

July 25, 2013

Abstract

This is my abstract

1 This is a section

1.1 This is a subsection

Here is text and below is an equation:

$$\hat{H} = -\frac{\nabla^2}{2} - \frac{1}{x} \tag{1}$$

1.2.2 Commands and environments

To understand how this file made it to output, let's look at the general features of the file. First of all, notice that every word that has a \ in front of it is not displayed. This is because commands in LaTeX start with a backslash. Commands represent blocks of code in TeX , the underlying language that is run to create and render all of the typeset objects that you see in the output document. Second, notice that every time there is a command of the form `\begin{` *environment* `}`, there is another of the form `\end{` *environment* `}` some time later. This is because these commands delimit (set the bounds of) blocks of code called "environments". Environments are spaces in which all code is interpreted in a specific way. Some commands and environments therefore only make sense or can be used within certain kinds of environments. By thinking in terms of *commands* and *environments*, LaTeX translates a few words into a lot of specific TeX commands that we would really rather not see (because they are incredibly tedious). This is why the system is so powerful: you only need to give LaTeX your content, and it does all the formatting according to rules you can understand and tinker with. **Learning how to use the system well is therefore really just a matter of understanding how the existing rules work and how new rules are made.**

1.3 How the existing rules work

The undisputed best way to understand how the rules of LaTeX work is to actually try to write some files. While this is true, it is handy to know about the basic structure and conventions of a `.tex` document when searching for documentation and for imitating examples found online. This section will therefore cover the structure of LaTeX through the lens of the basic document above.

1.3.1 The structure of a LaTeX document

Every document starts by choosing what class of document it is (is it an article? a letter? a book?) . This is done in the example with the `\documentclass{article}` command. When LaTeX sees this command it looks for a specific `.cls` file that has all of the basic formatting information. In the example, it looked for `article.cls` and obtained valuable information like the paper size, font sizes, and other environments and commands used in the document. This roughly constitutes the 'lowest-level' of code, and contains a lot of TeX code. You *do not* want to mess with these class files unless you have a lot of practice and no

existing project can be modified to suit your needs (very rare).

In general, the class file will also load packages in the `.sty` file format. These are higher level files that are usually task-specific and contain even more commands and environments. For instance, `hyperref` is a package which allows for hyperlinks to websites, files, or other parts of the document. Packages are loaded using the `\usepackage{}` command (e.g. `\usepackage{hyperref}`) between the `\documentclass{}` command and `\begin{document}` in what is referred to as the *top matter* or *preamble*. The preamble is very important because it is where all of the extra functions you might want to use are added. For instance, if you want to change the way the title formatting looks or want to define a function to get rid of the tedium or repeating code, the preamble is the place to put this code.

Once the compiler sees `\begin{document}` the preamble is over and everything it reads is document content – no more commands and environments may be added or defined. When it gets to `\end{document}`, that's the end of the file and the LaTeX compiler will not evaluate anything past it.

1.3.2 Anatomy of our basic example

Commands used in the example

`\documentclass[]{}`	**loads the class file for the document which contains the basic information (commands and environments) the program needs to start typesetting your code** and is at the start of every LaTeX document. This basic information is information like what size paper you're using, whether you're writing something long like a book or short like a letter, and whether you'll be using a lot of complicated symbols. All these things have an impact on what kinds of tools the program will need to do the job correctly. In general, the class file will therefore have options that you put in the `[]` for these kinds of things and so it will look more like `\documentclass[11pt, letterpaper]{article}`.
`\title{}`	stores the title information so that the `\maketitle` command can use it later.
`\author{}`	stores the author information so that the `\maketitle` command can use it later.
`\date{}`	stores the date information so that the `\maketitle` command can use it later.
`\today`	retrieves the current date and outputs according to a certain format. This command can be put anywhere that text can be placed.
`\maketitle`	renders a title using information from `\title{}`, `\author{}`, `\date{}`
`\hat`	puts a hat symbol over the next symbol in the equation : e.g. $\hat{\alpha}$, $\hat{3}$, $\hat{r}$
`\frac`	makes a fraction with the first `{}` defining the numerator, and the second `{}` defining the denominator
`\nabla`	makes a del (nabla) symbol that looks like ∇

Environments used in the example

`document`	contains the contents of the document. This means it contains the other environments. It also means that anything after `\end{document}` will not be typeset.
`abstract`	interprets its contents as text that contains the abstract of an article. As an environment it formats the contents to look a certain way: changes the margins and font to look like an abstract.
`section`	interprets its contents as body material and makes a title for the section based on the number of sections before it. The environment assumes that the first `{}` it sees contains the title for the section. Therefore, since no section preceded the one in the example, the section was formatted as '1 This is a section'.
`subsection`	interprets its contents as body material and makes a title for the subsection based on the number of subsections before it and the section it's in.
`equation`	interprets its contents in math mode. This means that it interprets text as a string of a variables unless it is a command like `\hat` or `\frac{}{}`.

1.3.3 Basics continued: text & math modes

In text mode, you delimit the math environment by using $ signs. This means that whenever you want to put in a symbol, subscript, or equation into a sentence you can easily do that by wrapping the content in dollar signs.

Code Example 1.1 — Math in text mode.

```
Einstein's famous equation $ E = mc^2$ was not rigorously proven by
Einstein, but rather by the mathematician Felix Klein.
```

produces

Einstein's famous equation $E = mc^2$ was not rigorously proven by Einstein, but rather by the mathematician Felix Klein.

Some environments, like the `equation` environment we saw before, will be in math mode automatically. If you want to put text inside an equation, use the `\text{}` or `\mbox{}` commands to get into text mode.

Code Example 1.2 — Text in math mode.

```
\begin{equation}
\Delta E = E_{\text{Final}} - E_{\mbox{Initial}}
\end{equation}
```

produces

$$\Delta E = E_{\text{Final}} - E_{\text{Initial}} \tag{1.1}$$

> As you can see, both `\text{}` or `\mbox{}` are effective in putting text into the equation, **but only `\text{}` properly does subscripts and superscripts.**

Whichever mode you're in, extra spaces and extra line breaks are ignored. This helps reduce syntax ambiguity and helps writers visually organize separate content in the `.tex` without impacting how the file output looks when file is run.

1.4 Getting more packages

As we've seen so far, LaTeX is just a way of automating the underlying TeX language by use of commands and environments that summarize a lot of dirty work we don't want to do. There are a lot of writers over the years who have decided do more work to create and increasing number of packages full of more commands and environments to make their lives easier. The good news is that since LaTeX is open source, everyone can use and has access to those packages. These packages are used for inserting pictures, doing complicated referencing, and even making animations. This is partially why LaTeX is so great for scientists – it's open, automates repetitive tasks, and continuously improves.

Since you probably have to make use of these packages, you will have to use the `\usepackage` command mentioned briefly in section 1.3.1. When you installed LaTeX however, you might not have installed all the packages you need. This is natural and unavoidable since new packages are written every day. To get at these packages though, the easiest thing to try first is just to use the by far is to use `\usepackage` command. If you have a good *package manager*, it will go find the package for you and install it. If you do not have such luck, you can access the package managers directly:

1. MiKTeX's package manager is called MiKTeX Package Manager (MPM) appropriately enough. You can get to it through the start menu (it'll say "Browse Packages") on PC.
2. MacTeX's package manager is called TeX Live Utility, and can be accessed through the Applications folder on a mac.

1.5 More basics: Tables, Figures and References

Now that you know how to write math in text mode and text in math mode, you are essentially ready to write papers. The three most common things you'll probably want to know about after that are (1) how to make tables, (2) how to put figures in a document and (3) how to reference things (other papers, figures, equations, etc.). For this you may need to use some new packages as explained in

1.5.1 To make a table

Look at the code examples at `http://en.wikibooks.org/wiki/LaTeX/Tables#Basic_examples`. The gist of it is that you have to give LaTeX some instructions about how to align and seperate the columns, and tell it where each cell is (using the `&` alignment character) and where each line ends (using `\\`). You can divide up rows using the `\hline` command.

1.5.2 To add figures

In LaTeX, you either import some picture file by grabbing it with some code, or you create a picture with some code. The most common thing physicists do is import picture files, since making pictures is very complicated. The easiest thing to do to import pictures is to make a folder with the picture files in it, and then put it in the same folder as your `.tex` file. That way, wherever the folder containing your document goes, your pictures will follow.

There are a few picture formats you can use, but you want to use `.pdf` or `.eps` since they are vector graphics and don't look cruddy when you change their scale factor.**To learn how to use figures, read `http://en.wikibooks.org/wiki/LaTeX/Floats,_Figures_and_Captions` and use one of their examples**.

R One of the most common problems people have with figures is getting them to go in the right place. LaTeX by default searches for a good place to put it based on the density of text you have, but you can make it bend to your will by using placement specifiers:

`http://en.wikibooks.org/wiki/LaTeX/Floats,_Figures_and_Captions#placement`

Basically, with the `float` package you can tell LaTeX to put the figure exactly where you say. Sometimes this doesn't look so great, since the spacing isn't optimized for word density, but sometimes it's necessary.

1.5.3 References

One thing LaTeX makes easy for scientists is citing references, making footnotes[2], referring to equations and figures in their documents, and referring to websites.

Referencing other parts of your document

The most general and versatile command for labelling things for referencing is `\label{marker}`: `http://en.wikibooks.org/wiki/LaTeX/Labels_and_Cross-referencing` With this labelling command you can label just about anything. To use it all you have to do is give whatever you're labelling a name (a marker) and then use the `\ref{marker}` to reference it later. For instance if you have an equation below:

$$\hat{H}^{\text{core}}(1) = -\frac{1}{2}\nabla_1^2 - \sum_\alpha \frac{Z_\alpha}{r_{1\alpha}} \tag{1.2}$$

You can label it like so:

```
\begin{equation}\label{1ecorehamiltonian}
\hat H^{\text{core}}(1) = -\frac{1}{2}\nabla^2_1 - \sum_\alpha \frac{Z_\alpha}
{r_{1\alpha}}
\end{equation}
```

Now whenever you say `\ref{1ecorehamiltonian}` anywhere in text mode, "1.2" shows up. With this method, no matter how many equations you have, the equation number will be matched to the reference and you won't have to go through and tweak the numbers.

R When you label a figure, be sure to put the label command *after* the `\includegraphics{}` command, or else the numbers will be screwed up!

Citing other authors

Citing authors can be complicated, but the LaTeX wikibook online has very good coverage of this topic:

1. If you're looking for something easy, use the `thebibliography` environment found at `http://en.wikibooks.org/wiki/LaTeX/Bibliography_Management#Embedded_system`
2. If you're looking for a professional solution, use BibTeX `http://en.wikipedia.org/wiki/BibTeX`.

1.6 Templates in LaTeX

The easiest way to get started with a new project is to download an appropriate template and to experiment with it. A very basic template for writing papers was written by Jonathan Blair and can be found at the

[2] Making footnotes is as easy as putting `\footnote {some text}` exactly where you want the footnote to show up on the page.

SPS website – http://www.ph.utexas.edu/~sps/. More complicated templates can be found all over the internet.

Template websites

http://publish.aps.org/revtex	The standard class file for in publishing
http://www.latextemplates.com/	Best for books and modern styling.
http://www.howtotex.com/templates/	More templates – more of the same
https://www.sharelatex.com/templates/	Great assortment of article styles

2 — More advanced features of LaTeX

2.1 Creating new commands and environments

Since LaTeX is a programming language, you can make new commands.

2.1.1 Simple Substitutions

Use the

```
\def\function_name{value}
```

command in the preamble to define a simple substitution. Anywhere LaTeX sees

```
\function_name
```

, `value` will appear.

2.1.2 Functions with Parameters

Use the

```
\newcommand{\cmnd_name}[num_vars]{...#n...}
```

in the preamble. Again, `\cmnd_ name` is the new function name, but you can tell LaTeX how many variables you want, then use them to create a full“macro" to substitute, using # n to get the value of the n-th variable.

■ Example 2.1 — Dirac Notation.
Say for instance you were doing a lot of quantum mechanics and you wanted to be able to write bras, kets, and brakets very simply. You might want to do this since you might like your equations to be easy to read when you write them and writing it out is somewhat verbose. To make a braket command, you could type.

2.2 Installing REVTEX

By now, if you have written a bit of code and browsed the wikibook, you probably have a pretty basic and functional understanding of LaTeX and can format papers in the `article` class pretty well. To make it easier for physicists to write papers and to standardize formatting in the APS and AIP journals, APS has released a package they call REVTEX. The samples in this public package both look great, and have

excellent documentation embedded in the documents. To install:

1. Look in your package manager (section 1.4) and install REVTEX. It probably shows up in your package manager as `revtex4-1`.
2. Go to the SPS webpage and download the UTSPS REVTEX Sample Pack.
3. When you unzip this file, it will contain three folders
   aapm contains a sample `aapmsamp.tex` and a template `aapmtemplate.tex` from the American Association of Physicists in Medicine.
   aip same documents instead from the American Institute of Physics
   aps same documents instead from the American Physical Society.

After you are done unpacking, these files can be excellent references for producting high-quality reports and can be very valuable all-encompassing references if you are just writing a paper. The wikibook is a great way to learn LATEX in general, but these official APS and AIP documents are excellent templates for PHY 353L or PHY 474.

3 — The Basics of Error Analysis

There are many books on error analysis, but no physics research guide would be complete without at least an introduction to errors and experimental uncertainty. If you're interested in a more in-depth approach, some good references with exercises are:

1. *Dealing with Uncertainties: a Guide to Error Analysis* by Manfred Drosg, 2007 – more of a 'philosophically correct' approach
2. *A Student's Guide to Data and Error Analysis* by Herman J. C. Berendsen, 2011 – a more instructive and concise approach
3. *Data Reduction and Error Analysis for the Physical Sciences* by Bevington & Robinson, 2003 – a more encyclopedic approach

3.1 Experimental measurements and scientific results

> "[T]here can be no scientifically relevant data without uncertainty"
>
> – M. Drosg (2007, pg. 1)

The goal of error analysis is to understand exactly how accuracy and precision affect the result of an experiment, and to use this information to devise better experiments and avoid fallacious conclusions. Understanding error is essential for any physicist who wants to test the theory in the "real world".

In a perfectly accurate and precise experiment, there is no error analysis to speak of. Since there are no perfectly accurate or precise scientific instruments, a large part of the validity of experimental work revolves around quantifying how inaccurate and imprecise experimental observations are.

There is no such thing as an exact number in science [1], so it is important to know how to represent experimental results in terms of accuracy and precision.

3.1.1 Accuracy and precision: stating scientific results

The difference between accuracy and precision can be portrayed well by the statement:

> *Accuracy is how close measurements of a physical quantity come to predicting the true value, whereas precision is how close measurements of a physical quantity are to eachother.*

[1] For good philosophical discussions of uncertainty in science see Karl Popper's *The Logic of Scientific Discovery (2002)* or M. Drosg's *Dealing With Uncertainties (2007)*

In an experiment, results must always be written as a value with an associated uncertainty (*value* $\pm$ *uncertainty*). The *value*, if you compare it to the accepted value or some other measurement, will give you some idea of your accuracy – the *uncertainty* will tell you your precision. One of the most important habits to get used to is quoting your error and precision in a way that makes sense. Writing $3.452673 \pm .1$ or $3.5 \pm .128619$ makes no sense – the right number of significant digits must be used to convey to what extent you know your accuracy and your precision. Neither of these expressions works because the first ($3.452673 \pm .1$) has useless digits in the accuracy because the precision is too low and the second ($3.5 \pm .128619$) gives useless digits in the precision because the accuracy isn't high enough for the millionths place precision to be useful.

A good result has an appropriate number of significant figures in the *value* and the *uncertainty* that makes every digit of the result useful. When quoting uncertainty, the default is the *standard deviation* or *root-mean-square error* of the estimated probability distribution.

■ **Example 3.1** An abstract reads "We measure the frequency of oscillation of a comatose squirrel strapped a swingset and calculate a value of $\omega_0 = 1.34 \pm 0.07$ Hz. This is consistent with classical predictions..."

Here, the correct interpretation of ± 0.07 is that it is the *standard deviation* (in Hz) of a set of measured frequencies.

3.2 Kinds of error

In experimental, there are two basic categories of error:

1. **Systematic Errors**: These are reproducible (but may be time dependent, watch out!) errors that cannot be reduced through statistical analysis. Common instances of this are calibration errors and false assumptions about the operation of the apparatus. These errors cannot be eliminated as there is always uncertainty in the system specificied that yields the data.
2. **Statistical Errors (Random Errors)**: These are errors that can be accounted for using statistical analysis. The typical example is a measurement which fluctuates in a gaussian or normal way about some mean. It is impossible to eliminate these errors but it is theoretically always possible to make these errors smaller than the systematic error with enough data. At some point the uncertainty as to what 'identical conditions' are dominates.

Roughly speaking, systematic error has to do with accuracy and statistical error has to do with precision. Of course this is not necessarily the case: there can be nonisolatable systematic errors which yield higher uncertainties in your analysis, as well as statistical effects which

3.3 Error propagation

3.3.1 The concept of estimator

In experiment, there is a crucial difference between the number that you calculate from some statistical formula, and the parameter you're estimating by using that formula. Because there are no exact numbers in experiment, there will necessarily be some uncertainty as to what the "true" value is. It is so important not to get confused between true parameters and estimates that in the following sections and in the next chapter, I denote estimates with hats $\hat{}$ and leave the true parameters alone.

This is common practice in statistics, and for a proper understanding of the formulae in physics it is essential. For instance, the definition of variance σ_a^2 is that in equation 3.7, but as experimenters we are limited to using an *estimator* of that value ($\hat{\sigma}^2$) given in equation 3.10. Other authors sometimes choose s_a^2 to denote "sample variance", but I find the latin-to-greek convention there somewhat obfuscating. For every true parameter x that exists in a theoretical model, I call the formula for it that uses experimental data "$\hat{x}$, the estimator of x."

3.3.2 How to estimate error in an algebraic formula

R The equations in this section are convenient for those cases in which you are either:

1. forced to estimate uncertainty without the use of statistics
2. sure that the uncertainty in your input is approximately the same over the range over the range of your dataset

Whenever you have a variable which is equal to some formulation of measured variables (an equation, for instance $P = nRT/V$), the deviation in that variable (P) will depend on the deviation in the other variables from their expected values. That is, if each of the variables in a formula have some uncertainty to them, the result will also have an uncertainty. Remember that in an experiment this is always true since whatever you are calculating is by definition a formulation of your measurements and all measurements have uncertainty if they are scientifically relevant. Under the assumption that the function is well-behaved (continuous and differentiable) in the region of parameters you're looking at, a simple approach to this deviation is simply to use a Taylor expansion of the formula and propagate error.

Say for instance that you have some function $f(a,b)$ of some parameters a and b, which have errors which you expect to be about δa and δb large, respectively. The resulting deviation can typically be represented well by using a Taylor expansion:

$$|\delta f(a,b)| = \left|\frac{\partial f}{\partial a}\delta a\right| + \left|\frac{\partial f}{\partial b}\delta b\right| \tag{3.1}$$

This equation serves as a good estimate of maximum error or deviation, but it is *not* a good estimate of uncertainty. This is because the equation estimates the deviation of output based on deviation in input. Since measures of uncertainty are quoted in standard deviations and measures of statistical dispersion, an estimate of 'maximum deviation' is not sufficient and will often overstate the actual error. As an example of this, consider the function $f = \alpha a + \beta b$, where α and β are constants:

$$\delta f = \left|\frac{\partial f}{\partial a}\delta a\right| + \left|\frac{\partial f}{\partial b}\delta b\right| = |\alpha\delta a| + |\beta\delta b| \tag{3.2}$$

Now let's suppose that these δa and δb were actually fairly close to being good descriptors of standard deviation – that is, $\sigma_a \simeq \delta a$. Assuming that the data in a and b were independent, the error would be:

$$\sigma_f = \sqrt{\left(\frac{\partial f}{\partial a}\right)^2 \sigma_a^2 + \left(\frac{\partial f}{\partial b}\right)^2 \sigma_b^2} = \sqrt{\alpha^2\sigma_a^2 + \beta^2\sigma_b^2} \tag{3.3}$$

This is substantially different than eq. 3.2, since $\sqrt{x^2+y^2} < x^2+y^2$ for any two positive x and y. In short, eq. 3.2 is only a conservative bound for error which serves as a starting point in an experiment or calculation.

In summary, if you have a good estimate of the uncertainties in a function input, use (3.2) if you are in a rush and (3.3) if you want to be thorough *and* have a reason to believe the input errors should not be correlated.

3.3.3 With statistical uncertainties

Often, it is necessary to look at errors statistically. This is especially true fluctuations in the measured values vary beyond uncertainty for what are ostensibly the same experimental conditions. While it is often possible to estimate uncertainties using device specifications there is no surer method of obtaining information about random error than by repeating an experiment and getting more data.

In the notation here, $\bar{a}$ denotes the mean $\sum a_i/N$, and all partial derivatives are evaluated at the mean values ($\bar{a}$ and $\bar{b}$). It is important to keep in mind that using the statistical method of estimating uncertainties, σ_f^2 is calculated for a single point in the parameter space which we assume is $(\bar{a},\bar{b})$

To calculate σ_f^2, instead of using δa we now just say that the deviations $f_i - \bar{f}$ are related by the same Taylor expansion to the *deviations in the input variables ($a_i - \bar{a}$ and $b_i - \bar{b}$)* :

$$f_i - \bar{f} \simeq \frac{\partial f}{\partial a}(a_i - \bar{a}) + \frac{\partial f}{\partial b}(b_i - \bar{b}) \tag{3.4}$$

Since standard deviations, means, and other standard statistical results are built out of these deviations, such an expansion forms the theoretical basis for all error propagation statistics. Once a distribution of devations $f_i - \bar{f}$ over the data set is obtained, then we can construct σ_f^2, which is the uncertainty in our result f:

$$\sigma_f^2 = \lim_{N\to\infty} \frac{1}{N} \sum_i (f_i - \bar{f})^2 = \lim_{N\to\infty} \frac{1}{N} \sum_i \left[\frac{\partial f}{\partial a}(a_i - \bar{a}) + \frac{\partial f}{\partial b}(b_i - \bar{b}) \right]^2 \tag{3.5}$$

Expanding equation 3.5,

$$\sigma_f^2 \simeq \lim_{N\to\infty} \frac{1}{N} \sum_i \left[\left(\frac{\partial f}{\partial a} \right)^2 (a_i - \bar{a})^2 + \left(\frac{\partial f}{\partial b} \right)^2 (b_i - \bar{b})^2 + 2 \left(\frac{\partial f}{\partial a} \right) \left(\frac{\partial f}{\partial b} \right) (a_i - \bar{a})(b_i - \bar{b}) \right] \tag{3.6}$$

The variance of a parameter (e.g. a) is:

$$\sigma_a^2 \equiv \lim_{N\to\infty} \frac{1}{N} \sum_i (a_i - \bar{a})^2 \tag{3.7}$$

While the covariance between two statistics (e.g. a and b) is:

$$\sigma_{a,b}^2 \equiv \lim_{N\to\infty} \frac{1}{N} \sum_i (a_i - \bar{a})(b_i - \bar{b}) \tag{3.8}$$

This means that equation 3.6 can be rewritten as

$$\hat{\sigma}_f^2 = \hat{\sigma}_a^2 \left(\frac{\partial f}{\partial a} \right)^2 + \hat{\sigma}_b^2 \left(\frac{\partial f}{\partial b} \right)^2 + 2\hat{\sigma}_{a,b}^2 \left(\frac{\partial f}{\partial a} \right) \left(\frac{\partial f}{\partial b} \right) \tag{3.9}$$

This is a very important equation (often called the *error propagation equation*), because it tells you how to propagate statistical error no matter your function is. Here the hats on the variances (e.g. $\hat{\sigma}_a^2$) denote them as *estimators* of the true variances (they are also sometimes called 'sample variances'). It is impossible to measure an infinite number of data points, so we are limited to using [2]

$$\hat{\sigma}_a^2 = \frac{1}{N-1} \sum_i (a_i - \bar{a})^2, \qquad \hat{\sigma}_{a,b}^2 = \frac{1}{N-1} \sum_i (a_i - \bar{a})(b_i - \bar{b}) \tag{3.10}$$

Some common examples of error propagation for different functions of x and y (c and d are constants):

To actually make this calculation, many programs have the ability to calculate what is called a covariance matrix. This is the matrix:

[2] The $N-1$ in the denominator here is due to an important correction in statistics known as Bessel's correction. Without this correction the estimator would be *biased* and underestimate the true variance. Don't be surprised if no one cares about this correction, but it is the right way to do it. At $N = 10$, failing to use this correction will make these estimates of variance off by about 10% on average

Function	Propagation
$f = cx + dy$	$\hat{\sigma}_f = \sqrt{c^2\hat{\sigma}_x^2 + d^2\hat{\sigma}_y^2}$
$f = cx - dy$	$\hat{\sigma}_f = \sqrt{c^2\hat{\sigma}_x^2 + d^2\hat{\sigma}_y^2}$
$f = cxy$	$\hat{\sigma}_f = c\sqrt{\bar{y}^2\hat{\sigma}_x^2 + \bar{x}^2\hat{\sigma}_x^2}$
$f = cx/y$	$\hat{\sigma}_f = \frac{c}{\bar{y}}\sqrt{\hat{\sigma}_x^2 + \frac{1}{\bar{y}}\hat{\sigma}_y^2}$
$f = ce^{-dx}$	$\hat{\sigma}_f = cde^{-d\bar{x}}\hat{\sigma}_x$

$$\Sigma = \begin{bmatrix} \hat{\sigma}_{11}^2 & \hat{\sigma}_{12}^2 & \dots & \hat{\sigma}_{1m}^2 \\ \hat{\sigma}_{21}^2 & \hat{\sigma}_{22}^2 & \dots & \hat{\sigma}_{2m}^2 \\ \vdots & \vdots & \ddots & \vdots \\ \hat{\sigma}_{m1}^2 & \hat{\sigma}_{m2}^2 & \dots & \hat{\sigma}_{mm}^2 \end{bmatrix}.$$

Most of the time, this matrix will be essentially diagonal since the variables you will be looking at will be independent. Sometimes though, these off-diagonal terms will be indicative of some physics you haven't taken into account (e.g. temperature fluctuations).

■ **Example 3.2** Let's say that ℓ and κ are constants and you are calculating resistance across two elements in series using the following formula:

$$R = \ell\frac{V_1}{I_1} + \kappa\frac{V_2}{I_2} \tag{3.11}$$

To estimate the uncertainty in R at that point, one would first calculate the error propagation for the equation. To do this we find the partial derivatives:

$$\frac{\partial R}{\partial V_1} = \frac{\ell}{I_1} \qquad \frac{\partial R}{\partial I_1} = -\ell\frac{V_1}{I_1^2}$$
$$\frac{\partial R}{\partial V_1} = \frac{\kappa}{I_2} \qquad \frac{\partial R}{\partial I_1} = -\kappa\frac{V_2}{I_2^2}$$

and insert them into the error progation equation:

$$\begin{aligned}\sigma_R^2 \simeq\ & \sigma_{V_1}^2\left(\frac{\partial R}{\partial V_1}\right)^2 + \sigma_{I_1}^2\left(\frac{\partial R}{\partial I_1}\right)^2 + \sigma_{V_2}^2\left(\frac{\partial R}{\partial V_2}\right)^2 + \sigma_{I_2}^2\left(\frac{\partial R}{\partial I_2}\right)^2 \\ & + 2\sigma_{V_1,I_1}^2\left(\frac{\partial R}{\partial V_1}\right)\left(\frac{\partial R}{\partial I_1}\right) + 2\sigma_{V_2,I_1}^2\left(\frac{\partial R}{\partial V_1}\right)\left(\frac{\partial R}{\partial I_1}\right) + 2\sigma_{V_1,I_1}^2\left(\frac{\partial R}{\partial V_1}\right)\left(\frac{\partial R}{\partial I_1}\right) + 2\sigma_{V_1,I_1}^2\left(\frac{\partial R}{\partial V_1}\right)\left(\frac{\partial R}{\partial I_1}\right) \\ & + 2\sigma_{V_1,V_2}^2\left(\frac{\partial R}{\partial V_1}\right)\left(\frac{\partial R}{\partial V_2}\right) + 2\sigma_{I_1,I_2}^2\left(\frac{\partial R}{\partial I_1}\right)\left(\frac{\partial R}{\partial I_2}\right)\end{aligned}$$

This is a very large formula, but there are ways to calculate this faster using a computer. If you have made n measurements of the state vector (V_1, I_1, V_2, I_2) at the same experiment settings, then you should have your data in an $n \times 4$ matrix. If you're using MATLAB and you've named your data matrix `data`, the command to get this covariance matrix is simply `cov(data)`. In Mathematica, the command is similar: `Covariance[data]`. Though the equation that propagates the error may be huge, it can be easy to calculate all at once if you're clever about how you use your matrices.

4 — Fitting models to data

4.1 Curve-fitting concepts

At the heart of all curve-fitting and regression techniques is an optimization problem:

> *Given a set of data with the associated uncertainties and a set of assumptions about the physics of the system, what is the best mathematical model that describes the data?*

The issue with this kind of statement is understanding what "best model" means. It is important to recognize that while a model may fit the experimental data very well, it might not be very scientifically descriptive. There are two extremes in analysis that the physicist must get used to avoiding:

1. *Overcomplicating:* Because of the way the math works, it is always possible to find a model complex enough to account for all the data. For example, you could always find a fourier series or a polynomial expansion which fits every point in the dataset. This is a big mistake because the goal is not to fit the data with ad hoc assumptions that look nice but to explain the data well using physical theories and principles.
2. *Oversimplifying:* It is easy to be afraid of overcomplicating the problem and go too far in the other direction by ignoring other physical processes and opting for simple descriptions that don't use the data to their fullest potential. For example, you might be tempted cut out half of your data because they are noisy or don't fit well. This should be treated with caution since all errors are explainable and may be indicative of different physics.

A good maxim to live by is "Everything should be made as simple as possible but no simpler" (attributed to Einstein, but he only said something similar). The model you use should be justified by the physics of the experiment. It is OK to model the things you don't know much about, but this is always dangerous territory and it is important to proceed with caution.

4.2 The basics of linear least squares fitting

In the foundations of scientific analysis some conventions have emerged that are handy both practically and theoretically. For instance, one definition of "best" developed by Gauss in the 19th century is that of the least squares criterion. This is the statement that under usual conditions (Gauss-Markov conditions) the best fit of a linear model to data will be the one that minimizes the sum of squared errors. The main Gauss-Markov conditions (assumptions) that guarantee this mathematically can be summarized below:

1. The situation the model describes actually is linear. If it's not then more complicated work may be required (e.g. nonlinear least squares).

2. The random error averages to zero – the noise in the signal does not push the data in any particular direction *on average*.
3. The error or noise in the data is about the same size over the whole dataset. When this is not true the data are said to be *heteroskedastic* and the model needs to be adjusted to focus on the more precise data.
4. The noise in the data is independently distributed throughout the dataset. This means that for two data points, no amount of information about one error can tell me anything about the other.

Even when these conditions aren't strictly true, least-squares algorithms usually do very well, especially in physics where the data are often very consistent and noise isn't that bad.

■ **Example 4.1** Let's say you measure a voltage and it fluctuates around some value in some gaussian way:

$$V_i = V_0 + \varepsilon_i, \quad \varepsilon_i \sim N(\mu, \sigma_\varepsilon^2) \tag{4.1}$$

The equation above says that every measured voltage V_i is equal to some nominal voltage V_0 with some gaussian noise term ε_i. Since we can never know V_0 with absolute certainty, we must come up with an estimator for it. According to the least-squares logic this will be the $\hat{V}_0$ that minimizes the quantity *SSR* (the sum of squared residuals):

$$SSR = \sum_i \hat{\varepsilon}_i^2 = \sum_i (V_i - \hat{V}_0)^2 \tag{4.2}$$

Note that $\hat{\varepsilon}_i$ is the estimator for ε_i expressed as $\hat{\varepsilon}_i = V_i - \hat{V}_0$. To find the optimal $\hat{V}_0$, just solve the first order conditions:

$$\frac{\partial SSR}{\partial \hat{V}_0} = 0 = \sum_i -2(V_i - \hat{V}_0) \implies \hat{V}_0 = \frac{1}{N}\sum_i^N V_i \tag{4.3}$$

Which is the same as the mean of the voltage data. This confirms something we have been using all our lives – under the right conditions *the sample mean is a good estimate of the true mean*.

4.3 The linear fit (Part I): OLS regression with one dependent variable

The most typical example of an ordinary least squares (OLS) linear regression is that of the form:

$$y_i = \alpha + \beta x_i + \varepsilon_i \tag{4.4}$$

since it is the next simplest linear equation after (4.1).

4.3.1 Finding fit parameters

To find estimators $\hat{\alpha}$ and $\hat{\beta}$ of the true parameters α and β, we first find the sum of squared errors:

$$SSR = \sum_i (y_i - \hat{\alpha} - \hat{\beta} x_i)^2 \tag{4.5}$$

and then solve the first order conditions:

$$\frac{\partial SSR}{\partial \hat{\alpha}} = 0 = \sum_i -2(y_i - \hat{\alpha} - \hat{\beta} x_i) \qquad \frac{\partial SSR}{\partial \hat{\beta}} = 0 = \sum_i -2x_i(y_i - \hat{\alpha} - \hat{\beta} x_i) \tag{4.6}$$

This gives us two equations with two unknowns. If the formula were more complicated and had n unknowns, there would be n equations from first order conditions. For a linear fit like this there is *always*

an analytic expression for the parameters that minimize the *SSR*. From the first equation we find that $\hat{\alpha} = \bar{y} - \hat{\beta}\bar{x}$. Substituting to find $\hat{\beta}$ we find:

$$\begin{aligned} 0 &= \sum_i x_i(y_i - \bar{y} + \hat{\beta}\bar{x} - \hat{\beta}x_i) \\ 0 &= \sum_i x_i y_i - x_i\bar{y} + \hat{\beta}\bar{x}x_i - \hat{\beta}x_i^2 \\ \hat{\beta}\sum_i x_i^2 - \bar{x}x_i &= \sum_i x_i y_i - x_i\bar{y} \\ \hat{\beta} &= \frac{\sum_i x_i y_i - \bar{x}\bar{y}}{\sum_i x_i^2 - \bar{x}^2} \end{aligned}$$

This is how a least-squares algorithm will calculate the best fit:

$$\hat{\beta} = \frac{\sum_i x_i y_i - \bar{x}\bar{y}}{\sum_i x_i^2 - \bar{x}^2} = \frac{\mathrm{Cov(x,y)}}{\mathrm{Var(x)}} = \frac{\langle xy\rangle - \langle x\rangle\langle y\rangle}{\langle x^2\rangle - \langle x\rangle^2} \qquad \hat{\alpha} = \bar{y} - \hat{\beta}\bar{x} = \langle y\rangle - \langle x\rangle\left[\frac{\langle xy\rangle - \langle x\rangle\langle y\rangle}{\langle x^2\rangle - \langle x\rangle^2}\right] \tag{4.7}$$

4.3.2 Finding uncertainties in fit parameters

At this point it is useful to look back at the sum of squared residuals:

$$\hat{\sigma}_\varepsilon^2 = \frac{1}{N-k}\sum_i \hat{\varepsilon}_i^2 = \frac{1}{N-2}\sum_i (y_i - \hat{y})^2 \tag{4.8}$$

where the $N-k$ represents the number of *degrees of freedom* (k is the number of parameters). In this case there are $N-2$ degrees of freedom, since if we knew $N-2$ of the residuals, we could solve for the remaining two using the first-order conditions on $\hat{\alpha}$ and $\hat{\beta}$. This estimate is useful in finding the *standard error of the regression* or *root mean squared error* given by:

$$\hat{\sigma}_\varepsilon = \sqrt{\hat{\sigma}_\varepsilon^2} = \sqrt{\frac{1}{N-2}\sum_i (y_i - \hat{y})^2} \tag{4.9}$$

The standard deviation of $\hat{\beta}$ can then be found as $\hat{\sigma}_\varepsilon/\mathrm{Var}(x)$. To see this, let's look at $\hat{\beta}$ again, substituting in $y_i = \alpha + \beta x_i + \varepsilon_i$ and $\bar{y} = \alpha + \beta\bar{x} + \bar{\varepsilon}$:

$$\hat{\beta} = \frac{\sum_i x_i y_i - \bar{x}\bar{y}}{\sum_i x_i^2 - \bar{x}^2} = \frac{\sum_i x_i(\alpha + \beta x_i + \varepsilon_i) - \bar{x}(\alpha + \beta\bar{x} + \bar{\varepsilon})}{\sum_i x_i^2 - \bar{x}^2} \tag{4.10}$$

which simplifies to:

$$\hat{\beta} = \beta + \frac{\sum_i x_i\varepsilon_i - \bar{x}\bar{\varepsilon}}{\sum_i x_i^2 - \bar{x}^2} = \beta + \frac{\mathrm{Cov}(x,\varepsilon)}{\mathrm{Var}(x)} \tag{4.11}$$

Ideally, the noise ε_i isn't correlated with x_i so that on average $\mathrm{Cov}(x,\varepsilon) = 0$. In every dataset, no matter how large, the sample covariance will not be zero and $\hat{\beta} \neq \beta$. Since it *approaches* zero in the limit of large numbers, more data imply a better measure of $\hat{\beta}$. It is important to remember that β itself is *not* random, but that only the estimator $\hat{\beta}$ is distributed due the random error ε_i. At this point we are able to use the familiar error propagation equation (eq. (3.9)) to find the standard deviation of our estimator (ε_i and ε_j are independent):

$$\sigma_{\hat{\beta}} = \sqrt{\sum_i \left(\frac{\partial\hat{\beta}}{\partial\varepsilon_i}\right)^2 \sigma_{\varepsilon_i}^2} \tag{4.12}$$

To do this we must obtain the partial derivative:

$$\frac{\partial\hat{\beta}}{\partial\varepsilon_i} = 0 + \frac{\partial}{\partial\varepsilon_i}\frac{\sum_i x_i\varepsilon_i - \bar{x}\bar{\varepsilon}}{\sum_i x_i^2 - \bar{x}^2} = \frac{\partial}{\partial\varepsilon_i}\frac{\sum_i \varepsilon_i(x_i-\bar{x})}{\sum_i x_i^2 - \bar{x}^2} = \frac{x_i - \bar{x}}{\sum_i x_i^2 - \bar{x}^2} \tag{4.13}$$

Of course, in the sum *this* is where the facts that the error is identically distributed and that ε_i and ε_j are independent become very important:

$$\sigma_{\hat{\beta}} = \sqrt{\sigma_\varepsilon^2 \frac{\sum_i (x_i-\bar{x})^2}{(\sum_i x_i^2 - \bar{x}^2)^2}} = \frac{\sigma_\varepsilon}{\sum_i (x_i^2-\bar{x}^2)}\sqrt{\sum_i (x_i-\bar{x})^2} \tag{4.14}$$

$$= \frac{\sigma_{\varepsilon_i}}{\sqrt{\sum_i (x_i^2-\bar{x}^2)}} \tag{4.15}$$

Again, we can never know σ_ε exactly. The best we can do is estimate (use the estimator $\hat{\sigma}_\varepsilon$):

The uncertainty in $\sigma_{\hat{\beta}}$ is therefore:

$$\hat{\sigma}_{\hat{\beta}} = \frac{\hat{\sigma}_{\hat{\varepsilon}_i}}{\sqrt{\sum_i (x_i^2-\bar{x}^2)}} = \frac{\sqrt{\frac{1}{N-2}\sum_i (y_i-\hat{y})^2}}{\sqrt{\sum_i (x_i^2-\bar{x}^2)}} \tag{4.16}$$

The uncertainty in $\hat{\alpha}$ can then be found by propagating the error in $\hat{\alpha} = \bar{y} - \hat{\beta}\bar{x}$:

$$\hat{\sigma}_{\hat{\alpha}} = \sqrt{\left(\frac{\partial\hat{\alpha}}{\partial\hat{\beta}}\right)^2 \sigma_{\hat{\beta}}^2} = \bar{x}\hat{\sigma}_{\hat{\beta}} \tag{4.17}$$

4.4 The linear fit (Part II): using error bars in a linear fit (WLS or χ^2 regression)

So far, in using the least-squares method we've treated all datapoints as if they were of equal importance. When you do an experiment and know some of the data points to higher precision than others, there is a way to incorporate this into the analysis using something called weighted least squares regression (WLS regression). Mathematically, this is the same as saying that at x_i the error ε_i has a true standard deviation σ_i, but at x_j the error ε_j has a true standard deviation of σ_j where $\sigma_i \neq \sigma_j$. Now it isn't true that the ε_i are identically distributed. *When the error term is not identically distributed (your error bars aren't all the same size – i.e. heteroskedasticity), the OLS estimators will be* **unbiased** *but they* **will not be minimum variance. That means that if you use the method in the last section, your estimates for the parameters α and β will be as accurate but less precise on average.**

To correct for this problem, instead of minimizing the sum of squared residuals (*SSR*), we instead minimize a quantity called χ^2;

$$\chi^2(\hat{\alpha},\hat{\beta}) = \sum_i \left(\frac{y_i - \hat{y}_i}{\hat{\sigma}_i}\right)^2 = \sum_i \left(\frac{y_i - (\hat{\alpha}+\hat{\beta}x_i)}{\hat{\sigma}_i}\right)^2 \tag{4.18}$$

Here, the $\hat{\sigma}_i$ are the values of uncertainty calculated from the characteristics of the apparatus (the error bars). This is the same as saying that the data points with low uncertainty $\hat{\sigma}_i$ (higher precision) are weighted more highly in the fit than those with higher uncertainty (in a specific way). When the error ε_i is corrected by σ_i, the error $\varepsilon_i^* = \varepsilon_i/\sigma_i$ is identically distributed and the problem is the same as before.

In the last section we went manually through the calculation to find the parameters $\hat{\alpha}$ and $\hat{\beta}$. In this section we'll use matrices which can make solving this problem a little bit easier on the computer. WLS regression is exactly like OLS regression except each of the points is weighted.

4.4.1 Matrices help to make solving fits faster (especially WLS)

Again, as before, minimizing χ^2 is like minimizing *SSR* only the equation looks beefier since we have weighted the terms according to their precision:

$$\frac{\partial\chi^2}{\partial\hat{\alpha}} = 0 = \sum_i \frac{-2(y_i - \hat{\alpha} - \hat{\beta}x_i)}{\hat{\sigma}_i^2} \qquad \frac{\partial\chi^2}{\partial\hat{\beta}} = 0 = \sum_i \frac{-2x_i(y_i - \hat{\alpha} - \hat{\beta}x_i)}{\hat{\sigma}_i^2} \tag{4.19}$$

This leaves us with two equations and two unknowns (again) :

$$\sum_i \frac{y_i}{\hat{\sigma}_i^2} = \hat{\alpha}\sum_i \frac{1}{\hat{\sigma}_i^2} + \hat{\beta}\sum_i \frac{x_i}{\hat{\sigma}_i^2} \qquad \sum_i \frac{x_i y_i}{\hat{\sigma}_i^2} = \hat{\alpha}\sum_i \frac{x_i}{\hat{\sigma}_i^2} + \hat{\beta}\sum_i \frac{x_i^2}{\hat{\sigma}_i^2} \tag{4.20}$$

This can be put into a matrix equation as:

$$\underbrace{\begin{bmatrix} \sum_i \frac{y_i}{\hat{\sigma}_i^2} \\ \sum_i \frac{x_i y_i}{\hat{\sigma}_i^2} \end{bmatrix}}_{\vec{c}} = \underbrace{\begin{bmatrix} \sum_i \frac{1}{\hat{\sigma}_i^2} & \sum_i \frac{x_i}{\hat{\sigma}_i^2} \\ \sum_i \frac{x_i}{\hat{\sigma}_i^2} & \sum_i \frac{x_i^2}{\hat{\sigma}_i^2} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \hat{\alpha} \\ \hat{\beta} \end{bmatrix}}_{\vec{m}} \tag{4.21}$$

This matrix is in the form of $A\vec{m} = \vec{c}$. Here, $\vec{m}$ denotes the vector of model parameters , and the vector $\vec{c}$ and matrix A represent constraints derived from first order conditions,. It is extremely unlikely that the matrix A will not be invertible, so the solution is simply $A^{-1}\vec{c} = \vec{m}$. Creating these vectors and matrices may seem simple, but a *really* fast way of getting these matrices is to use a matrix representing the data used in the regression:

$$X_{OLS} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \qquad X_{WLS} = \vec{w} \odot X_{OLS} = \begin{bmatrix} 1/\sigma_1 & x_1/\sigma_1 \\ 1/\sigma_2 & x_2/\sigma_2 \\ \vdots & \vdots \\ 1/\sigma_n & x_n/\sigma_n \end{bmatrix} \tag{4.22}$$

Here $\odot$ represents element-wise multiplication. In MATLAB for instance, multiplying vectors or matrices (`A` and `B`) can be done element-wise with the command `A.*B`. When you use these matrices, it is very easy to construct $\vec{c}$ and $\vec{A}$:

$$\vec{c} = X^\intercal \vec{y} \qquad A = X^\intercal X \tag{4.23}$$

In summary, χ^2 or WLS fitting uses exactly the same mathematics as OLS fitting, except that weighting each point makes calculating fit parameters and uncertainties involve more operations. **In general, you should use the χ^2 method of least-squares fitting because it is almost always true that the error bars you use are not all the same size. If they are, it turns out that the χ^2 method will give you the same result as the OLS method.** This is actually very easily provable:

$$\hat{\sigma}_i = \hat{\sigma}\forall i \implies \chi^2 = \sum_i \frac{(y_i - \hat{y}_i)^2}{\hat{\sigma}_i^2} = \frac{1}{\hat{\sigma}^2}\sum_i (y_i - \hat{y}_i)^2 = \frac{SSR}{\hat{\sigma}^2} \tag{4.24}$$

Since $\hat{\sigma}_i$ is the same size throughout the dataset, it can be factored out of the sum, yielding the sum of squared residuals, the function to be minimized in OLS fitting. Dividing the *SSR* by a constant does not change where the minimum is located, so the parameters you solve for must be the same.

4.4.2 Uncertainties in χ^2 fitting

The uncertainty in fit parameters can be found in the χ^2 fitting the same way as in the OLS case: that is, we must propagate error. In the interest of brevity, the derivation will not be shown here. Using matrices, a pattern emerges which is quite convenient (which you can derive if you want). As it turns out, the matrix we already calculated from first-order conditions (A^{-1}) contains all the information necessary to propagate error.

The model parameters are given in order in a vector by $\vec{m} = A^{-1}\vec{c} = (X^\intercal X)^{-1}X^\intercal\vec{y}$. With the residuals defined as the difference between the data and the model ($\vec{\varepsilon} = \vec{y} - X\vec{m}$), the uncertainty in each parameter is obtained as:

$$\hat{\sigma}_i = \hat{\sigma}_\varepsilon\sqrt{(A^{-1})_{i,i}} \qquad \text{where } \hat{\sigma}_\varepsilon^2 = \sum_i \frac{\hat{\varepsilon}^2}{N-k} = \sum_i \frac{(y_i - \hat{y}_i)^2}{N-k} \tag{4.25}$$

Here the $\hat{\sigma}_i$ is the estimated uncertainty in the i-th parameter ($\hat{\sigma}_\alpha = \hat{\sigma}_1$ and $\hat{\sigma}_\beta = \hat{\sigma}_2$), the element $(A^{-1})_{i,i}$ is the i-th element on the diagonal of the matrix A^{-1}.

4.5 Nonlinear fitting

When you fit a curve to data – no matter how many linear variables you put in the formula – there is an analytic expression for the fit parameters that minimizes the sum of squared errors (sum of squared residuals *SSR* or χ^2 in the weighted case). This must always be true, since when you have linear combinations of parameters, the first-order conditions must generate a set of n equations with n unknowns.

However, when you have a *nonlinear* model, the first-order condition equations will most often not yield closed solutions. As an example let's say you have the model $y_i = \alpha e^{-\beta x_i} + \varepsilon_i$. When you solve first-order conditions, you will find that they are identical:

$$\frac{\partial\chi^2}{\partial\hat{\alpha}} = \sum_i \frac{(y_i - \hat{\alpha}e^{-\hat{\beta}x_i})e^{-\hat{\beta}x_i}}{\sigma_i^2} = \frac{\partial\chi^2}{\partial\hat{\beta}} \tag{4.26}$$

What this means about nonlinear fitting is that it should be done numerically with the aid of a fitting algorithm. These are simply algorithms that attempt to minimize the χ^2 or *SSR* by looking for the best fit parameters. Because these sums of squares depend very complexly on the parameters, it is possible for these algorithms to diverge or stall out in a variety of ways.

4.5.1 Nonlinear fitting guidelines

Even when you have the right model, it is possible for your curve-fit to fail because you haven't given the computer enough the right information about where to look for a minimized χ^2 value. Let's say you have a peak in some data that you would like to fit to a gaussian, and you are using the model $y = a_1 e^{-((x-b_1)/c_1)^2}$ or $\texttt{f(x)} = \texttt{a1}*\texttt{exp}(-((\texttt{x}-\texttt{b1})/\texttt{c1})\texttt{\^{}2})$ in MATLAB. For the data in figure 4.1, if you choose starting points for these parameters of $\texttt{a1} = 1, \texttt{b1} = 0, \texttt{c1} = 1$, you will get a bad result because the algorithm has not been looking in the right region for the minimum of χ^2.

```
General model Gauss1:
     f(x) =  a1*exp(-((x-b1)/c1)^2)
Coefficients (with 95% confidence bounds):
       a1 =     4.02550e+69
       b1 =     0.00000
```

```
       c1 =     1.00000

Goodness of fit:
  SSE: 1.048e+06
  R-square: -0.913
  Adjusted R-square: -0.9944
  RMSE: 149.3

  Warning: A negative R-square is possible if the model
           does not contain a constant term and the fit
           is poor (worse than just fitting the mean).
           Try changing the model or using a different StartPoint.
```

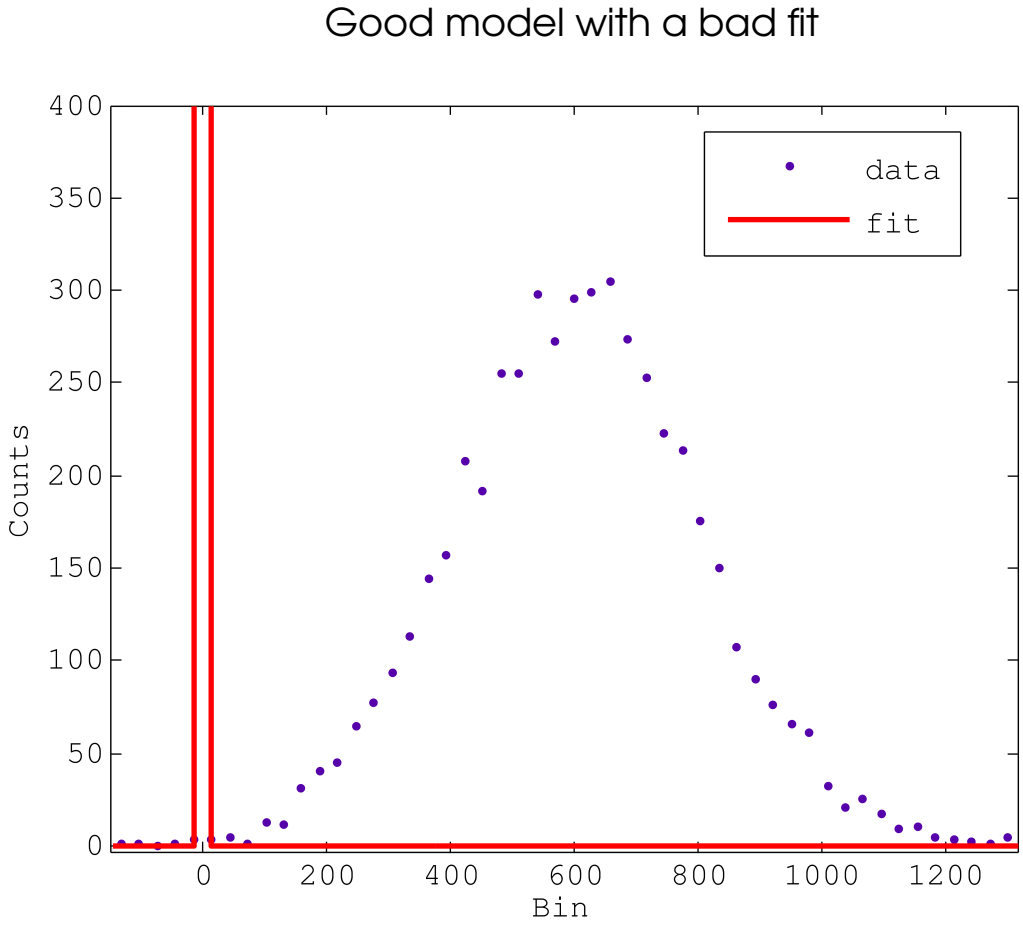


Figure 4.1: Fitting gaussian data generated on a computer with the function $\texttt{f(x)} = \texttt{a1} * \texttt{exp}(-((\texttt{x}-\texttt{b1})/\texttt{c1})\texttt{\^{}2})$ in MATLAB. Choosing initial conditions of $\texttt{a1}=1, \texttt{b1}=0, \texttt{c1}=1$, the computer algorithm that tries to minimize χ^2 diverges.

To fix this problem, and *in general whenever you start any nonlinear fit*, it is a good idea to calculate a ballpark estimate of where the parameters should end up. In the example above, we know that the peak

of the gaussian function should be about 300 high and located somewhere around 600 with a standard deviation on the order of 200. If we just start with parameters decently close to their final outcome, e.g. $\texttt{a1} = 300, \texttt{b1} = 600, \texttt{c1} = 200$, we get the following fit output:

```
General model Gauss1:
     f(x) =  a1*exp(-((x-b1)/c1)^2)
Coefficients (with 95% confidence bounds):
       a1 =   296.88732  (290.61741, 303.15724)
       b1 =   603.50693  (598.73503, 608.27883)
       c1 =   276.73864  (269.98996, 283.48732)

Goodness of fit:
  SSE: 3605
  R-square: 0.9934
  Adjusted R-square: 0.9931
  RMSE: 8.758
```

And the plot looks *much* better:

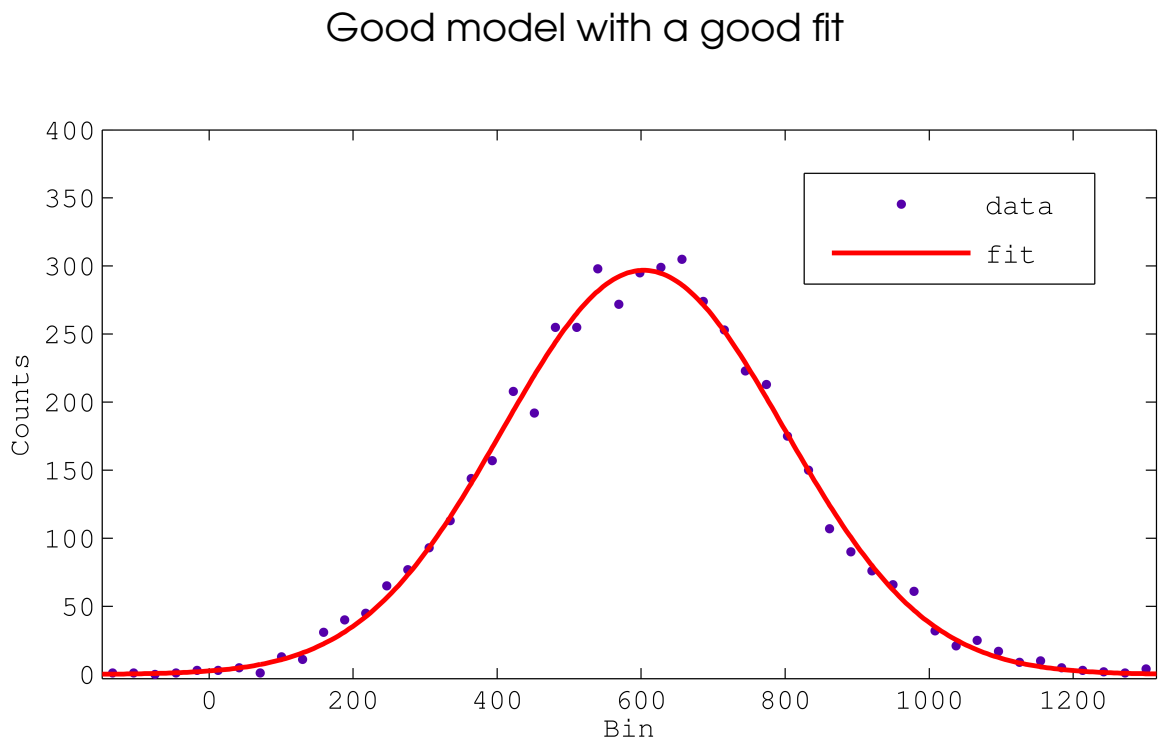


Figure 4.2: Fitting gaussian data generated on a computer with the function $\texttt{f(x)} = \texttt{a1} * \texttt{exp}(-((\texttt{x}-\texttt{b1})/\texttt{c1})\texttt{\^{}2})$ in MATLAB. Choosing initial conditions of $\texttt{a1} = 300, \texttt{b1} = 600, \texttt{c1} = 200$, the computer algorithm that tries to minimize χ^2 settles to a good estimate.

Now that we have a good fit, how can we use the fit output in a physics paper or report? The output that is produced is in the confidence interval format, which is a perfectly reasonable format to quote. For example, you could say:

"After fitting the data with a gaussian using a trust-region nonlinear least squares algorithm (default in MATLAB), we found the peak parameter b converged to a value of 604 within a 95% confidence interval (599, 608)."

How do we decide what significant figures to use when the computer, when run long enough, could calculate 95% confidence intervals out to 500 decimal places? It's not the lowest number of significant figures in the data, because the point of a fit is to reduce uncertainty by means of statistical analysis. The answer is that the confidence interval `(598.73503, 608.27883)`, as a measure of uncertainty, determines to what extent the parameter value `603.50693` is useful. The confidence interval can be expressed alternatively as `603.50693 (+ 5.7719, - 4.7719)`, at which point it becomes clear that everything after the ones' place is not useful (the uncertainty is of order 10^0. Now to get to an answer in the usual format of $\hat{\mu} \pm \hat{\sigma}$ must first convert to 68% confidence (1σ) intervals by dividing the deviations by 1.96. This is because a 95% of a normal distribution lies between -1.96σ and $+1.96\sigma$. So, enforcing significant figures, the value of `604 ± 5` at the 95% confidence level becomes `604 ± 3` at the standard 68% confidence level.

In summary, if you are having problems with fitting, there are really only two things that can go wrong: either you're not looking for a minimized χ^2 in the right place or you're using the wrong model.

4.6 Evaluating Fits – reduced χ^2

If you don't have the right model, how do you know in what way it's wrong? Well, there are really only two ways in which a model can be wrong.

1. **The model oversimplifies the experiment.** When you haven't taken all of the physics into account, and your model at best only roughly approximates the data you've collected, there will likely be no combination of parameters that fit the data very well.
2. **The model overcomplicates the experiment.** When you have taken the physics of the experiment into account, and use extra parameters to describe the data, you will likely find a few combinations of parameters that work to describe the data very well. This means that the complexity of your model is *artificial* and does not reflect reality.

To see whether the model is appropriate, physicists use a tool called reduced chi-squared (χ^2_{red}). This is a quantity which represents the ratio of the estimated uncertainty in the model fit to the estimated uncertainty from experiment. Recall that χ^2 was defined earlier as:

$$\chi^2 = \sum_i \left(\frac{y_i - \hat{y}_i}{\hat{\sigma}_i} \right)^2 \tag{4.27}$$

Since $y_i - \hat{y}_i$ is the deviation of the data point y_i from the model fit $\hat{y}_i$, and $\hat{\sigma}_i$ is the uncertainty calculated for the experiment at that data point, the contribution of any datapoint to χ^2 is near one when the deviation of the model is about the same size as the uncertainty. What this means is that if we divide χ^2 by the number of datapoints and get something close to one, then we can say that our model is good so long as we are confident in our estimate of uncertainty. Just like with Bessel's correction of the estimated standard deviation however (eq. 3.10), we can't simply divide by the number of data points we have – we have to divide by the *degrees of freedom* $\nu = N - k - 1$.

$$\chi^2_{\text{red}} = \frac{1}{\nu} \sum_i \left(\frac{y_i - \hat{y}_i}{\hat{\sigma}_i} \right)^2 = \frac{\chi^2}{\nu} \tag{4.28}$$

As before, N is the number of data points and k is the number of parameters in the model. Of course, if $k+1 \ll N$, then this correction doesn't make much difference (it is however formally correct). When we accept χ^2_{red} as a good estimate of the ratio of model deviation to expected deviation, we can evaluate whether the model is good by examining the following cases:

1. $\chi^2_{\text{red}} \gg 1$. When the reduced chi-squared of a fit is much larger than one, the only possible ways for this to happen are for:
   (a) the model to fit so poorly that $(y_i - \hat{y}_i)^2 \gg (\hat{\sigma}_i)^2$ on average.
   (b) the uncertainty $\hat{\sigma}_i$ to be so underestimated that $(y_i - \hat{y}_i)^2 \gg (\hat{\sigma}_i)^2$ on average, *even though* the model fits fairly well. This is only practically possible if you think you have much more precision on your individual measurements than you *actually* do.
2. $\chi^2_{\text{red}} < 1$. When the reduced chi-squared of a model fit is less than one, the only way for this to happen is for the error in your fit to be overestimated so that $(y_i - \hat{y}_i)^2 < (\hat{\sigma}_i)^2$ on average.

5 — Mathematica, MATLAB, and Python

5.1 Computational tools in physics

The next three chapters will be on the basic features of Mathematica, MATLAB and Python. These are three of the most common platforms broadly used for data analysis in physics. There certainly are other platforms that physicists use, but they are most often used in particular subfields (e.g. ROOT for high energy physics) even if they are broadly applicable. This chapter will serve as a guide to Mathematica, MATLAB, and Python because those platforms are available either in the PMCL and via student license (Mathematica and MATLAB) or can be downloaded for free (Python).

Feature	Mathematica	MATLAB	Python
Syntax	Designed to be user-friendly and 'like english', but non-conventional and maybe awkward at first	Inflexible but robust and easy to use	More complicated (it's actually a programming language) but relatively straightforward.
Purpose	All-encompassing primarily symbolic platform for mathematical sciences	All-encompassing primarily numerical platform for experimental and engineering applications	A fast interpretted language with broad functionality and a relatively shallow learning curve.
Coverage	Covers most analysis techniques but is somewhat weak and the error messages can be difficult to interpret.	Without the toolboxes, it is only useful if you download scripts or are willing to write them yourself. With the toolboxes ($$), it is excellent.	Covers everything that you would want to do with data analysis, but requires an understanding of the language to use.

5.2 Learning these programs fast

The eternal rule of learning to use any technology is

Learn to use the documentation

Learning how to use the documentation in the program or device you are using is essential to making progress with it. Reading about the platform is often the only alternative to guessing. Unfortunately

`Sample0001.txt` is a plain text file of numbers that can be downloaded from the SPS website. The first column is time in some units. The rest of the columns have the x,y, and z components of the position of a few particles from a simulation. In this exercise you will preform various calculations, plots and analysis of this data.

1. Load the data from `Sample0001.txt` into a matrix
2. Plot a few of the columns of data as a function of column 1 (the time).
3. Compute the mean and standard deviation of all the columns after the first.
4. Are the results for the x, y and z coordinates different? Are the results for the different particles different? Answer this question quantitatively.
5. Make a histogram plot of columns 2 and 5. What is the connection between these plots and question 2?
6. Make a parametric plot of columns 2 and 3 versus the time. The points in your plot should be connected by lines.
7. Fit the plot from question (4) to a Gaussian function. Your can check your fit parameters since they should have an obvious connection to the results from (2). Is a Gaussian a good fitting function?
8. Your time series plots from question 1 have some clear oscillations. Take the Fourier transform of this data and plot it.
9. Suggest what physical situation the data is computed for.
10. Do some other operation on the data of your choosing. For example, are there any correlations between the position coordinates for particles 1 and 2?
11. Merit badge question: the parametric plot is messy because there are a lot of data points. Plot the data such that the first batch of points is one color, the next batch is another, etc. A half dozen colors should be enough: move across the spectrum, red to blue, as you plot the data.

6 — Using Mathematica for Data Analysis

Besides being a great program for doing complicated algebraic calculations and working the calculus we're too lazy or pressed for time to slog through, Mathematica is a great program for basic plotting and fitting, and has many built in features which can instantly render figures and tables for use in a paper or lab report.

6.1 Running commands

If you've never used Mathematica ever before in your life, it may be useful to read this section.

Mathematica is structured so as to act like a notebook you write down your equations in and manipulate your theoretical ideas. To make the feel of this program work, the software does not run like an ordinary scripting language. Instead of writing a program and executing a file, Mathematica evaluates one line at a time. This has some advantages, as you can change everything on the fly, but it may be confusing at first if you've never seen it before or are expecting to "compile" the whole program.

6.1.1 Conventions

When entering commands in Mathematica, a few command conventions hold:

1. Commands always begin with capital letters and use square brackets (`[]`) are used for command arguments or as locations in a matrix when used twice (e.g `[[]]`)

   ■ **Example 6.1** Function arguments: `ListPlot[]`, `Table[]`

   ■ **Example 6.2** Location in a matrix or "table": `A[[1,2]]` refers to the element in the first row and second column of matrix `A`

2. Parentheses are used to group terms in an equation or an expression and are *not* used in functions or denoting function arguments at all.
3. Greek letters like α, β, γ etc. are obtained by sandwiching the corresponding roman letters between escape characters. For instance, press escape then the letter `a` followed by escape again – you should get α.

6.2 Loading data

Two common ways to import data are the `Import["file.txt","Format"]` and the `ReadList["file.txt","Format"]` functions. On the computers in the PMCL, Mathematica will look specifically in your Documents folder for the data file you specify. One way to check whether this is true (it *might not be*) is to execute the command `Directory[]`: this command will tell you where Mathematica is looking for files. However, if you give Mathematica the full file path, it will know exactly where to go.

Once your data files are in this folder (e.g. `"C:\Users\`*uteid*`\Documents"`), you can simply use `Import[]` and `ReadList[]` without providing the full file address. This is a fast way to work if you like to download your files off of your email and continually send the updated versions to yourself. If you have a flashdrive however, a much more efficient solution can be found in useful tip 6.1

■ **Example 6.3** `rawdata = Import["Filewithdata.txt", "Table"];`

■ **Example 6.4** `rawdata = ReadList["Filewithdata.txt", Number, RecordLists→True];`

Useful Tip 6.1 — Probably the most convenient solution to uploading data.
If you would like to change directories, for instance if you like to work off of a flash drive, you can quickly access files that are in the same folder as your notebook by using the `NotebookDirectory[]` command. The way to do this is by *concatenating* the notebook directory with the file name you're interested by using the <> between `NotebookDirectory[]` and `"Filewithdata.txt"`. The string you give Mathematica is then a complete filepath description of where the file is, and so this is very robust and useful. For example:

■ **Example 6.5** If you have both your data file and your notebook file in a folder on a flash drive, let's call it `Flashdrive Folder`, and you want to import a text file with only comma-separated or tab-delimited data, you can use the following command and obtain a matrix with all of the data:

```
rawdata = Import[NotebookDirectory[] <> "Sample0001.txt", "Table"];
```

Here, `NotebookDirectory[] <> "Sample0001.txt"` retrieves the string for the filepath `I:\Flashdrive Folder\Sample0001.txt`. **This means that every time you move** `Flashdrive Folder`**, Mathematica will be able to find the data files and you won't have any problems.**

If the file contains non-number (strings) in any of the columns, it is more useful to use the `Readlist[]` command as you can specify which columns don't contain numerical data. Let's say you have three columns in your `.txt` file where your first column specifies the date and the second and third contain numerical data. By putting the list `{Word, Number, Number}` in as an option, Mathematica knows that the data in the first column are words and not numbers:

```
rawdata =
ReadList[NotebookDirectory[] <> "Sample0001.txt", {Word, Number, Number}];
```

In general, if you just want to input a datafile with only a matrix matrix, use:

```
rawdata =
ReadList[NotebookDirectory[]<>"Sample0001.txt", Number, RecordLists→True];
```

6.3 Plotting numerical data

To plot two columns of numerical data against eachother in Mathematica, you can't just give Mathematica two lists of data to plot. This is because the program likes to think of plotting numbers in terms of $\{x,y\}$ pairs. Instead, if you have a matrix (`data`) with columns of numerical values imported correctly, you will have to construct a list of pairs. Once you get the hang of this, it can become reasonably intuitive. If you're used to Excel or MATLAB, it might be confusing at first though.

■ **Example 6.6** To plot data in column 1 against data in column 2 of an imported dataset `data`, use:

```
ListPlot[data[[All,{1,2}]], AxesLabel → {"Column 1", "Column 2"},
ImageSize → Large]
```

At the end of the command, `AxesLabel` and `ImageSize` are *options* that Mathematica has built-in settings for. If you are ever confused about what these options do, search the documentation for the function you're using and click on "details" on that page. This should tell you everything you need to know about what the function can do.

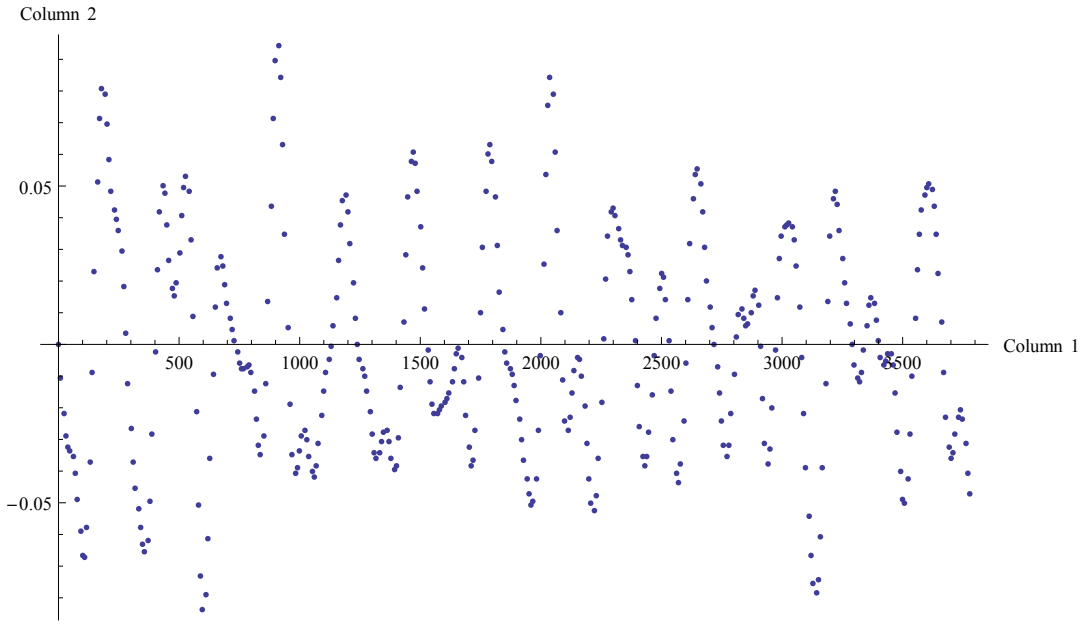


Figure 6.1: A `ListPlot[]` of sample data. One thing you may notice about this plot is that it is rather cluttered and doesn't convey the fact that it is a timeseries very well. Also, the axes font isn't very readable at this scale.

To make the plot more readable, there are many options one can use in `ListPlot[]` or `ListLinePlot[]`. Some common options include:

1. **Making axes font bigger:** `BaseStyle → {FontSize → 20}`, or whatever pt font you would like to use. The `BaseStyle` option includes ways of making the fonts bold or italic as well.
2. **Titling plots:** `PlotLabel → "Title"` `PlotLabel → Style["Title", "Style"]`. Mathematica has a lot of style formats like " Graphics" and "Section" which mimic Mathematica's typsetting features. These can look pretty good at times, but it's a good habit to use LaTeX for plot titles if you can get that to work.
3. **Adding grid lines:** `GridLines → Automatic`. This puts grid lines on the plot in a basic unassuming way. To make the lines dashed use `GridLineStyle → Dashed`. You can also specify where the grid lines are and change the color.

```
In[3]:= ListLinePlot[data[[All, {1, 2}]], AxesLabel → {"Column 1", "Column 2"},
 ImageSize → Large, PlotLabel → Style["Column 1 vs. Column 2", "Section"],
 BaseStyle → {FontSize → 20}, GridLines → Automatic, GridLinesStyle → Dashed]
```

Out[3]=

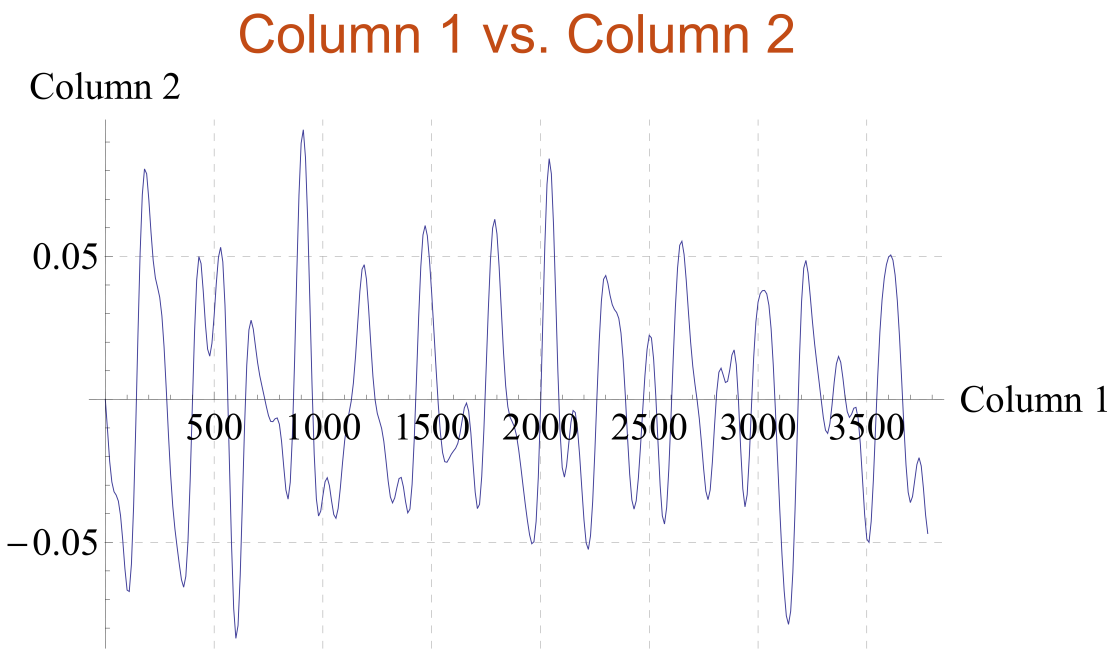


Figure 6.2: As you can see, this plot is more easy to look at and showcases the basic plotting options.

6.3.1 Histograms

Making histograms is very straightforward using the `Histogram[`*list*`,` *bins*`]` command

```
In[9]:= Histogram[data[[All, 2]], 30]
```

Out[9]=

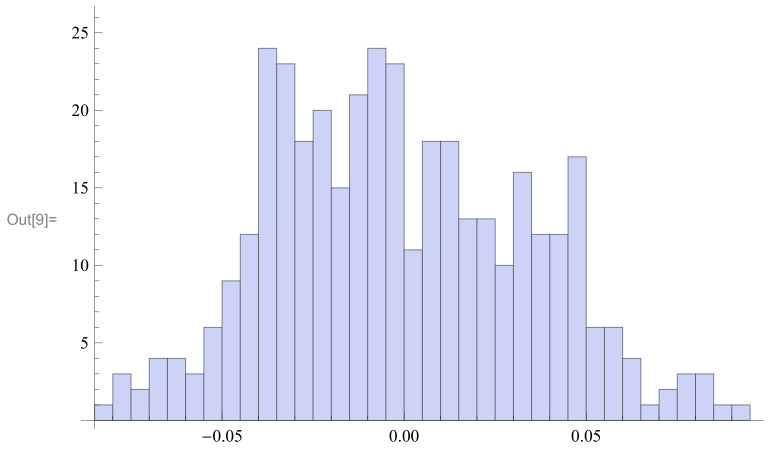


Figure 6.3: A histogram of a column of data

6.4 Tables and Grids

In mathematica, it is fairly straightforward to make a table of data with formatted headers. What mathematica needs to see is a matrix or list-of-lists of some form that can be put into a grid. In the example below, the code creates a column of means and a column of standard deviations for 18 columns of data, and then puts this information into a table with labelled columns and rows using the `TableForm[]` command:

```
In[4]:= muandsigma = Table[{Mean[data[[All, i]]], StandardDeviation[data[[All, i]]]},
          {i, 2, Length[data[[1]]]}];
        rowheadings = Table["Column " <> ToString[i], {i, 2, 19}];
        colheadings = {"Mean", "Standard Deviation"};
        ScientificForm[
         TableForm[muandsigma, TableHeadings → {rowheadings, colheadings}], 4]
```

Out[7]//ScientificForm=

	Mean	Standard Deviation
Column 2	-1.29×10^{-3}	3.51×10^{-2}
Column 3	-4.723×10^{-5}	2.3×10^{-2}
Column 4	-1.995×10^{-3}	2.873×10^{-2}
Column 5	7.679	3.677×10^{-2}
Column 6	2.282×10^{-4}	3.162×10^{-2}
Column 7	-2.192×10^{-3}	3.514×10^{-2}
Column 8	-1.095×10^{-4}	3.116×10^{-2}
Column 9	5.439	2.17×10^{-2}
Column 10	-1.913×10^{-4}	2.342×10^{-2}
Column 11	7.681	3.041×10^{-2}
Column 12	5.44	2.267×10^{-2}
Column 13	1.164×10^{-4}	2.516×10^{-2}
Column 14	1.92	2.652×10^{-2}
Column 15	9.52	1.886×10^{-2}
Column 16	-1.358	2.841×10^{-2}
Column 17	8.781×10^{-4}	2.663×10^{-2}
Column 18	8.16	1.762×10^{-2}
Column 19	-2.718	1.652×10^{-2}

6.5 Fitting arbitrary functions to data

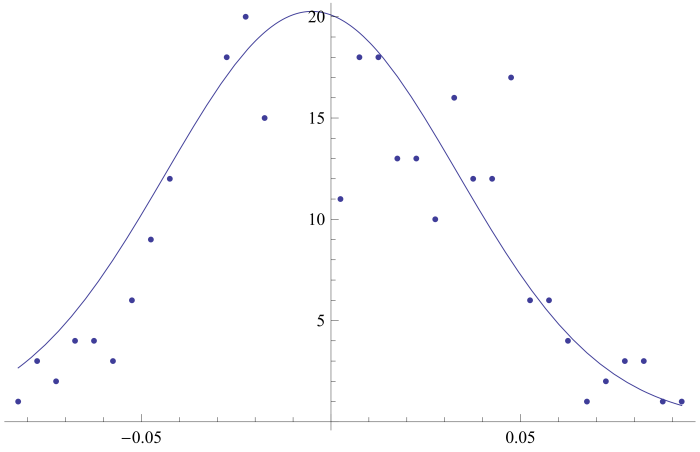
If you have some data and you are interested in fitting it with some nonlinear function, mathematica can do that without having to install a toolbox like MATLAB requires. The way to do this is to take some data and use the `NonlinearModelFit[]` command. In the example below, the data used in figure 6.3 is fit with a gaussian function. Because mathematica outputs a set of histogram bin edge positions instead of bin centers, the first four lines of the example are used to prepare a dataset that can be fit with a gaussian.

```
In[10]:= h = HistogramList[data[[All, 2]], 30];
         centers = MovingAverage[h[[1, All]], 2];
         counts = h[[2, All]];
         hd = Table[{centers[[i]], counts[[i]]}, {i, 1, Length[counts]}];
         gaussian = a Exp[- (b - x) ^2 / c^2];
         fit = NonlinearModelFit[hd, gaussian, {a, b, c}, x, ConfidenceLevel → 0.68]
         Show[Plot[fit["BestFit"], {x, First[centers], Last[centers]}], ListPlot[hd]]

         TableForm[{fit["BestFitParameters"], fit["ParameterConfidenceIntervals"]},
          TableHeadings → {{"Parameter", "Confidence Interval"}}]
```

Out[15]= FittedModel[$20.263\, e^{-338.622 \ll 1 \gg^2}$]

Out[16]=



Out[17]//TableForm=

Parameter	a → 20.263	b → -0.00507746	c → 0.0543429
Confidence Interval	18.9639 21.5621	-0.00792057 -0.00223435	0.0501713 0.0585145

This example again makes use of the `TableForm[]` command, useful for bundling the parameters and confidence interval informations into a table for easy reading. The general idea here is that `NonlinearModelFit[]` returns (creates) an object we decided to call `fit`.

Useful Tip 6.2 — Interpretting this data: why `ConfidenceLevel` is important.
The parameters and confidence intervals of this model have been estimated using a nonlinear least-squares algorithm. This means that the uncertainties in the parameters have already been found, assuming that the noise in the data is uniformly distributed. So, if we want to read off uncertainties in the standard format of $\hat{x} \pm \hat{\sigma}_x$, all we need to do to find $\hat{\sigma}_x$ is divide the width of the confidence interval by two (the confidence interval takes up $2\hat{\sigma}$). If `ConfidenceLevel→ 0.95`, then the confidence interval would take up about $4\hat{\sigma}$ instead. This comes from the standard normal distribution.

7 — Using MATLAB for Data Analysis

MATLAB stands for "MATrix LABoratory", which means that the program is on the one hand *extremely* good at working with matrices and on the other *somewhat limited* to those problems that can be solved using matrices. Naturally, if you spend some money and buy extra toolboxes it can do somewhat more elaborate calculations. In particular, the curve fitting toolbox is excellent and can do most if not all of the data analysis you might encounter in PHY 353L or PHY 474. In general though, if you are thinking about a problem and you can think of a way to quickly solve it using matrices and vectors, MATLAB is a good option.

If you get MATLAB and think you might also buy the curve-fitting toolbox, you should get it at the same time as the main program since Mathworks updates their product very frequently (usually about twice per year), and if you have an academic license you can only get the most recent editions of the toolboxes online.

7.1 Introduction

Since MATLAB is designed to work predominantly with lab data, it has many features which are tailored specifically to common tasks engineers and experimentalists do on a regular basis. For instance, the program has a file organizing system which can be viewed at any time and graphical user interfaces for common tasks like curve-fitting and importing data. Mathematica, on the other hand, is designed to encompass a much wider range of problems and so often falls short of MATLAB when it comes to convenience and ease of use in a laboratory setting.

7.1.1 Scripts – what makes MATLAB so popular

When you want to do anything in MATLAB, you will always either use the buttons in the menus, or you will execute expressions through the 'Command Window'. Because the expressions you might want to execute might be complicated and long, MATLAB has a file type ('.m') that it interprets as a long chain of commands. This is called a *script* and is used widely in the MATLAB community to do a wide variety of computations. When you open MATLAB you will see a few open 'panes' in the program.

- **Current Folder** – This is where MATLAB looks for anything new (data, functions, scripts). If you're trying to load data make sure the 'Current Folder' is looking at the file you're trying to use.
- **Editor** – This is where you can edit and create new scripts and functions.

- **Workspace** – This is the pane that displays all variables and their basic characteristics. This is very useful for checking up on variable characteristics.
- **Command Window** – This is where expressions are executed. The most common things are functions, commands, and scripts.
- **Command History** – This is where past expressions are stored so that you can access them easily. One very useful trick while coding in MATLAB is to press the 'up' key on the keyboard to access the last command so you can edit it without having to write it out again.

The easiest ways to make a new script are (1) to press `Ctrl+N` while in the editor or (2) right-click on the Current Folder window and choose the 'New File → Script' option. Once you have a `filename.m` file up, you can execute the entire file by simply typing `filename` in the command window.

7.1.2 Syntax & functions

In MATLAB, commands & functions are executed from the 'command window'. Functions are of the form `functionName(`*arguments*`)` in which case they call the `functionName` *function*; commands are of the form `command` *argument*, in which case they run the `command` *command*. In MATLAB, function inputs are always separated with commas and command inputs are always separated with spaces. Code is always interpreted in this way, which makes it easy to read. For instance:

■ Example 7.1 — Command syntax.
Some common examples of command syntax:

1. `clear all -except var1` clears all variables except for the variable named `var1`.
2. `load filename.mat` loads the variables saved to the filename.mat file.
3. `disp 'This is a sentence which is also a string'` displays the sentence: `This is a sentence which is also a string`

■ Example 7.2 — Function syntax.
Some common examples of function syntax:

1. `plot(x,y)` plots the vector `x` against the vector `y`, connecting each successive pair of points with a line segment *note:* these vectors must be the same size.
2. `scatter(x,y)` plots the vector `x` against the vector `y` in a scatter plot, rather than connecting points.
3. `disp(a)` displays the object `a`, which might be a string, a matrix or some other object.

To make a comment, just as in LaTeX, you put a `%` sign at some point in a line. Everything after that `%` sign but before the next line gets commented out.

■ Example 7.3 — Comments and displaying variables.
Running the script:

```
% This is a comment and will not be displayed
a = [ 1 2 ; 4 5; 7 8]; % the matrix describing something important
b = [ 1 2 7 8]; % a vector  describing something important
disp 'The important matrix is:';  disp(a);
disp 'The important vector is:';  b
```

yields the output:

```
The important matrix is:
     1     2
     4     5
     7     8

The important vector is:

b =

     1     2     7     8
```

In this way, you can get a script to output the answer to a calculation in a convenient format.

7.1.3 Loading data

When you open the program, you will see a box on the left titled 'Current Folder'. MATLAB can only see the files in your current folder, so this is where all your data files should go. Once a data file is in that folder, you can simply right-click the file and say 'Import Data...'. Alternatively, you can go to the 'File' menu and choose 'Import Data...'. In either case you will end up in the same import wizard. You will be previewing the data to make sure it is in the right format. MATLAB is usually pretty good about identifying what kind of data you have (e.g. 'comma-separated' or 'tab-delimited'). Once you have your data imported into a matrix (the most convenient) or into a collection of differently named vectors, you can save the workspace into what is called a `.mat` file. This is MATLAB's special file format for saving workspaces. Now whenever MATLAB's current folder contains the your `data.mat` (assuming you named it `data.mat`), you can just use the `load data` command to import all the data that was in your workspace when you saved it.

Useful Tip 7.1 — Importing multiple files .
If you have a lot of files of the same type and you want them all to import the same way, one way to do that is to check the 'Generate MATLAB Code' box in the import wizard. What this does is generate the code to import data of the form you specified in the prompt. If you save this code as 'importfile' once it comes up, you will be able to use this command to import files quickly and reliably in a script.

This way you don't have to click through the wizard n times if you have n data files.

7.1.4 Plotting functions and data

While in Mathematica, it is easy to tell the computer to plot a function against a dependent variable (or some other function of a dependent variable), MATLAB only understands vectors. This means that to plot $\sin(x)$ against x for instance, you need to first decide which x points you want. After you have an independent variable vector, and have the ability to construct a dependent variable vector (or if you already have two vectors of the same length), you can easily make a plot by calling the `plot(xdata,ydata)` function.

```
x = 0:1e-2:2*pi;
plot(x,sin(x));
```

In the above code, the statement `x = 0:1e-2:2*pi;` creates a vector of increasing values starting at `0` and working up in increments of `1e-2` (1×10^{-2}) until the point `2*pi`$= 2\pi$ is reached. Again, once the domain of the function is created, it is easy to simply apply the sine function while creating the `plot()`. Note that a semicolon has been placed after each line – the first is to keep the variable `x` from being displayed, the second is for aesthetics. In general there's no harm in putting down a semicolon so use them everywhere.

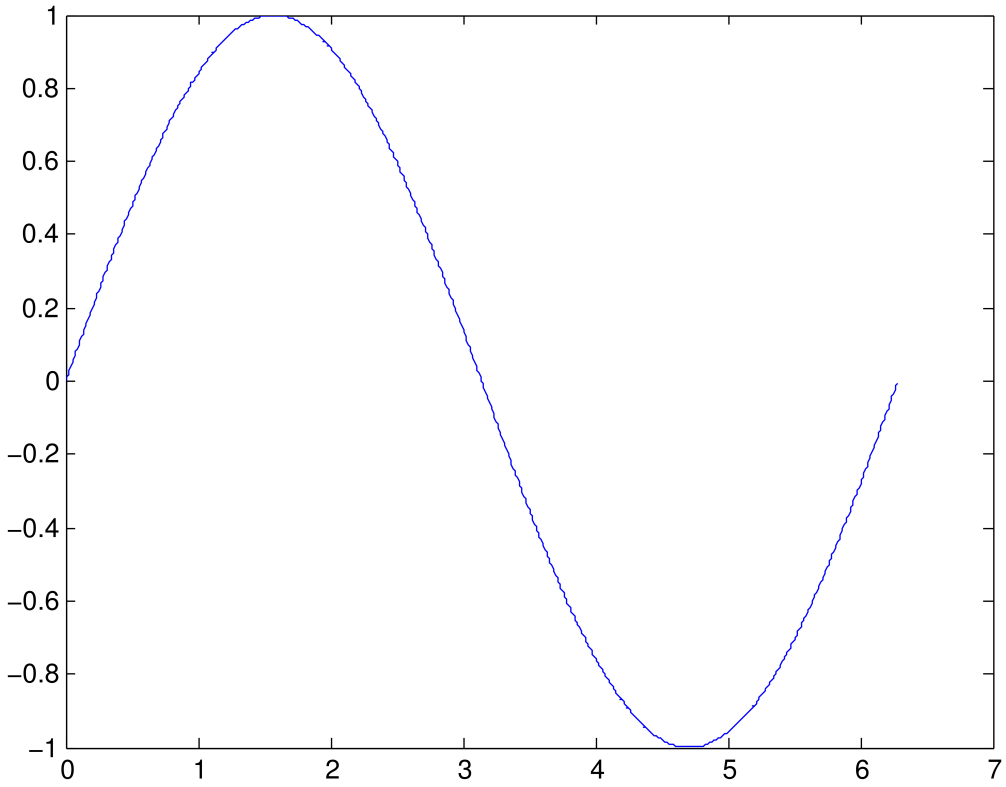


Figure 7.1: Plot of `sin(x)` against `x`, where `x` is a 1×629 vector from 0 to 2π.

To plot data, it is often most convenient to have all your data in a matrix. It is a good practice to document your variables and how you name them somewhere. Documenting how the code you write is relevant to the experiment can easily be done with comments in the same file as your analysis using % to create comments just as in LaTeX. In the code below, the data has already been imported using the import data wizard mentioned in section 7.1.3 and titled `data`. Fig. 7.2 shows the result:

```
plot(data(:,1),data(:,2));
xlabel('Column 1');
ylabel('Column 2');
title('Column 1 vs. Column 2');
```

This is the most basic kind of plot. However, often you may want to plot your data as points (in a scatter plot), rather than as a line plot. MATLAB has another function called `scatter()` which for two vectors works in exactly the same way as `plot()`. As in Mathematica, these functions are highly customizable.

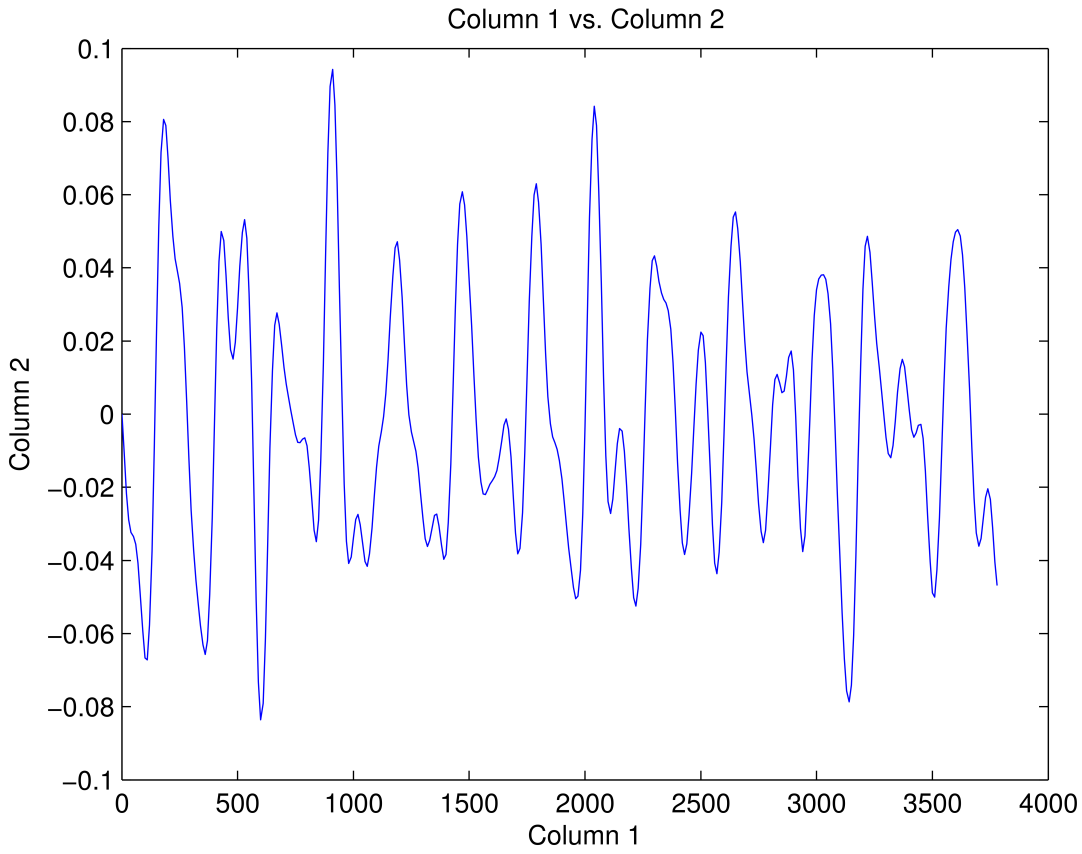


Figure 7.2: Plot of one column of data (`data(:,1)`) against another (`data(:,2)`).

R In MATLAB there's a very nice way to customize plot which so long as you only have one plot is very easy and efficient. It is called the *property editor*. You can get to it via the view menu if you have a plot open. It is a graphical user interface (GUI) with many options for titling plots, changing colors, sizes, line styles, etc.

7.1.5 Saving pictures

If you're happy with your plot and want to put it into your paper you must first export it by going to *Export Setup*. You can get to export setup through "File → Export Setup" if you have an element on the figure selected in the Property Editor, or in the Property Editor window.

Remember: you want to export files in the .pdf or .eps format as those do not have problems scaling when incorporated into a LaTeX document.

7.1.6 Histograms

To make a histogram of a vector or matrix of data, the simplest thing is simply to write `hist(data)`. This will create a histogram plot with a number of bins that MATLAB thinks is appropriate. However, you can specify the number of bins with `hist(data,bins)` and if you would like to plot the bin centers against the bin counts, you can assign the output of the `hist` to two vectors representing this as in the example below:

```
[count, bin] = hist(data(:,2),30);
subplot(2,1,1); plot(bin,count);
title('Plot from Histogram Output');
subplot(2,1,2); hist(data(:,2),30);
title('Regular Histogram Plot');
```

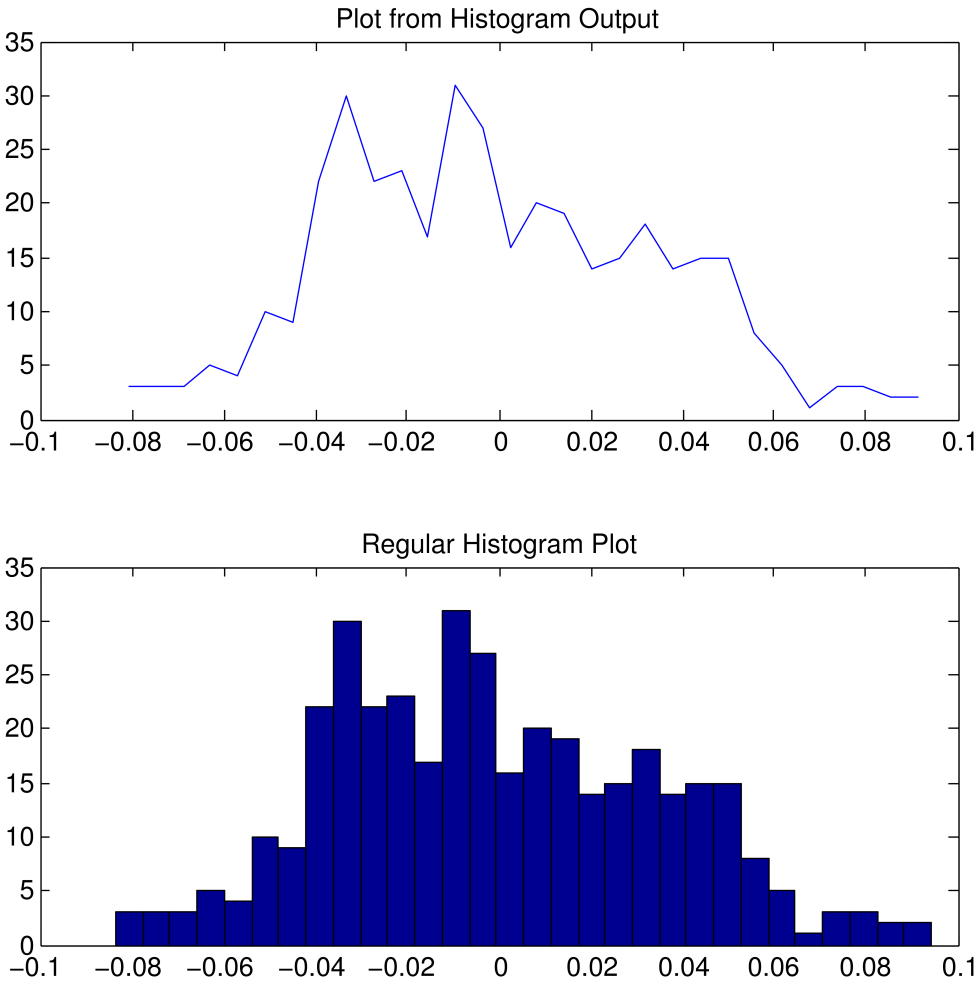


Figure 7.3: Plots of the same histogram portrayed in two different ways. (Top) Histogram data is stored in two variables (`[count, bin]`) from the `hist(data(:,2),30);` command which is a 30 bin histogram on `data(:,2)`. (Bottom) The traditional histogram from `hist(data(:,2),30);`. As you can see, `hist(data(:,2),30);` automatically outputs a plot unless you get output data from the function. If you were interested in how the data were distributed, `[count, bin] = hist(data(:,2),30);` would be most useful because you could fit the `bin` and `count` with some distribution function (a gaussian for instance).

Bibliography