Introduction To The PC

Mark Orchard-Webb
mark.orchard-webb@mcgill.ca

January 6, 2015

1 Introduction

Welcome to the 2015 version of this "tutorial!" This is the first year in which we are using Python as the environment, in the past we have used MATLAB™, the language C, and before that QuickBASIC™. This is the second year where we will exclusively be using the Arduino UNO™microcontroller to perform data acquisition. Previously, we had used Labmaster™data acquisition boards, which were phased out due to their reliance upon the very archaic ISA bus.

Each workstation is equiped with a workstation, it is a relatively comfortable Pentium IV with a half gigabyte of memory. Please do not reboot the computer, if there is a problem, please bring it to the attention of the technician. The workstation is named `ws339-`*nn*`.ugrad.physics.mcgill.ca`, where *nn* is the bench number. You can remotely login to your workstation via secure-shell (SSH).[1] To transfer files to and from your workstation, you can use secure-copy (SCP).[2]

It may be the case that some of you have experience with Python, but this tutorial will assume none. For those who do have experience, don't assume this will give you a free ride; there have been numerous instances in this course where students with less experience out performed their more experienced peers — success is more about understanding the problem than being able to write syntatically correct code.

Please refer to the document **Introduction to Linux System as implemented in the McGill Physics Department Undergraduate Laboratories** for details of the user interface.

2 Data and Experiments

Effectively any experiment in this course will consist of collecting data and then processing it in order to make statistical inferences about the system you have measured. These statistical inferences will be governed by uncertainties on the originally measured quantities. Basically a measurement without uncertainty is a useless measurement because nothing can be concluded from it. Before making a measurement it is critical to define exactly what you intend to measure. For example, you tentatively design an experiment to measure the mass of the caffeine molecule. Sounds reasonable, but it could mean you wish to measure the average mass of caffeine molecules in a sample of coffee, or it could mean you wish to measure the average mass of a specific caffeine molecule. In the second case your requirement

[1] **ssh ws339-nn.ugrad.physics.mcgill.ca** in a terminal on a Linux or MacOS machine, or using **PuTTY** on a windows machine.
[2] **scp** command on Linux or MacOS, **WinSCP** is a good, free tool for Windows.

for experimental precision will be much higher. Any quantity we can measure will have statistical fluctuations, so long as we are using equipment sufficiently precise to characterize this fluctuation we can make good statistical inferences about the quantity we are trying to measure. In the case where we wish to measure the average mass of a caffeine molecule in a sample the largest contribution to statistical fluctuation in the mass will be the distribution in ^{12}C and ^{13}C isotopes, if we can measure to a precision better that 1 atomic mass unit per molecule then we can characterize the distribution of masses and make statistically relevant statements about the parent population of caffeine molecules in the sample. In the case where we are trying to measure a single molecule[3] we will see statistical fluctuation due to vibration of the molecule, but this would require a precision better that 10^{-12} atomic mass units per molecule which is not possible in this laboratory.

A popular myth held by students entering this lab is that when there is no statistical fluctuation in a signal, then an appropriate error bar to use is half the smallest division available. This does give a reasonable estimate of accuracy, but is not very useful for statistical analysis of precision. For rigorous statistical purposes you need a more precise measurement device. However, an interesting and rather non-intuitive result is that you can make a measurement which is much more precise that the smallest division using a measurement device which is quantized by adding Gaussian noise with an average of zero and a standard deviation of a few divisions before the measurement. This will allow you to make sound statistical statements about the mean value of a signal while giving little insight into the distribution of values about this mean.

2.1 Error-bars

A brief review of what we mean by an error-bar will be useful. When one states

$$m_e = 9.10938215(45) \times 10^{-31} kg, \tag{1}$$

we are stating is that, given all we know, the probability of the our measured value of $9.10938215 \times 10^{-31}\ kg$ being more than $0.00000045 \times 10^{-31}\ kg$ different from the true value of m_e is 32%. Fortunately in nature random fluctuations are normally distributed, so we can usually get away with a more liberal interpretation that our measurements of m_e have been normally distributed about the value $9.10938215 \times 10^{-31}\ kg$, and that the standard deviation about this mean is $0.00000045 \times 10^{-31}\ kg$.

We graphically represent error-bars as vertical lines on a graph extending from one error-bar below the data-point to one error-bar above the data-point. Do not join up the points with a line, solid curves should be reserved to indicate model predictions.

Never present a measurement or derived value without an estimate of error, neither graphically, nor as a number!

2.2 Propagation of Error

As previously mentioned, random fluctuations in nature are normally distributed. If one has a set of independent variables which are normally distributed, which are combined, then the result will also be normally distributed, provided that the function by which they are combined is well behaved. This should be intuitively apparent if you observe that a well behaved function can be linearly expanded about a point (the mean value of the variable). In physical situations, if the value of the variable is fluctuating over such a wide range that this linear expansion is not accurate, then the source of fluctuation is

[3] it doesn't have to be a single molecule, we just have to make sure the molecules are identical in construction

probably not random. To restate a result with which you should be familiar, if a derived quantity Y is obtained from a set of independent random variables X_i, each with uncertainty σ_i, via the function

$$Y = f(X_1, X_2, ..., X_N), \tag{2}$$

then

$$\sigma_Y^2 = \sum_{i=1}^{N} \left(\frac{\partial f(X_1, X_2, ..., X_n)}{\partial X_i} \right)^2 \sigma_i^2, \tag{3}$$

where σ_Y, is the estimate of uncertainty on the derived value Y. When using this, be sure to verify that your variables are truly random, and that such a linear extrapolation is legitimate.

2.3 Measurements

Now, perhaps the following is a paradigm which will change in the future, it is the presumption that for most of the experiments you will perform, there is a measurement phase, during which a parameter is varied, the result of that variation are measured and recorded as a triplet of values. These values being the independent variable (the controlled parameter), the dependent variable (the result), and the uncertainty (derived from the time dependent fluctuations of the dependent variable). The reason for this being the constraints on storage space, it takes much less storage space to record two values,

$$\bar{v} = \frac{1}{N} \sum_{i=1}^{N} v_i \tag{4}$$

and

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (v_i - \bar{v})^2} \tag{5}$$

than it is to store all N values of v_i, when N is on order 1000. The astute reader may have observed that Eq.5 implies that the individual measurements must be stored since $\bar{v}$ is not known until after all N measurements have been made, however this calculation can be reformulated such that

$$\sigma = \sqrt{\frac{1}{N} \left[\sum_{i=1}^{N} v_i^2 - N\bar{v}^2 \right]}, \tag{6}$$

Thus one only need to keep track of two running sums. In the future, it may be more interesting to collect histograms or even entire raw data sets for later processing.

So, for the present, we have sets of data consisting of triplets of data, these are traditionally stored as files with one triplet per line, each line reading from left to right contains the independent variable, the dependent variable and the error-bar. In this tutorial we will develop tools for manipulating this data in Python. If you are proficient with some other language, please don't feel constrained to do it this way, this is an evolution of how we did things in the past, not a dogmatic prescription of how it must be done, but be aware that the materials provided will by Python based.

3 Getting Started

Python is an interpreted language. There are a few different ways of interacting with it. My personal preference is to write scripts and run them as arguments to Python, however other people enjoy using

en environment like iPython. Even if you do use iPython, while you could just type (or cut and paste) directly into the environment, but I would strongly advocate writing scripts. To do this you will need to familiarize yourself with an editor. My personal preference is **Emacs**, others prefer **Vim**. If you have a preferred editor but it is not installed, not hesitate to ask me for it.

You can move around on the virtual desktop by pressing and holding the flag key, [⊞], and one of arrow keys. When you first log in you will be located in the top left pane of the virtual desktop. Notice there is a thumbnail map of the entire virtual desktop in the top right corner of the screen.

To open a terminal, press and hold the flag key, [⊞]-[T] (Press and hold flag-key then press [T]).

I would suggest that you maintain a hierarchy of directories in order to keep your data, scripts and other files in logical, easily accessible locations. Figure 1. illustrates the process of making directories.

```
mark.orchard-webb@ws339-99:~$ mkdir 339
mark.orchard-webb@ws339-99:~$ cd 339
mark.orchard-webb@ws339-99:~/339$ mkdir intro
mark.orchard-webb@ws339-99:~/339$ cd intro
mark.orchard-webb@ws339-99:~/339/intro$
```

Figure 1: Log of commands used to create subdirectories

3.1 Running Python

Let us work through a few useful tasks. Copy all the files in the directory `/mnt/resources/339/intro` into this directory. These steps are illustrated in figure 2.

```
mark.orchard-webb@ws339-99:~$ cd 339/intro
mark.orchard-webb@ws339-99:~/339/intro$ cp -v /mnt/resources/339/intro/* .
‘/mnt/resources/339/intro/gaussian.data’ -> ‘./gaussian.data’
‘/mnt/resources/339/intro/histogram.data’ -> ‘./histogram.data’
‘/mnt/resources/339/intro/sinusoid.data’ -> ‘./sinusoid.data’
‘/mnt/resources/339/intro/triplet_plot.py’ -> ‘./triplet_plot.py’
```

Figure 2: Log of commands used to copy files from public resource area to current directory. Note the presence of the trailing dot in the `cp` command; the dot represents the current directory, which is the destination for the copied files.

The files with the extension `.data` are sets of triplet data. `gaussian.data` is the datafile used in the following examples, `sinusoid.data` and `histogram.data` are files to be used in the exercise. The last file, `triplet_plot.py` is a demo of plotting triplet data. Figure 3 shows how it can be used.

```
mark.orchard-webb@ws339-99:~/339/intro$
mark.orchard-webb@ws339-99:~/339/intro$ python triplet_plot.py gaussian.data
Data filename = 'gaussian.data'
File contains 200 rows, 3 columns
```

Figure 3: Log of execution of script `triplet_plot.py`.

This should result in a window popping up as shown in figure 4.

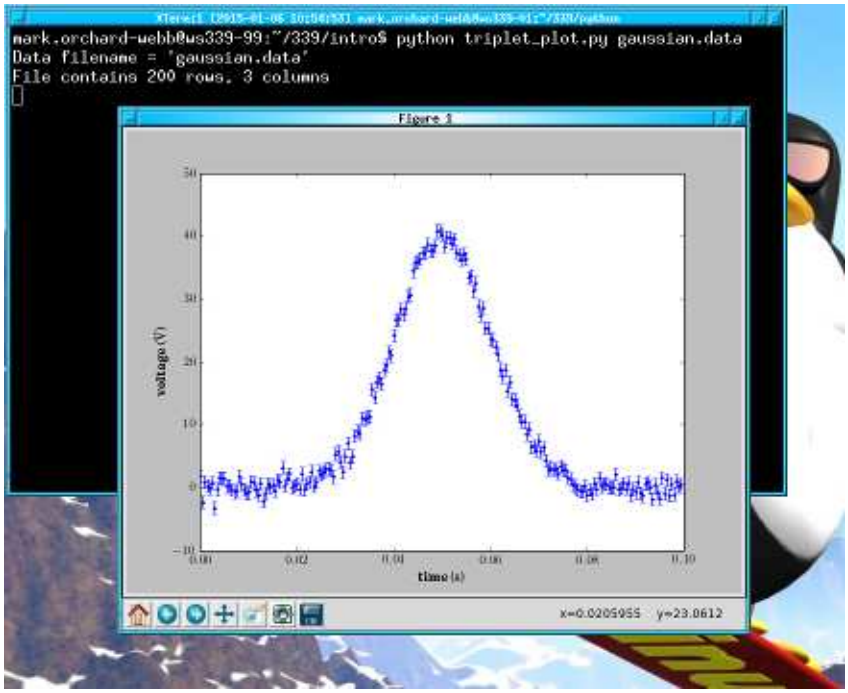

Figure 4: Graph generated by Python

For use in reports on would like to save representations of the graph, this is achieved with the diskette icon at the right-hand edge of the collection of icons at the left-bottom corner of the graph. I mention this, because many of you may not know what a floppy diskette is. This icon is hilighted with red arrows in figure 5. I would recommend saving as *EPS* or *PDF*, since they are vector based as opposed to pixel based, and with thus scale better.

Finally, to close the graph window, you can press `Ctrl-W` inside the graph window. `Ctrl-W` is the hot key used by the graph widget. You could kill Python using `Ctrl-C` in the controlling terminal, this is less friendly. You could also use the window manager as shown in figure 6 by clicking on the button in the top left corner of the window and selecting `Close`.

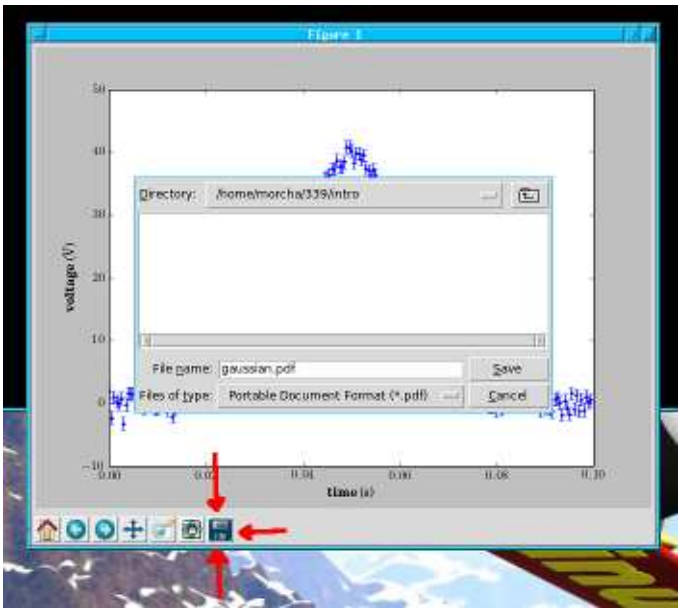

Figure 5: Saving a graph — icon to be pressed is hilighted with red arrows

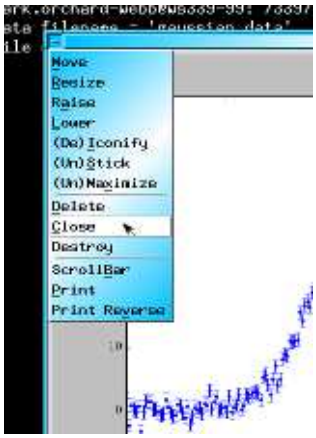

Figure 6: Closing a graph using the window manager

3.2 Analysis of `triplet_plot.py`

I will briefly describe the contents of this script.

First, lets open it in an editor. Unless you have your own preference, let's use `Emacs` as demonstrated in figure 7

```
mark.orchard-webb@ws339-99:~/339/intro$ emacs triplet_plot.py
```

Figure 7: Opening `triplet_plot.py` using `emacs`

This should result in the emacs window opening as shown in figure 8

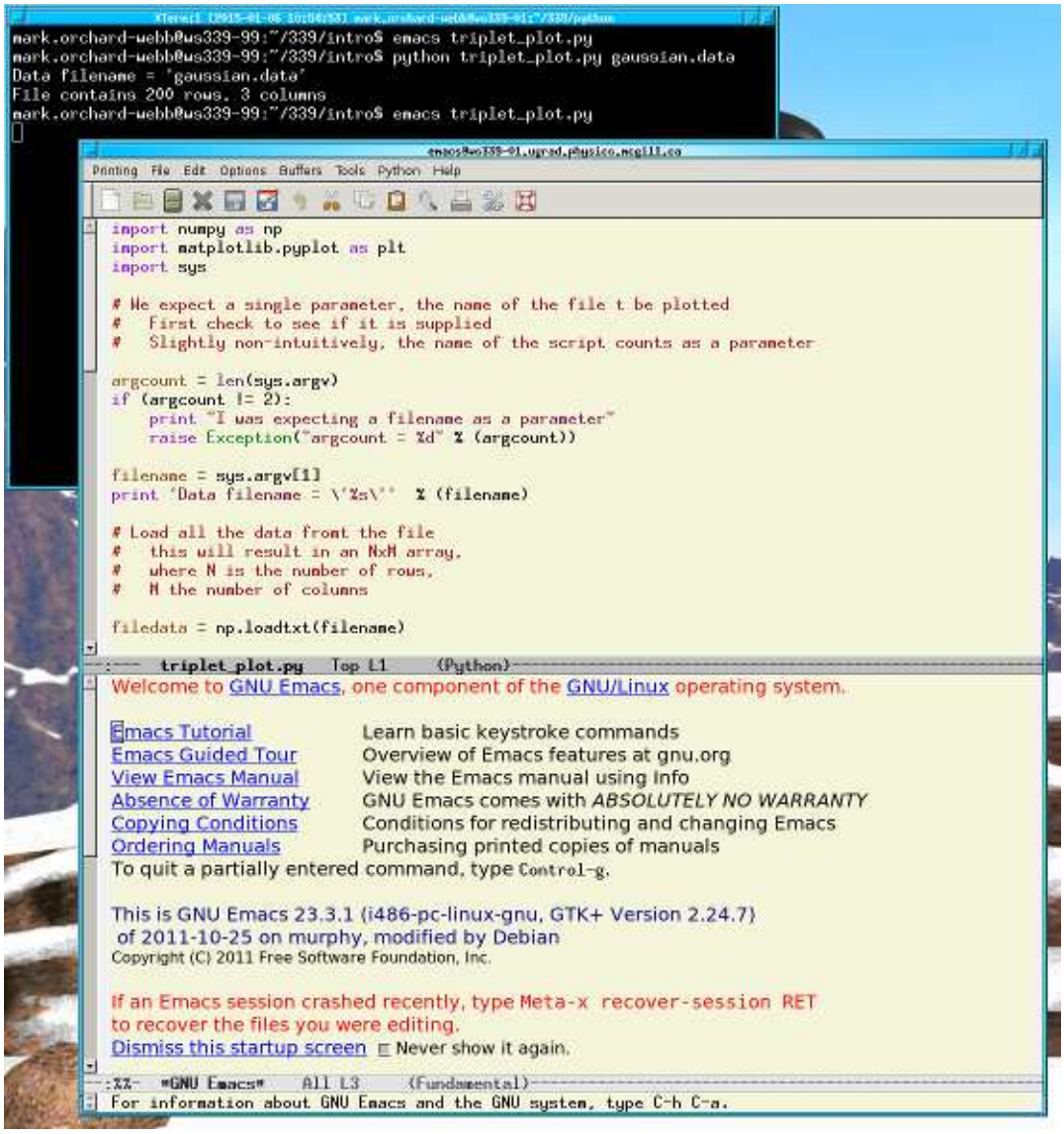

Figure 8: Successful launch of `Emacs` editor

As you can see the lower half of the screen is wasted by information which might be useful to new users. However you are physicists, you know what to do. We can maximize the `Emacs` window by presing `[⊞]-[X]` (Press and hold flag-key then press `[X]`).

In Python the number sign, `#`, is used for comments. Everything from, and including, the number

sign to the end of the line is ignored during execution of the script. This allows you to comment your code, this is a very good idea!

The first few lines at the top of the screen which start with the *keyword* `import` requests the modules which we intend to use. When the keyword `as` is used it allows us to give a module a nickname or alias, this is useful because in the case of the second `import`, rather than having to use the painfully long name `matplotlib.pyplot` every time we which to access the module methods, whe can simple use the shorter alias, `plt`. In the case of the module `sys` there seems little point in defining an alias.

For the most part the file is reasonably commented, so there seems little point in repeating the same text.

4 Characterization of `gaussian.data`

We will now examine some of the characteristics of the data contained in the `gaussian.data` data set. For example, we might wish to know the sum of all the independent variable measurement in the set,

$$Y = \sum_{i=1}^{N} y_i. \tag{7}$$

Applying error propagation, it is evident that

$$\sigma_Y = \sqrt{\sum_{i=1}^{N} \sigma_i^2}. \tag{8}$$

In MATLAB™, we would calculate these parameters as follows:

```
>> Y = sum(dep)

Y =

   2.0111e+03

>> sigma_Y = sqrt(sum(eb.^2))

sigma_Y =

   14.1421
```

Figure 9: Calculation of the sum of all measurements contained in `gaussian.data`.

So we get the result that

$$Y = 2.011(14)\ kV$$

.

Given the resemblance that the data has to a Gaussian distribution, we might wish to test the hypothesis that the data is described by a Gaussian distribution. To review, a Gaussian distribution is

defined as

$$G(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, \tag{9}$$

where μ in the mean and σ is the standard deviation of the distribution. The following are properties of the Gaussian distribution:

$$\int_{-\infty}^{\infty} G(x,\mu,\sigma)dx = 1, \tag{10}$$

$$\int_{-\infty}^{\infty} xG(x,\mu,\sigma)dx = \mu \tag{11}$$

and

$$\int_{-\infty}^{\infty} (x-\mu)^2 G(x,\mu,\sigma)dx = \sigma^2. \tag{12}$$

We can make a hypothesis that

$$v(t) = V_0 G(t,\mu,\sigma). \tag{13}$$

If we integrate $v(t)$, then we obtain V_0,

$$\int_{-\infty}^{\infty} v(t)dt = V_0 \int_{-\infty}^{\infty} G(t,\mu,\sigma)dt = V_0. \tag{14}$$

Similarly,

$$\int_{-\infty}^{\infty} tv(t)dt = \int_{-\infty}^{\infty} tV_0 G(t,\mu,\sigma)dt = V_0\mu. \tag{15}$$

Finally

$$\int_{-\infty}^{\infty} (t-\mu)^2 v(t)dt = \int_{-\infty}^{\infty} (t-\mu)^2 V_0 G(t,\mu,\sigma)dt = V_0\sigma^2. \tag{16}$$

One small point which comes to mind is that we don't have data spanning the range $t = -\infty...\infty$, only the range $t = 0...0.1\ s$. Oh well, Physics is the art of approximation!

To address the simplest problem first, what we want to do is use MATLAB™ to numerically calculate the integral

$$\int_{-\infty}^{\infty} v(t)dt \approx \int_{0\ s}^{0.1\ s} v(t)dt \tag{17}$$

A reasonably method would be to use the trapezoidal rule, that is

$$\int_a^b f(x)dx \approx (b-a)\frac{f(a)+f(b)}{2}, \tag{18}$$

where a and b would represent adjacent values of t in our vector `indep`, and $f(a)$ and $f(b)$ would represent the corresponding values in the vector `dep`.

So,

$$\int_{t_1}^{t_N} v(t)dt \approx \sum_{i=1}^{N-1} (t_{i+1}-t_i)\frac{v_{i+1}+v_i}{2}. \tag{19}$$

It may seem, due to the presence of both t_i and t_{i+1} elements in eq 19 that we would need to iterate over the vectors using a for loop, which we have not covered yet, but in fact by pulling out subranges of the vectors we can do it the MATLAB™ way, where looping is generally discouraged.

Create a new file `integrate.m` as listed in figure 10.

```
% integrate(x, y) : numerically integrates y(x) over range using
% trapizoidal rule
function [int_y] = integrate (x, y)
    x_left = x(1:end-1);                % x_left contains first N-1 points
    x_right = x(2:end);                 % x_right contains last N-1 points
    y_left = y(1:end-1);
    y_right = y(2:end);
    dx = x_right - x_left;              % delta x between adjacent
                                        % points in original x vector

    int_y = sum(0.5 * (y_right + y_left) .* dx);
```

Figure 10: Listing of `integrate.m`.

Briefly examining the contents of `integrate.m`, you should be comfortable that integrate is a function which accepts two vectors as parameters, and returns a single parameter.

The line `x_left = x(1:end-1);` creates a new vector `x_left` which contains a subrange of the vector `x`. You should, make youself very comfortable with this. See figure 11

```
>> A = [1:5]

A =

     1     2     3     4     5

>> A(1:end-1)

ans =

     1     2     3     4

>> A(2:end)

ans =

     2     3     4     5
```

Figure 11: Testing concept of extracting subrange of a vector.

The line `A = [1:5]` illustrates MATLAB™'s method of generating a vector.

Returning to the last line of `integrate.m`, note that the operator `.*` means perform an element by element multiplication of vectors, that is `c = a .* b`, where `a` and `b` are vectors, the vector `c` will contain elements $c_i = a_i b_i$. Finally the function `sum` returns the sum of a vector, that is `b = sum(a)`, where `a` is a vector will result in a scalar `b` with the value $\sum a_i$.

Verify that `integrate.m` faithfully implements the concept shown in eq 19.

To test the function `integrate`, I contrive a quick test as shown in figure 12.

```
>> x = [0:0.01:1];
>> y = x.^2;
>> integrate(x,y)

ans =

    0.3334
```

Figure 12: Testing `integrate` function with a polynomial.

The line `x = [0:0.01:1]` is a slight variation of the vector generating method in MATLAB™, it generates a vector over the range $0...1$ in steps of 0.01. Happily, we get the expected result

$$\int_0^1 x^2 = \frac{1}{3}. \tag{20}$$

We can now apply eq 14 through eq 16 to determine the parameters in our model, as shown in figure 13.

```
>> V_0 = integrate(indep,dep)

V_0 =

    1.0050

>> mu = integrate(indep,indep.*dep) ./ V_0

mu =

    0.0499

>> sigma = sqrt(integrate(indep,(indep-mu).^2.*dep) ./ V_0)

sigma =

    0.0102
```

Figure 13: Calculation of parameters in model.

Finally, it would be nice to write a script which would automate the work we have done so far. This, and a little extension is illustrated in figure 14.

```
function process (data,graphname)
triplet_plot(data,'+');
[t,v,e] = triplet_explode(data);
V_0 = integrate(t,v);
mu = integrate(t,t.*v) ./ V_0;
sigma = sqrt(integrate(t,(t-mu).^2.*v) ./ V_0);
G = exp(-0.5.*((t-mu)./sigma).^2)/(sqrt(2.*pi).*sigma);
hold on
plot(t, V_0.*G,'r');
xlabel ('Time (s)');
ylabel ('Voltage (V)');
legend ('data','model');
print('-depsc', graphname)
```

Figure 14: Code for function process, which automates work done thus far in tutorial, with extension.

A few notes on the extensions. The line beginning `G = exp(-0.5` is an implementation of eq 9. The line `hold on` is required to allow the later addition of a second curve to the figure created by the `triplet_plot`. function. The command `plot(t, V_0.*G,'r');` adds the curve, from our hypothesis in eq 13, to the plot of the data. `xlabel`, `ylabel` and `legend` add the appropriate text labels to the graph. Finally, `print('-depsc', graphname)` saves the figure to disk with the name which is supplied as a parameter to the function `process`.

Figure 15 shows an example of the use of the `process` function. Figure 16 shows the EPS graph produced by that usage.

```
>> load 'gaussian.data'
>> process(gaussian,'final.eps')
```

Figure 15: Illustration of usage of `process` function.

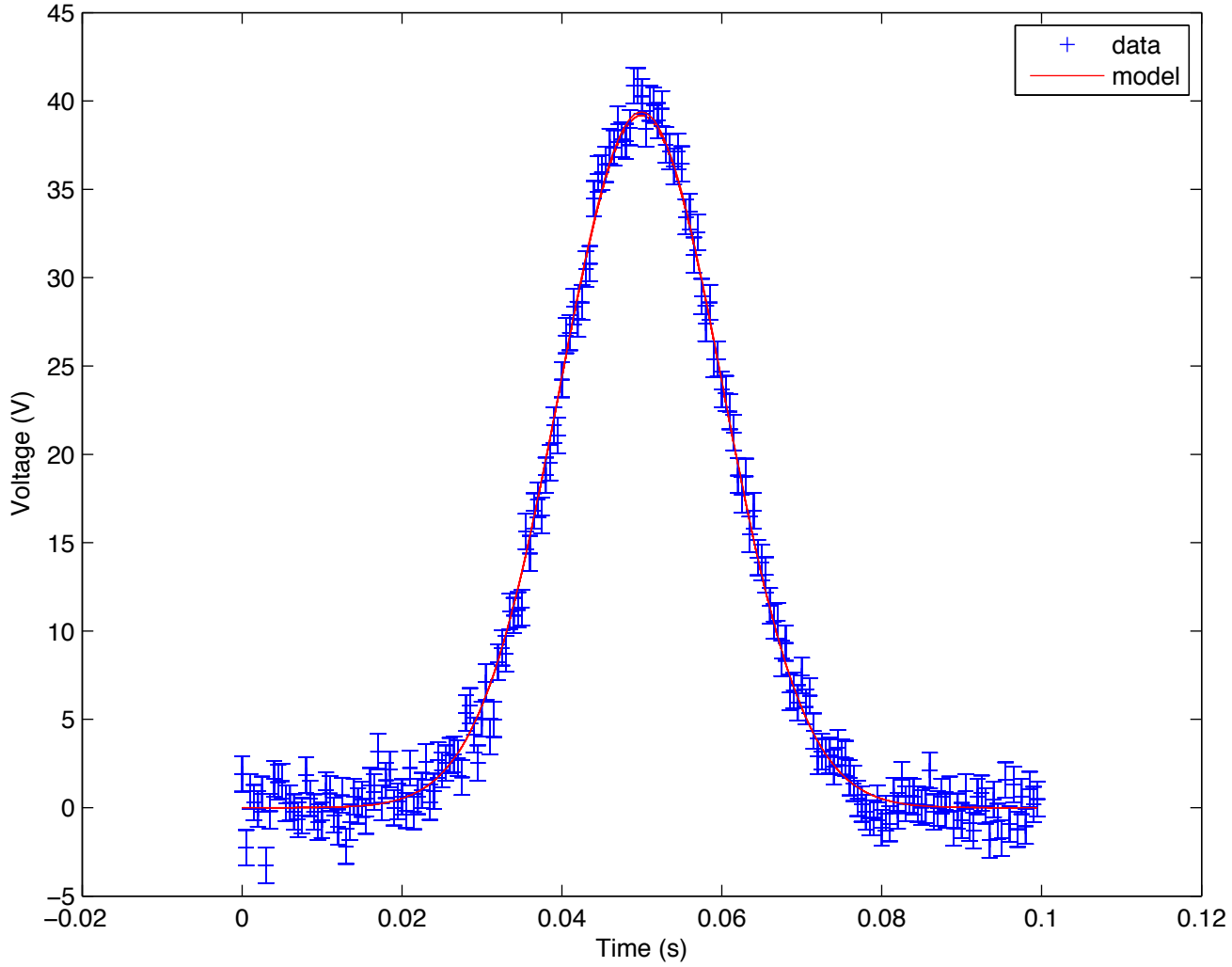

Figure 16: EPS graph produced by session illustrated in figure 15.

4.1 Formats for plotting

The following is *adapted verbatim*[4] from the MATLAB™ help.

Various line types, plot symbols and colors may be obtained with PLOT(X,Y,S) where S is a character string made from one element from any or all the following 3 columns:

b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	–	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
w	white	v	triangle (down)		
		∧	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

5 Apologies

Previous years' students have been asked to provide feedback on how this document could be improved, but they indicated that it was all fine. This is probably not true, so please feel free to provide your own feedback.

Never give in, never, never, never—never, in nothing, great or small, large or petty—never give in except to convictions of honor and good sense. – *Winston Churchill*

[4] 21st Century meaningless statement