Data Analysis (Python)

January 16, 2018

1 Introduction

1.1 Background Reading

- "Measurements and their Uncertainties", Ifan G. Hughes and Thomas P. A. Hase.
- "Data Reduction and Error Analysis for the Physical Sciences", Philip R. Bevington and D. Keith Robinson.
- "Beginners Guide to Python" `https://wiki.python.org/moin/BeginnersGuide`

1.2 Objectives

- Become familiar with a data analysis program such as Python.
- Learn techniques of graphing data and functions.
- Learn statistical methods that can be applied to experimental data.

Your grade for this lab will be based on a report only; no lab notebook is required.

1.3 Equipment Needed

- A computer with Python including numpy, scipy, and matplotlib libraries.
  The Python(x,y) distribution is recommended, but you may use any distribution or version that you choose.

1.4 Motivation

Modern experimental physics measurements make ample use of computers. Computers are capable quick and accurate computation and can process and handle large data samples if needed. Computers are also used extensively to simulate physical systems to model complex behavior of a system that may depend on many parameters. In this lab, you will explore various projects related to data plotting, analysis and fitting. There are also exercises in error analysis and propagation relevant to the labs in the course.

There are now a number of computer programs that provide a way to make calculations and graph their results. You can get your own copy of Python by downloading it from the web free of charge (see below). R, Matlab/Octave, Mathematica, and Maple are alternatives that you may use, although they are only recommended if you are already proficient in their usage. Python, R, and Octave can be downloaded for free. For most of the data analysis, Python will be the primary program used in this lab. All of the exercises below are presented along with examples in Python.

Use the interpreter prompt interactively for trying different functions, and help, but save work into an `.py` script using an editor (e.g. spyder). An alternative is to cut and paste the scripts and the graphics into word processing software and use a monospace font, such as Courier. The purpose of including annotated versions of this is to convince the instructor you did the work and to explain what you did well enough that a student like you could repeat the work. To avoid losing your files, save regularly and store the output on your personal flash drive or computer.

2 Starting Python

If you are using the Python(x,y) distribution, we recommend starting with the "IPython (Qt)" program, which is available under the Python(x,y) folder in the Start menu, for trying out commands interactively. We recommend using "spyder" for creating an editing Python files. Python scripts created in spyder can be run within spyder, but we recommend changing the default "Interpreter" in "Run Settings" to "Execute in a new dedicated Python interpreter" to make the plotting work as expected. Alternatively, Python scripts can be run by double-clicking on the script in Windows. You can install a (free) copy of Python on your own computer by downloading it from the Python(x,y) website: `https://code.google.com/p/pythonxy/`
The default installation should work well for everything in this course.

You can also use the Linux computers in Barnard Hall (EPS) 254, which have all needed software installed. On those (or another Linux computer), you can just type "python" in a terminal to get an interactive python interpreter. You can edit a python (.py) script file in a text editor such as gedit, vim, nano, etc. and run a python script file by typing e.g. "python myfile.py" in a terminal.

3 Python Primer

Here are some simple steps to get started. In addition, I have posted links to the Python resources and tutorials on the course website. The basic Python interpreter does not load the scientific plotting and fitting libraries by default. You need to load three packages: **numpy** for array manipulation, **matplotlib** for plotting, and part of **scipy** for fitting. Note that these packages come along with the Python(x,y) distribution listed above, and they are installed on the lab computers. To load the libraries, you always need to add these three lines at the beginning of your session or script.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
```

After importing the libraries in this way, Python syntax requires that you precede plotting commands by "`plt.`" and numpy commands by "`np.`". The scipy command that we will most commonly use can be accessed simply as "`curve_fit`". For example,

```
plt.errorbar(t, y, yerr=err) # plot a set of x,y data points with y-error bars.
```

The `#` symbol is used to add a comment. Everything on a line after the `#` is ignored by the interpreter. Another very useful function is `np.linspace`:

```
x = np.linspace(0, 10, 50) # 50 x points from 0 to 10
```

generates an array of 50 numbers equally spaced in the interval from 0 to 10 and assigns it to the variable `x`.

matplotlib has a well organized and detailed users guide at
`http://matplotlib.org/users/`.
A good starting point is the beginner's guide which has most of the commands that are needed to make plots for this class. I also suggest going through the tutorials sections for added practice. A user manual for **numpy** and **scipy** tools can be found at
`http://docs.scipy.org/doc/`
or at
`https://scipy-lectures.github.io/intro/`
More curve fitting examples are easily be found on the internet with keywords "python curve fitting". These websites show some curve fitting examples which you may find useful.
`http://wiki.scipy.org/Cookbook/FittingData`

4 Exercises: Python and Plotting

Below are a series of exercises with some sample commands to get you started. In your report, include your inputs and outputs and append clearly labeled plots where needed.

4.1 Numbers and Arrays

TYPE WHAT IS IN THIS COLUMN (COMMAND)	TO GET ACTION IN THIS COLUMN (COMMENT)
`5.0/3`	divide 5 by 3 giving the answer as a real number
`5/3`	division between two integers is truncated
`5.0**2`	use python as a calculator - almost any arithmetic expression is possible
`np.sin(np.pi/6)`	np.pi is π and the argument is in radians
`t = 10.`	define the variable t to be 10.0
`t2 = 2*t`	define the variable t2 to be twice t
`g = -9.8`	acceleration due to gravity in m/s^2
`y = g*t**2/2.`	calculate y as the distance (in meters) something falls in the time t = 10 s
`print y`	display the value of the variable y

4.2 Plotting Data and Curves

Goal: to write a python script (a `.py` file) to plot the mock data (the `t` and `y` points above) with error bars along with a mock theory curve (plot a line for the quadratic function `ytheory = 1000. + g*t**2/2` on the same graph).The table below has most of commands to get you started.

Note: in some consoles, such as ipython, matplotlib runs in *interactive mode*. In interactive mode, the calls to `plt.show()` used below are not necessary, and each plot command you enter will update the open plot immediately. This is helpful for interacting with the plot and testing commands, but does not work in Python scripts. So, you should get familiar with the use of `plt.show()` even if you first try out the commands in ipython.

Remember to include these lines at the beginning of your script:

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

COMMAND	COMMENT
`g = -9.8`	acceleration due to gravity in m/s^2.
`t = np.linspace(1, 10, 20)`	defines t as an array of 20 values from 1 to 10 (linspace returns an array of evenly spaced numbers over a specified interval)
`print t`	display the array you just created
`t = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])`	alternatively the values of t can be directly assigned (or read in from a file as in the exercise below)
`y = 1000.  + g*t**2/2`	creates corresponding y array that calculates how far object has fallen from initial height of 1000 m
`print y`	display the results for y evaluated at the values of t specified in the range.
`plt.plot(t, y)`	create a plot of t and y on a graph, but the graph isn't shown on some consoles or in scripts
`plt.show()`	display the graph (not needed in some consoles, but always needed in python scripts)
`err = 0.05*y`	generates array (will use to plot 5 percent y error bars)
`plt.errorbar(t, y, yerr=err, fmt='o')`	plot points with error bars
`t_theory = np.linspace(0., 11., 56)`	set function range and sampling for theory curve
`y_theory = 1000.  + g*t_theory**2/2`	define the theory curve
`plt.plot(t_theory, y_theory)`	plot the line on the same graph
`plt.show()`	do this at the end to show the plot

To complete this exercise make the changes listed below to the plot. In your report, describe the changes to the code and show the results.

a. Label the x and y axes.

b. Change the color, type, and size of symbols and change the color and line type.

c. Add a legend to the plot (for example "mock data points" for the points and "theory" for the curve).

4.3 Example of Input/Output

The goal of this exercise is to generate a sample of data points, write them to a file, and read them from the file to plot. You will need to be able to read data points in from a file for analysis in many of the experiments. In your report, include your code and include or describe the results.

COMMAND	COMMENT
`x = np.linspace(0, 10, 50)`	`50 x points from 0 to 10`
`y = np.exp(-x)`	`generate y values with exponential function`
`data_out = np.column_stack((x, y))`	`combine x and y into a two-dimensional array with two columns`
`np.savetxt('output.dat', data_out)`	`saves data file`

Check that the file `output.dat` has been created, inspect its contents in a text editor, and now read it in and plot it.

COMMAND	COMMENT
`x, y = np.loadtxt('output.dat', unpack=True)`	`reads from the file`
`plt.plot(x, y)`	`make the plot`
`plt.show()`	`show the plot`

5 Exercises: Analysis using Simple functions

In this section, you will use and plot several of the functions you will most commonly use for labs. In your report, please write up your answers and append clearly labeled plots where needed.

5.1 Linear and Quadratic Functions

These simple exercises could be done without a sophisticated Python program.

1. Two parameters are needed to specify a line in a plane; we normally use slope and y-intercept:

$$y = mx + b \tag{1}$$

where m is the slope, and b is y-intercept. For $m = 2$ and $b = -2$, calculate and plot this line in the range $x = 15$ to $x = -15$. Get comfortable with changing the limits of the graph and the positions of the tic-marks.

2. What is the slope of the line perpendicular to the above line? Draw it on the same graph. Calculate it and plot it so that the x and y axes have the same scale to prove the lines are perpendicular.

3. Plot a graph for the simplest quadratic: $y = x^2$ in the interval $x = [-5, 5]$.

4. The standard formula for a parabola is $(y - y_0) = a(x - x_0)^2$. What are the importance of the 3 parameters, x_0, y_0, a? Demonstrate qualitatively on a graph by varying these parameters.

5. Projectile motion with no resistance follows a parabola with the parametric equations

$$y(t) = y_0 + v_{0y}t - \frac{1}{2}gt^2 \tag{2}$$

$$x(t) = x_0 + v_{0x}t \tag{3}$$

To simplify, take $x_0 = y_0 = 0$. Choose $|\vec{v}| = \sqrt{v_{0x}^2 + v_{0y}^2} = 60$ m/s for the magnitude of the initial velocity. What is the equation for the initial trajectory angle in terms of the components of the initial velocity? Plot the trajectory y versus x, for an initial angle of 25° and the time interval [0, 5] seconds.

6. The initial trajectory angle for which the projectile travels the furthest in x is 45°. Verify this by plotting trajectories for 45°, slightly larger, and slightly smaller angles, all with the same initial speed. You should produce a second plot that focuses in on the end of the trajectory to show graphically which line has the longest range.

5.2 Exponential Function

The number $e = 2.71828...$ is very special because of its mathematical properties. The most obvious way this shows up is in calculus, where

$$\frac{d}{d\,x}e^x = e^x,$$

and

$$\int e^x d\,x = e^x.$$

Since many natural systems can be described in terms of exponential functions, those functions are found in many circumstances. For example, all radioactive nuclei decay according to an exponential function, a natural consequence of the concept that the probability for an individual nucleus to decay is constant. In addition, the sine and cosine functions can most easily be used in terms of exponentials: $\cos\theta = \Re(e^{i\theta})$ and $\sin\theta = \Im(e^{i\theta})$, where $\Re$ and means "take the real part of a complex number" and $\Im$ means the imaginary part. This property is used very often in circuit theory and the description of waves.

1. Calculate and plot the exponential function, $y = e^{-\mu x}$ for $\mu = 0.5$ in the interval $x = [0, 10]$. Now, change the vertical scale on the plot to logarithmic. How is μ related to the slope of the straight line that is now plotted?

2. This is the function which describe how a radioactive material decays (N_0 atoms, half-life $\tau_{1/2}$ of radioactive nuclei) and the charge in a capacitor decays in an RC circuit (capacitance C, resistance R, and charge Q_0). For each of these cases, how is μ related to the physical properties of the system?

5.3 Gaussian Function

The Gaussian function is very important in experimental work and statistical analysis of data. The Gaussian probability density function is given by

$$f(x) = N_0 \exp(-(x-\mu)^2/2\sigma^2) \tag{4}$$

where μ is the mean, σ is the standard deviation and N_0 is a normalization constant. The value of N_0 is defined so that

$$\int_{-\infty}^{\infty} f(x)dx = 1 \tag{5}$$

which gives $N_0 = 1/(\sigma\sqrt{2\pi})$.

1. For $\sigma = 2$ and $\mu = 0$ plot the normalized Gaussian function $f(x) = N_0 e^{-(x-\mu)^2/2\sigma^2}$, in the interval $x = [-6, 6]$. (see the example in the Appendix for defining a Gaussian function). Shift the center of the Gaussian and change the standard deviation. Replot the function on the same graph.

2. Verify that the correct normalization for the Gaussian is $N_0 = 1/(\sigma\sqrt{2\pi})$ using the Python integrate function "`quad`". You will need to import "`quad`" from scipy using the line "`from scipy.integrate import quad`" (see the example in the Appendix for integrating a simple function). For practical calculations, you can put in specific values, and upper and lower limits of the integration can be set to $\pm 10\sigma$.

3. Demonstrate numerically that the mean of the Gaussian is μ using the definition of the mean of a continuous function

$$\mu = \frac{\int_{-\infty}^{\infty} x f(x) d\,x}{\int_{-\infty}^{\infty} f(x) d\,x}. \tag{6}$$

4. An important property of the Gaussian function is the fraction of probability between various multiples of σ. Demonstrate numerically using Python that 68.3% of the probability is in the range $[\mu - \sigma, \mu + \sigma]$. Therefore, *roughly one third of all individual measurements will be more than one standard deviation away from the average of many measurements.* If you have apparatus that is not very accurate, you must make a number of trials to be sure you have good data. On the other hand, using a vary accurate device such as the Keithley multimeter allows you to make one measurement and quote the result with confidence. Now find the integrated probability within $\pm 2\sigma$, $\pm 3\sigma$,$\pm 4\sigma$, and $\pm 5\sigma$. You should see that the probability to get a data point more than three standard deviations away from the average of many trials is very low.

6 Exercises: Statistical Analysis

All measurements have uncertainties associated with them which must be estimated and reported along with the result. An experiment is performed to test a hypothesis or to measure a previously unknown property. The result is compared with a prediction from a calculation or with an independent measurement. In either case, the experimenter must produce an objective estimate of the measurement uncertainty independent of the expected answer. Then, the job is to understand why the objective experimental result agrees or disagrees with the prediction based on the hypothesis.

In your report, please write up your answers and append clearly labeled plots where needed.

6.1 Errors, Means and Standard Deviations

It is important to understand the following point: the act of making measurements has inbuilt statistical error. There can be many reasons why the same quantity measured many times does not return the same value. Typically we assume that the quantity x being measured is drawn from a so-called *parent* probability distribution. That is, if we could measure it infinitely many times, the frequency of occurrence of different sample values would have the form of this parent probability distribution. Since we can never do this, we must do our best to estimate the parameters of the parent probability distribution. A standard assumption is that this parent distribution is a Gaussian distribution characterized by two quantities: μ and σ. The first represents the "true" value of the variable x and σ represents the variation in x we observe when we make measurements. Therefore, we are trying to estimate both of these values. To estimate μ we use the following quantity known as the sample mean,

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{7}$$

where N is the number of trials. This sample mean is the best estimate of the mean of the true underlying distribution μ, but won't be equal to it unless we make an infinite number of measurements. We also want to understand the following two questions:

- If one new trial measurement is performed, how far will it likely lie from the sample mean? In principle this is answered by knowing σ. But we don't know σ, so this is usually expressed in terms of a sample standard deviation s or sample variance s^2. s can be calculated from

$$s^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2 = \frac{N}{N-1}[\bar{x^2} - (\bar{x})^2] \tag{8}$$

  We say that $\bar{x}$ and s are the mean and standard deviation of the *sample* distribution, respectively, while μ and σ are the mean and standard deviation of the *parent* distribution, respectively. We will show below that approximately 68.3% of the time, another measurement of x will lie in the range $[\mu - \sigma, \mu + \sigma]$.

- Suppose N new trial measurements are performed. If the sample mean of these observations is calculated, how likely is it that this answer will be close to the original sample mean? This is expressed in terms of the variance in the sample mean or the standard deviation in the

mean, s_m. For independent, identically distributed measurements,

$$s_m = \sqrt{\frac{1}{N(N-1)}\sum_{i=1}^{N}(x_i - \bar{x})^2} \approx \frac{1}{\sqrt{N}}\sigma \tag{9}$$

While s_m gets smaller as N increases, the same is not true for s. This is because s estimates the width of the parent distribution, which is a fixed quantity. However, s_m estimates the accuracy of our knowledge of μ, which gets better as we increase N. The actual mean μ (mean of the parent distribution) will lie in the range $[\bar{x} - s_m, \bar{x} + s_m]$ 68.3 % of the time.

In this exercise, we are considering measurements of the length of a table. All data shown below is in units of centimeters.

212.3	211.5	210.8	209.8	211.1
210.6	213.2	211.7	212.6	210.3
212.1	211.5	210.6	213.0	212.1
211.7	212.1	211.3	211.8	211.4
213.4	210.5	211.0	211.1	212.7

Using this data set:

1. Calculate the sample mean, standard deviation and the standard deviation in the mean for these measurements (use `np.mean` and `np.std`). In future labs, all measurements should be reported as $\bar{x} \pm s_m$. If you do not make many measurements, then the error usually is reported by referring to the instrument's uncertainty quoted in the manufacturer's datasheets.

2. Plot a histogram of the results. (Code in the appendix includes an example of histogramming with the command "`hist`" in python).

3. Superimpose a Gaussian curve on your data points (see example code in the Appendix). Use the sample mean and sample variance you calculated as parameters for the Gaussian. To do this you will need to arrange it so that the total integrated area under the Gaussian is equal to the total integrated area under the histogram. Either apply the data sample normalization to the Gaussian curve or alternatively normalize the data histogram to 1 by dividing by the total number of events.

4. Perform a Gaussian fit to the data points and compare with your result. (See code in Appendix).

6.2 Error Propagation

Once you have a measurement, e.g. $\bar{x} \pm s_m$, you must carry the uncertainty in that value through any calculations you do with that number. This is known as *error propagation*. You can find the details of how to propagate errors in Hughes and Hase. In all future labs, you will be expected to propagate your errors whenever you are performing calculations with measured values.

6.3 Curve Fitting

This exercise is an introduction to linear fitting. You will encounter this again and will need to perform curve fits for data analysis in many of the labs.

1. Below, x and y are simulated experimental data from an experiment.

$$\begin{aligned} x &= (-2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0) \\ y &= (17.3, 18.9, 10.3, 11.4, 2.7, 3.7, -3.5, -4.0, -7.9, -12.1, -17.3, -15.9) \\ y_{err} &= (2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0) \end{aligned}$$

   Note that in a real experiment both x and y might have uncertainties. In that case, we would choose the variable with the fractionally larger uncertainties to be y, and ignore the uncertainties in x. Fitting while taking into account uncertainties in both x and y is beyond the scope of this course.

2. Plot these data points with error bars and *estimate* the slope and intercept of the straight line that best matches your data. Plot your estimated best fit line on the same graph using Python, and fiddle with the best fit line until you feel it is optimal based on the plot.

3. The '`curve_fit`' function in Python can automatically perform a fit for you. `curve_fit` works by minimizing a quantity called χ^2, which is equal to the sum of squared deviations between data and fit weighted by the estimated data point error,

$$\chi^2 = \sum_i \frac{(y_i - y_{fit}(x_i))^2}{\sigma_i^2} \tag{10}$$

   This process is called least squares fitting. Perform a fit to a line using '`curve_fit`' in Python and plot the best fit line using a different color or line style on the graph (see appendix for examples). Comment how well this line matches your estimate in the previous step.

4. Qualitatively assess the goodness of your estimated fit and the results of `curve_fit` by counting the fraction of points where the fit is within the error bars; for a 'good' fit, that will be about $\frac{2}{3}$.

5. Quantitatively, the goodness of the fit can be measured with the final value of χ^2 after minimization. A good fit will have it approximately equal to the number degrees of freedom of the fit, which is the difference between the number of data points N and the number of fit parameters M. In this case, $N - M = 10$. It can be convenient to define the reduced χ^2

$$\chi^2_{red} = \frac{\chi^2}{N - M} \tag{11}$$

   A $\chi^2_{\text{red}} \gg 1$ indicates the data is not well fit by the function *or* the estimated data point errors are too small. If $\chi^2_{\text{red}} \ll 1$, the fit might be too good; perhaps the estimated data point errors are too large. Compare the χ^2_{red} you get from the different lines you obtained in the previous steps (see Appendix 7.5 for how to calculate χ^2_{red}).

6. Try to "eyeball" the uncertainty in the slope and the intercept. To do so, begin varying the slope and intercept so that the plotted line goes through only $\sim 1/3$ of the points (including their error bars). Find the maximum and minimum such slope and intercept.

7. Quantitatively, the uncertainties in the fit parameters (slope and intercept) can be calculated by taking the square root of the diagonal elements of the covariance matrix, which is returned by `curve_fit` (see Hughes and Hase, Chapter 7 and code in the appendix). Evaluate the uncertainties in the slope and intercept returned by `curve_fit` and compare them to your "eyeballed" uncertainties.

8. What happens to the $\chi^2_{\rm red}$, fit parameters, and fit parameter uncertainties if you increase the y_{err} values by a factor of 4? What if you reduce them by a factor of 4? Why could you have expected these results?

9. Using the original y_{err} values, create a new array $r_i = y_i - y_{fit}(x_i)$ which are known as the residuals. Plot r versus x, with error bars equal to y_{err}. What do you expect to see? Comment on what your results show.

10. Finally, consider a new function `wrongfit = 10.5*x - 7.2`. Imagine that there was a "demon" in your computer who decided to mess with your fitting program. How would you know if this had happened? Plot the residuals from your data with this fit function. What do you observe? Can you use the residuals to correct the "wrong" fit function? Explain and carry out the procedure.

We conclude this section by emphasizing that there are many subtleties to curve fitting that we cannot hope to cover in this brief introduction. Through experience, it is useful to follow the procedures below for different types of problems.

- If your fit doesn't look right, be sure to give `curve_fit` a good initial guess for the fit parameters. This is often not needed for linear fits, but almost always needed for fitting to more complex functions (e.g. see Appendix 7.5).

- Never stop graphing till you see the error bars! This means residuals are nearly always the best check of goodness of fit.

- When all the information has been extracted from a data set, the points on the residual plot should be randomly scattered about the x-axis ($r = 0$), with $\sim 2/3$ of the points being within about one error bar away from it (why?).

- For an ideal fit, one in which data is randomly scattered about the best-fit line and the error bars are consistent with the scatter, the value of $\chi^2_{\rm red} \sim 1$. As a corollary, one can use the scatter in the data to estimate the error bars, for instance in those situations where you did not repeat the experiment many times for each point on the x-axis.

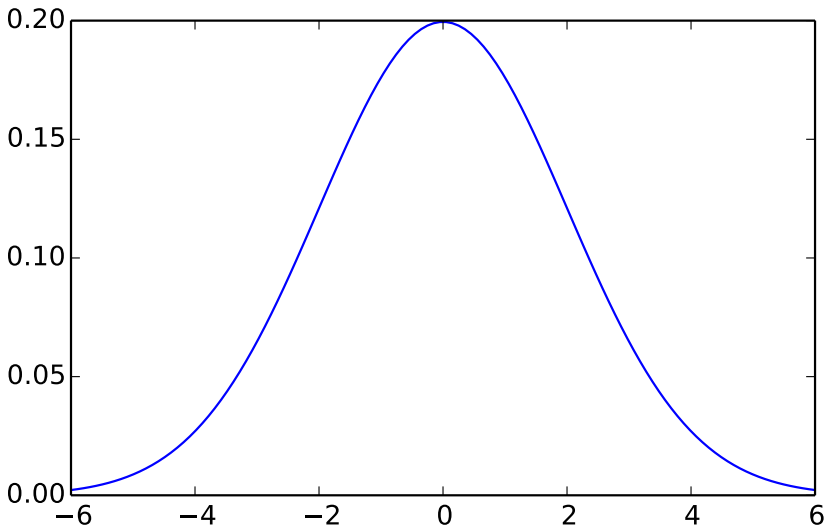
7 Appendix: Python Examples

7.1 Gaussian Plot

Example code to generate and plot a Gaussian function.

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

# define a function which calculates a Guassian
def gaussian(x, N0, mu, sigma):
    return N0*np.exp(-0.5*((x-mu)/sigma)**2)

mu = 0.
sigma = 2.
N0 = 1/(sigma*np.sqrt(2*np.pi))

x = np.linspace(-6., 6., 120)

plt.plot(x, gaussian(x, N0, mu, sigma))
plt.show()
```

Output:



7.2 Numerical Integration

Example code to integrate a simple function.

```
import numpy as np
from scipy.integrate import quad

# define the a function to integrate
def integrand(x, a, b):
    return a*x + b

a = 2.
b = 1.
lower_bound = 0.
upper_bound = 1.

I = quad(integrand, lower_bound, upper_bound, args=(a,b))

# the first number is the integral, the second is an upper limit on the error
print I
```

Output:

```
(2.0, 2.220446049250313e-14)
```

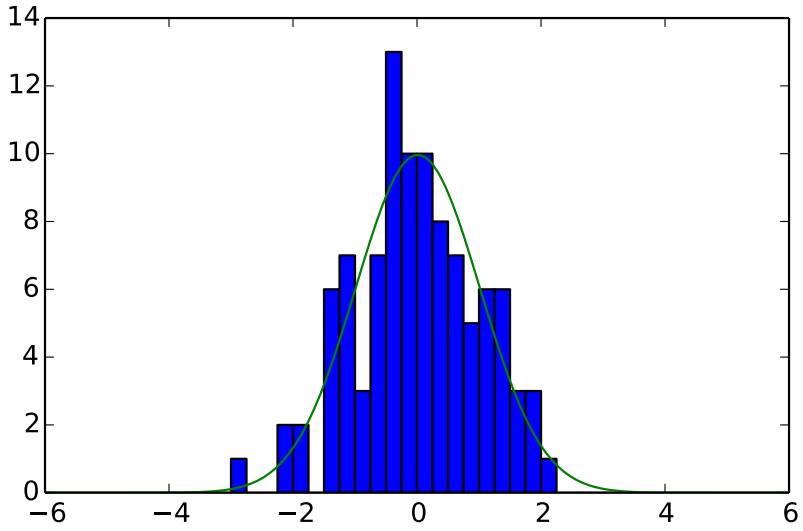
7.3 Gaussian Fit

The code below generates a set of data points distributed with a Gaussian (also called a 'Normal') distribution. The data will change each time you run the code. The data set can be saved in a file and read back in to be studied again. The code also performs a Gaussian fit to the data and returns the best fit values. (you can compare them with those of the generated true distribution).

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

sigma = 1. # Gaussian sigma parameter
mu = 0.    # Gaussian mean parameter
N = 100    # number of points to generate

# generate data points with a random Gaussian distribution
s = np.random.normal(mu, sigma, N)

# plot the data points as a histogram
b = np.linspace(-6, 6, 49)
plt.hist(s, bins=b)
```

```
# fit to a Gaussian distribution
mu_fit, sigma_fit = norm.fit(s)

# print best fit parameters
print mu_fit, sigma_fit

# overlay the fitted curve on the data points
def gaussian(x, N0, mu, sigma):
    return N0*np.exp(-0.5*((x-mu)/sigma)**2)

x = np.linspace(-6., 6., 120)
N0 = 1/(sigma_fit*np.sqrt(2*np.pi))  # normalize area to 1

# to properly normalize, multiply by number of events and bin size
renorm = N*(b[1]-b[0])
plt.plot(x, renorm*gaussian(x, N0, mu_fit, sigma_fit))
plt.show()
```

Output:



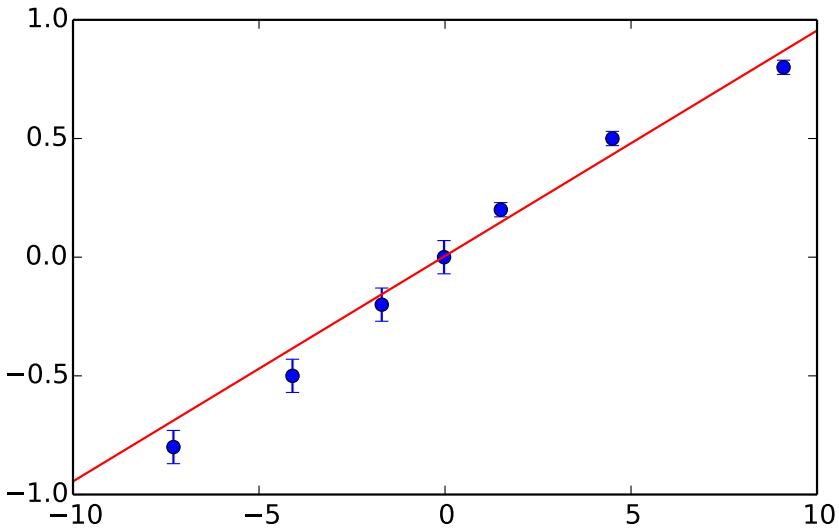
7.4 Linear Curve Fitting

Example using `curve_fit` to fit a line.

```
import numpy as np
```

```
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

x = np.array([-7.30, -4.10, -1.70, -0.02564, 1.50, 4.50, 9.10])
y = np.array([-0.80, -0.50, -0.20, 0.0, 0.20, 0.50, 0.80])
yerr = np.array([0.07, 0.07, 0.07, 0.07, 0.03, 0.03, 0.03])

# model function you want to fit to the data
def function_to_fit(x, a, b):
    return a*x + b

# the point at which you want to evaluate and plot the fit function
#   this is only used for plotting, not for fitting
plot_x = np.linspace(-10, 10, 20)

# evaluate the best fit parameters
popt, pcov = curve_fit(function_to_fit, x, y, sigma=yerr, p0=None)
print "Fit Parameters:", popt
print "Fit Parameter Uncertainties:", np.sqrt(np.diag(pcov))

# plot the data and the fit
plt.errorbar(x, y, yerr, fmt='o')
plt.plot(plot_x, function_to_fit(plot_x, *popt), 'r')
plt.show()
```

Output:



7.5 Nonlinear Curve Fitting

Example of using `curve_fit` from scipy to do a least squares fit to an exponential function. First, generate mock data with random Gaussian scatter. Error bars are also generated with random sizes to demonstrate their effect on the fit. The red curve is the fitted curve weighted by error bars. Note that the data and errors will change each time you regenerate the data.

First, here is the script to generate the data:

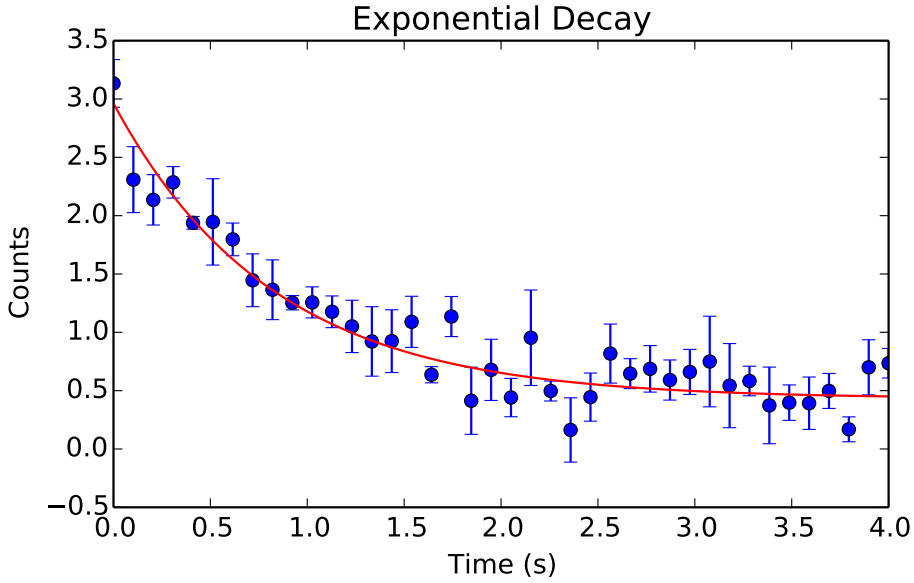
```
import numpy as np

def decay(t, a, b, c):
    return a*np.exp(-b*t) + c

t = np.linspace(0, 4, 40)
a = 2.5
b = 1.3
c = 0.5
y = decay(t, a, b, c)

# add some noise to the data
y = y + 0.2*np.random.normal(size=len(y))

# generate random error bars
yerr = 0.2 + 0.08*np.random.normal(size=len(y))

np.savetxt("curve_fit_data.dat", np.column_stack((t, y, yerr)))
```

Now, analyze the data. The data is loaded from a text file with x, y, and y_{err} columns. With small changes, this script can be used for almost all the curve fitting needed in this course.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# name of the file to load
data_filename = "curve_fit_data.dat"

# model function you want to fit to the data
#   include as many parameters as you need
def function_to_fit(x, a, b, c):
    return a*np.exp(-b*x) + c

# initial guesses for fit parameters
# the length of this array MUST match the number of fit parameters
fit_guess = np.array([1., 2., 0.])
```

```
# the points at which you want to evaluate and plot the fit function
#   this is only used for plotting, not for fitting
plot_x = np.linspace(0, 4, 100)

# load the data and evaluate the best fit parameters
x, y, yerr = np.loadtxt(data_filename, unpack=True)
popt, pcov = curve_fit(function_to_fit, x, y, sigma=yerr, p0=fit_guess)
print "Fit Parameters:", popt
print "Fit Parameter Uncertainties:", np.sqrt(np.diag(pcov))

# calculate chi^2 and reduced chi^2
chi2 = np.sum(((y-function_to_fit(x, *popt))/yerr)**2)
print "chi^2:", chi2
print "Reduced chi^2:", chi2/(len(x) - len(fit_guess))

# plot the data and the fit
plt.errorbar(x, y, yerr, fmt='o')
plt.plot(plot_x, function_to_fit(plot_x, *popt), 'r')
plt.xlabel("time (s)")
plt.ylabel("counts")
plt.title("Exponential Decay")
plt.show()

# plot the residuals
plt.errorbar(x, y-function_to_fit(x, *popt), yerr, fmt='o')
plt.plot(np.array([plot_x[0], plot_x[-1]]), np.array([0, 0]), 'r')
plt.xlabel("time (s)")
plt.ylabel("Residual Counts")
plt.title("Exponential Decay Residuals")
plt.show()
```

Output: The data and fit:



Output: The residuals: