PHYS 326 Lab 3

Graham Roberts

In this lab, we will practice fitting a line to data. The task is not particularly hard, but will give those who are new to programming a chance to practice some basic methods. You may adapt this template for your write-up. Don't forget your name!

Part 1a:

Let's start with something to which we already know the answer:

In [71]:
```
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
```

In [72]:
```
x=np.array([1,2,3])
y=np.array([3,5,7])
```

From the textbook (equations 8.10 through 8.12), we know the coefficients are

$$A = \frac{\sum x^2 \sum y - \sum x \sum xy}{\Delta}$$

$$B = \frac{N \sum xy - \sum x \sum y}{\Delta}$$

where $\Delta = N \sum x^2 - \left(\sum x\right)^2$

In [73]:
```
#A=np.sum(x*x)*np.sum(y)-np.sum(x)*np.sum(x*y)
#B=x.size*np.sum(x*y)-np.sum(x)*np.sum(y)
#delta=x.size*np.sum(x*x)-(np.sum(x))**2
```

In [74]:
```
def findA(x,y):
   return np.sum(x*x)*np.sum(y)-np.sum(x)*np.sum(x*y)
```

In [75]:
```
def findB(x,y):
    return x.size*np.sum(x*y)-np.sum(x)*np.sum(y)
```

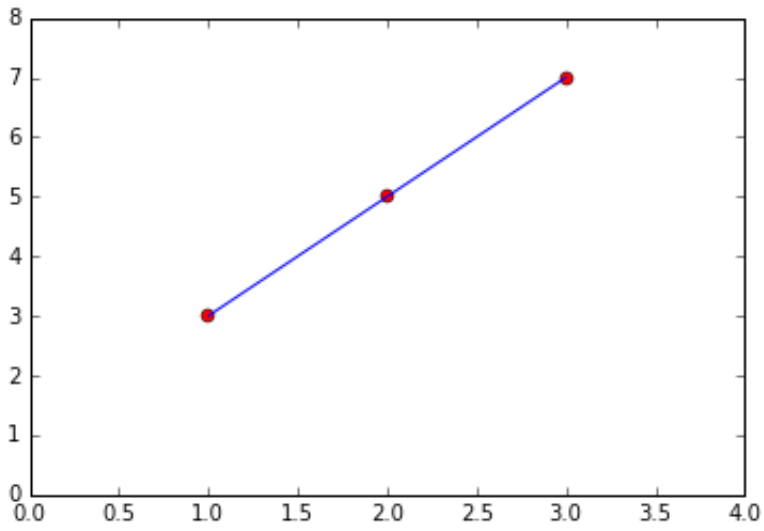
In [76]:
```
def findDelta(x):
    return x.size*np.sum(x*x)-(np.sum(x))**2
```

In [ ]:
```
```

In [77]:
```
delta=findDelta(x)
A=findA(x,y)/delta
B=findB(x,y)/delta
```

In [78]:
```
### SOMETHING MUST BE FIXED FOR THIS TO WORK CORRECTLY!
fig,ax=plt.subplots(1,1)
ax.set_xlim(0,4)
ax.set_ylim(0,8)
ax.plot(x,y,'ro')
ax.plot(x,B*x+A)
```

Out[78]:
```
[<matplotlib.lines.Line2D at 0x10cfc7690>]
```



Hmm. Something went wrong. This doesn't look like a very good fit!

Fix the cell above and run it again. (Hint: B isn't b... check the textbook again.)

What is the slope of the best-fit line? Does it make sense?

This was a fun trick question. The book gives equation 8.1

$$y = A + Bx$$

It also includes the warning, "Unfortunately, many different notations are used for a linear relation; beware of confusing the form (8.1) with the equally poular $y = ax + b$.$ which is the problem we had encountered.

In [ ]:

The uncertainties in the fit parameters A and B must be estimated given our best estimate for A,B, and the normal spread of y. We need to calculate the deviations of y using eq 8.15 from our textbook

$$\sigma_y = \sqrt{\frac{1}{N-2}\sum_{i=1}^{N}(y_i - A - Bxi)^2}$$

Then using this value we can calulate the uncertainties in A and B using our book's equations 8.16 and 8.17 respectively.

$$\sigma_A = \sigma_y \sqrt{\frac{\sum x^2}{\Delta}}$$

$$\sigma_B = \sigma_y \sqrt{\frac{N}{\Delta}}$$

In [79]:
```
def ySigma(x,y,A,B):
    return np.sum((y-A-B*x)**2)/(float(len(x))-2.)
```

In [80]:
```
def ASigma(x,sigma,delta):
    return sigma*np.sqrt(np.sum(x*x)/delta)
```

In [81]:
```
def BSigma(x,sigma,delta):
    return sigma*np.sqrt(float(len(x))/delta)
```

In [82]:
```
yUncertainty=ySigma(x,y,A,B)
AUncertainty=ASigma(x,yUncertainty,delta)
BUncertainty=BSigma(x,yUncertainty,delta)
print "The uncertainty in the y values given is "+str(yUncertainty)
print "The uncertainty in the intercept A is "+str(AUncertainty)
print "The uncertainty in the slope B is "+str(BUncertainty)
```

```
The uncertainty in the y values given is 0.0
The uncertainty in the intercept A is 0.0
The uncertainty in the slope B is 0.0
```

This is hardly surprising given our perfect values created for the sake of the excercise.

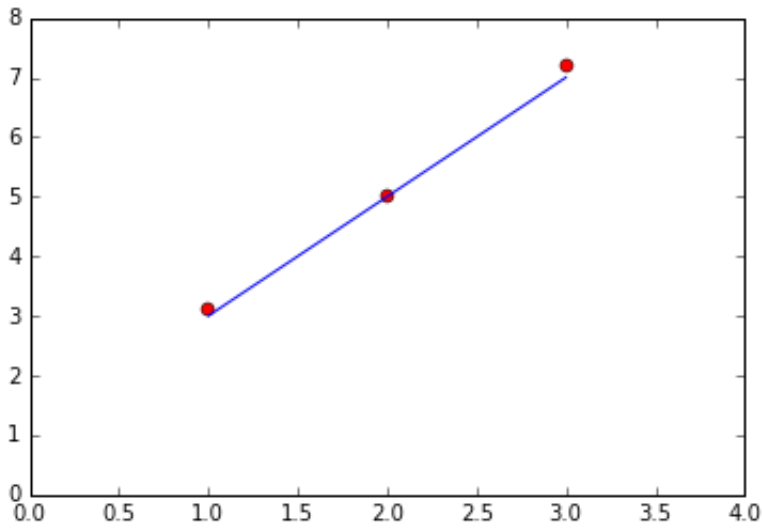
##Part 1b:##

Repeat the exercise above for slightly less obvious values. If you want to play around here, feel free!

In [83]:
```
x2=np.array([1,2,3])
y2=np.array([3.1,5,7.2])
delta2=findDelta(x2)
A2=findA(x,y)/delta2
B2=findB(x,y)/delta2
yUncertainty2=ySigma(x2,y2,A2,B2)
AUncertainty2=ASigma(x2,yUncertainty2,delta2)
BUncertainty2=BSigma(x2,yUncertainty2,delta2)
```

In [84]:
```
fig2,ax2=plt.subplots(1,1)
ax2.set_xlim(0,4)
ax2.set_ylim(0,8)
ax2.plot(x2,y2,'ro')
ax2.plot(x2,B2*x2+A2)
print "The uncertainty in the y values given is "+str(yUncertainty2)
print "The uncertainty in the intercept A is "+str(AUncertainty2)
print "The uncertainty in the slope B is "+str(BUncertainty2)
```

```
The uncertainty in the y values given is 0.05
The uncertainty in the intercept A is 0.0707106781187
The uncertainty in the slope B is 0.0353553390593
```



Part 2:

Now you are ready to find the best-fit value for the magnetic moment μ of the magnet in the pool ball from the last two weeks. Clearly indicate your data, explain the method, plot the data with the best-fit line, and report your value of μ with uncertainty. You will probably employ lots of cut-and-paste, but make sure that your write-up here is complete.

From the handout we know that:

$$\tau = r\mathrm{X}mg$$

and

$$\tau = \mu\mathrm{X}B$$

leading us to conclude that:

$$r \cdot m \cdot g \cdot \sin(\theta_A) = \mu \cdot B \cdot \sin(\theta_B)$$

Where θ_A is the angle made between the rod and the vertically descending gravitational force, and $theta_B$ is the angle created by the rod and the vertically ascending magnetic force, which are congruent angles and thus their sin values can be cancelled from either side. Then dividing accross we get:

$$B = \frac{r \cdot m \cdot g}{\mu}$$

in this case g, m, and μ are constants. I'm going to assume that g is $9.81 \frac{\mathrm{m}}{\mathrm{s}^2}$, the mass cauing tourque has been measured to be $1.25 \pm .05\mathrm{g}$. Thus creating a graph of B over r the slope generated will be the value $\frac{m \cdot g}{\mu}$.

My values as calculated previously are

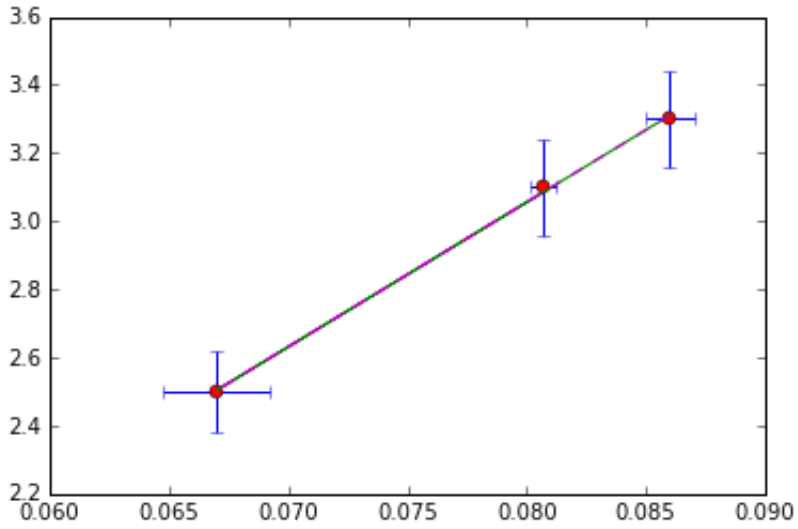
B	δB	r	δr
3.3	.14	86	1.05
2.5	.12	67	2.25
3.1	.14	80.7	0.55

In [87]:
```
m=.001125
g=9.81
magnetic_field_strength=np.array([3.3,2.5,3.1])
delta_magnetic_field_strength=np.array([0.14,0.12,0.14])
radius=np.array([.086,.067,.0807])
delta_radius=np.array([1.05,2.25,0.55])*1e-3
delta_function=findDelta(radius)
intercept=findA(radius,magnetic_field_strength)/delta_function
slope=findB(radius,magnetic_field_strength)/delta_function
```

In [88]:
```
fig3, ax3 =plt.subplots(1,1)
ax3.plot(radius,magnetic_field_strength,'ro')
ax3.errorbar(radius,magnetic_field_strength,xerr=delta_radius,yerr=delta_
#ax3.xlabel(r'radius from cm of ball to cm of weight [mm]')
ax3.plot(radius,slope*radius+intercept)
model=np.polyfit(radius,magnetic_field_strength,1)
print model[1]-intercept
print model[0]-slope
ax3.plot(radius,model[0]*radius+model[1],'m--')
```

```
-3.50275364269e-14
6.8212102633e-13
```

Out[88]:
```
[<matplotlib.lines.Line2D at 0x10d47df10>]
```



now that I have found the slope I can get a value for μ back from the slope. Remeber the slope of this graph is:

$$\frac{m \cdot g}{\mu} = B$$

Then a little algebra returns the expression

$$\mu = \frac{m \cdot g}{B} \pm \delta(\frac{m \cdot g}{B})$$

$\delta(\frac{m \cdot g}{B}$ is of course the least squares propogation of the independent values for m and B assuming that g in known with certainty to within the signiificanse of my measured variables. The uncertainty in B can be calculated from the function I created above. Thus:

$$\delta\mu = \sqrt{(\delta m)^2 + (\delta B)^2}$$

In [ ]:
```
y_uncertainty_function=ySigma(radius,magnetic_field_strength,intercept,sl
slope_uncertainty=BSigma(radius,delta_function,y_uncertainty_function)
mu=(m*g)/slope
delta_mu=np.sqrt((.05/1.25)**2+(slope_uncertainty/slope)**2)*mu
print 'mu equals {0} plus or minus {1}'.format(mu, delta_mu)
```

Thus we can se that the value for ther magnetic dipole moment:

$$\mu = (2.60 \pm 0.10) \cdot 10^{-4}$$

Part 3:

There are, of course, functions that will do this analysis for you. For example, check out (i.e. Google it then try it)

```
np.polyfit(x,y,1)
```

on one of the parts above. Compare and comment.

Thanks, I actually used ployfit last week. Ive added it in magenta to my plot for measured data, I've also used a magenta dotted line, so as not to cover up the line I created. This function is virtually the same as mine, and I checked the difference between this one and mine, they are the same down to 10^{-13} which is pretty good. One thing I've noticed that np.polyfir returns in the form $ax + b$ which is worth noting I suppose.

If you're really interested in functions that do this for you I whould reccomend looking into the glm(generalized linear model) function supplied by the rpy2 package, it goes into far more depth than just the coefficients. there are many other functions supplied by this packag, it's just an api that gives python acces to R. It doesn't work wity ipython notebook, at least not without installing R, but I checked on one of the computers in the astrolab and it works with getdatajoy.com. At that point you could just use R though. Speaking of which R is an open source language made special for analyzing data, I personally think you should use it as one of the other tools this class delves into.

In [ ]: