Error Analysis Exercise (EAX)

Oliver Gorton

March 27, 2016

1 First problem

We want to attain a specific activity with a precision of 1% from measurements at a rate of 1000 decays per 5 minutes of observation. The accuracy of an average value such as this is better by a factor of

$$\sqrt{N}. \tag{1}$$

This means that a precision P:

$$P = \sqrt{N}/N \tag{2}$$

is aquired when N samples are taken. If we are looking for a precision of 1% then we can calculate the require number of samples:

$$\begin{aligned} P = 0.01 = \sqrt(N)/N \rightarrow \\ N = 10,000 \ samples. \end{aligned} \tag{3}$$

In order to gather this many samples at the prescribed rate of decays (1000 decays per 5 minutes), we must measure for approximately:

$$\begin{aligned} Time \ to \ sample &= \frac{10,000 \ samples}{1,000 \ samples/5 \ min.} \\ &= 50 \ minutes. \end{aligned} \tag{4}$$

2 Second Problem

We are given two measurements and their associated uncertainties:

$$A \pm \sigma_A \tag{5}$$

$$B \pm \sigma_B \tag{6}$$

Often we want to calculate the error associated with the combination of two or more quantities for which an uncertainty is known. The genral expression for the uncertainty of a quantity with functional dependence on several other quantities is:

$$\sigma_f^2 = \sum_{i=1}^{n} (\frac{\partial f}{\partial x_i})^2 \sigma_i^2. \tag{7}$$

For each of the following combinations of A and B, the corresponding error of the resulting quantity is calculated (the propagated error).

2.1 A+B

$$\sigma_{A+B}^2 = (1)^2\sigma_A^2 + (1)^2\sigma_B^2 \tag{8}$$

$$\sigma_{a+b} = \sqrt{\sigma_A^2 + \sigma_B^2} \tag{9}$$

This so called propagated error is used in the following way:

$$\begin{aligned}(A \pm \sigma_A) + (B \pm \sigma_B) &= (A + B) \pm \sigma_{a+b} \\ &= (A + B) \pm \sqrt{\sigma_A^2 + \sigma_B^2}\end{aligned} \tag{10}$$

2.2 A-B

$$\sigma_{A-B}^2 = (1)^2\sigma_A^2 + (-1)^2\sigma_B^2 \tag{11}$$

$$\sigma_{A-B} = \sqrt{\sigma_A^2 + \sigma_B^2} \tag{12}$$

2.3 2A+2B

$$\sigma_{2A+2B}^2 = (2)^2\sigma_A^2 + (2)^2\sigma_B^2 \tag{13}$$

$$\sigma_{2A+2B} = \sqrt{4\sigma_A^2 + 4\sigma_B^2} \tag{14}$$

2.4 A x B

$$\sigma_{A\times B}^2 = (\sigma_B)^2\sigma_A^2 + (\sigma_A)^2\sigma_B^2 \tag{15}$$

$$\sigma_{A\times B} = \sqrt{2\sigma_A^2\sigma_B^2} \tag{16}$$

3 Third Problem

In this problem we are examining lists of normal distribution random numbers. My subscription to MATLAB has expired, so the rest of the problems will be solved using Python.

I used in this problem the Python package NumPy which is equivilent to the MATLAB randn functions. The theoretical standard deviation and mean of this psuedo random number generator are 1 and 0, respectively.

1. If we sample 5 times ($N = 5$) then we expect the mean to be $1/\sqrt{5}$.

3.1 Code solution

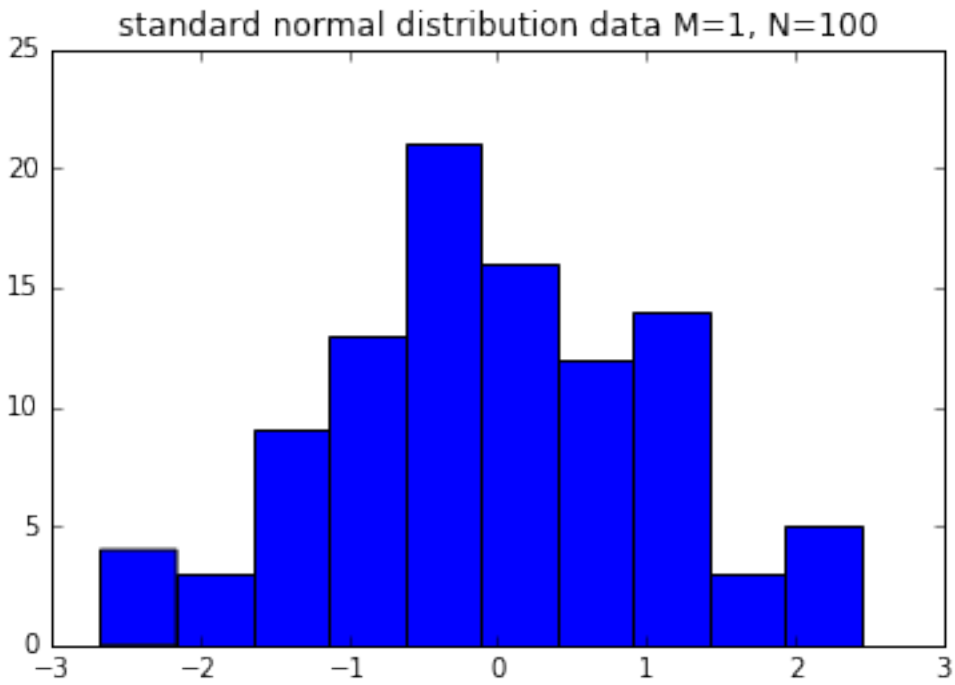
```
#Exercise 3
import numpy as np
import scipy as sp
from scipy import stats
import matplotlib.pyplot as plt
%matplotlib inline
```

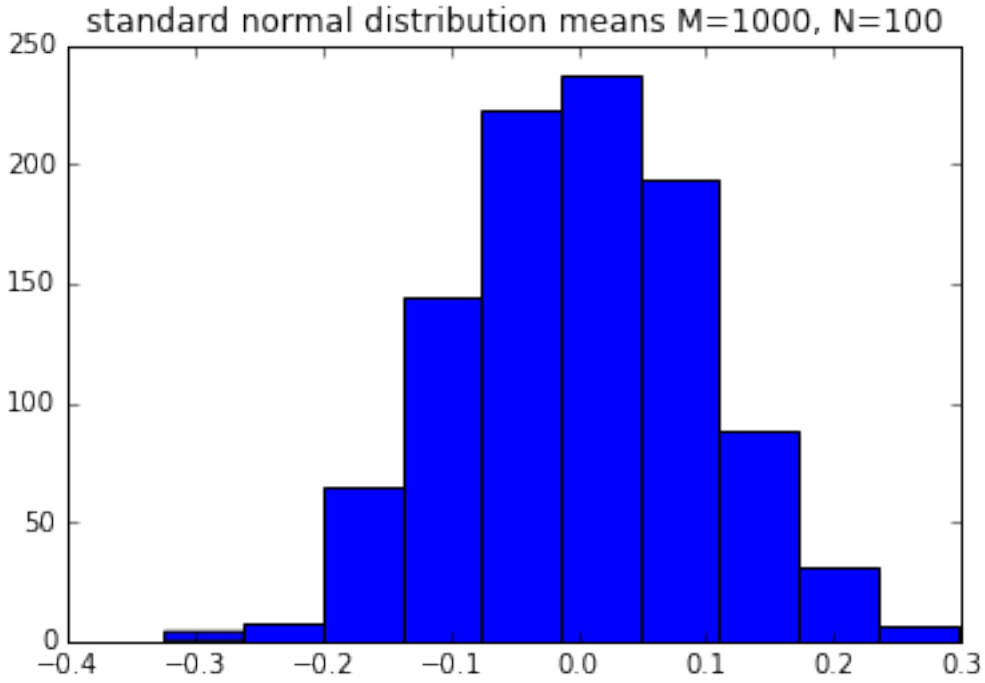
```
print'parts 1-5: '
#1.
x=np.random.standard_normal(100)
#2.
plt.hist(x)
plt.title('standard normal distribution data M=1, N=100')
#3.
print 'mean: ', sp.mean(x)
print 'Standard Deviation: ', np.std(x)
mean_theory=0
print 'error on the mean: ', sp.mean(x)-mean_theory
experr=sp.std(x)/np.sqrt(100)
print 'we expect the error on the mean to be std/sqrt(N): ', experr
print 'as you can see these the result is close to expectation.'
#4.
print 'median of the distribution: ', np.median(x)
#5.
hist1,bins=np.histogram(x)
mode_loc=np.argmax(hist1)
print 'mode: ', x[mode_loc]
print ' '
print 'part six: '
#6\
N=100
i=0
means=[]
stds=[]
errs=[]
while i<1000:
    x=np.random.standard_normal(N)
    means.append(np.mean(x))
    stds.append(sp.std(x))
    errs.append(sp.std(x)/np.sqrt(N))
    i+=1

plt.figure()
plt.hist(means)
plt.title('standard normal distribution means M=1000, N=100')
#7
Ns=[10,50,1000,10000]
RMSmeans=[]
for elt in Ns:
    N=elt
    i=0
    means=[]
```

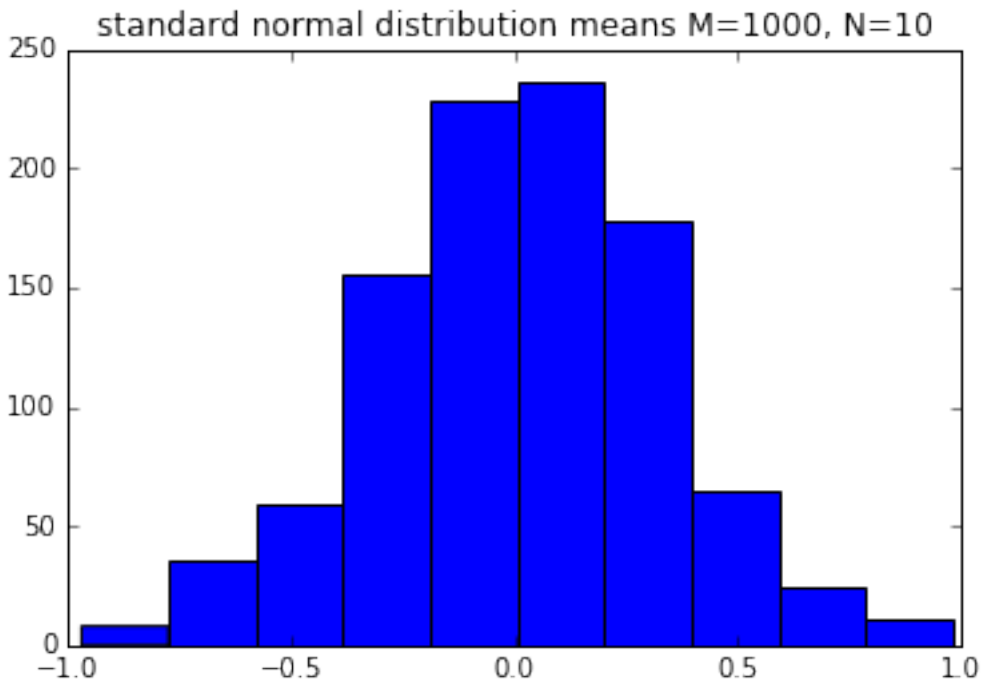
```
    stds=[]
    errs=[]
    while i<1000:
        x=np.random.standard_normal(N)
        means.append(np.mean(x))
        stds.append(sp.std(x))
        errs.append(sp.std(x)/np.sqrt(N))
        i+=1
    RMSmeans.append(np.std(means))
    plt.figure()
    plt.hist(means)
    plt.title('standard normal distribution means M=1000, N=%i' %N)

plt.figure()
plt.plot(Ns, (RMSmeans), 'o')
plt.xlabel('N')
plt.xlim(0,10500)
plt.ylabel('RMS of means')
plt.title('RMS of the distribution of the Means as a function of N')
#We see that as the number of samples increases, the error on the mean
#goes down like 1/sqrt(N)
#This is reflected in the narrowing of the histograms and in the plot of
#RMS of mean versus N
```

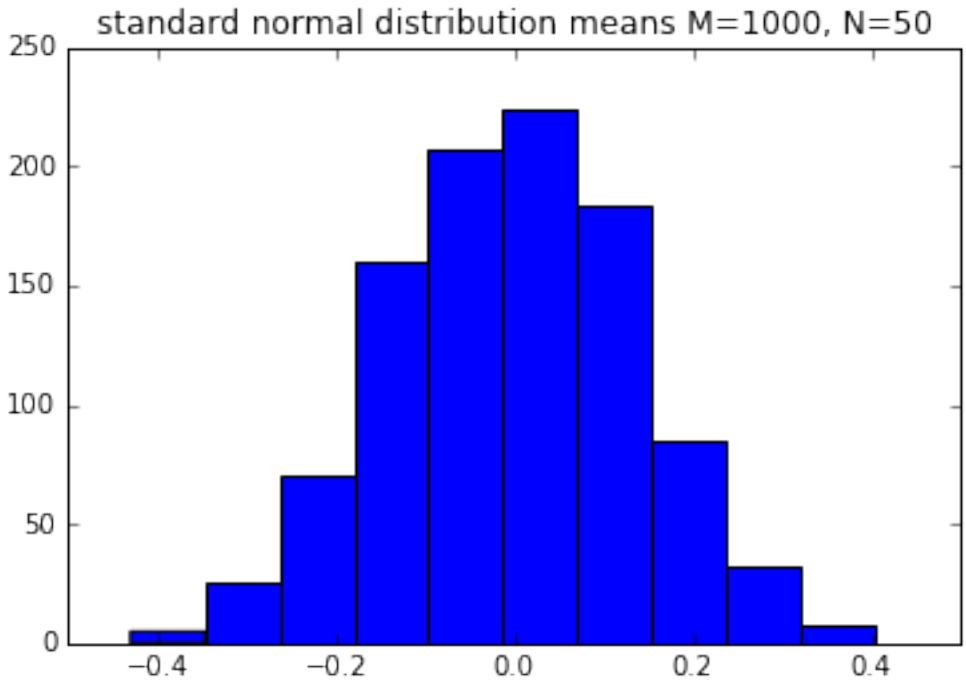
3.2 Code output

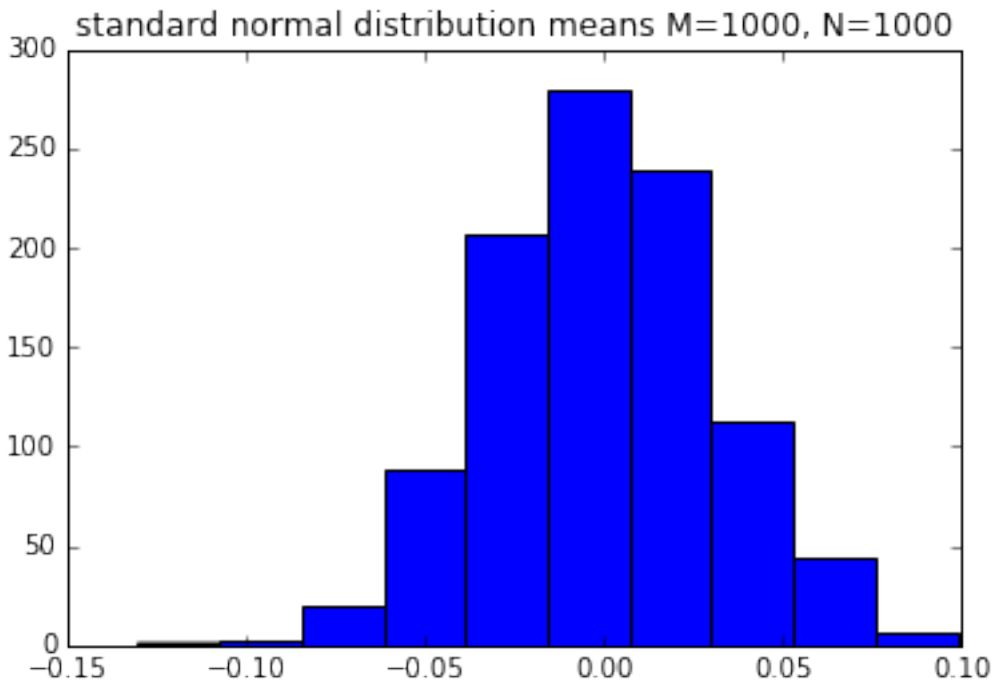
```
mean:  -0.0573312415722
Standard Deviation:  1.09812210642
error on the mean:  -0.0573312415722
we expect the error on the mean to be std/sqrt(N):  0.109812210642
as you can see these the result is close to expectation.
median of the distribution:  -0.115846132088
mode:  -1.12205307095

part six:
```

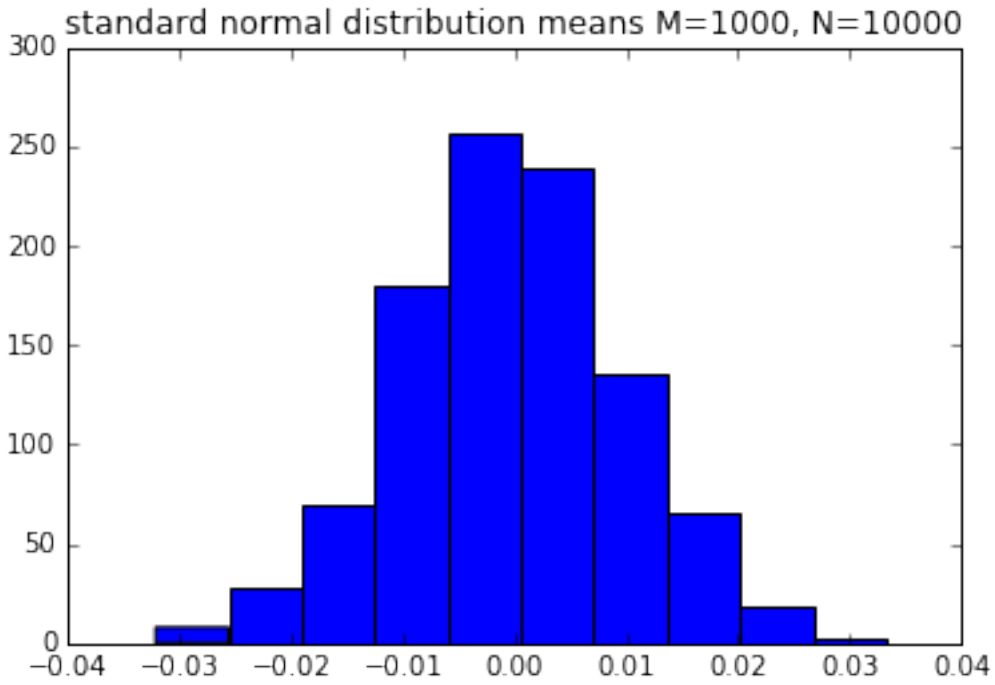

standard normal distribution data M=1, N=100
25
20
15
10
5
0
−3
−2
−1
0
1
2
3

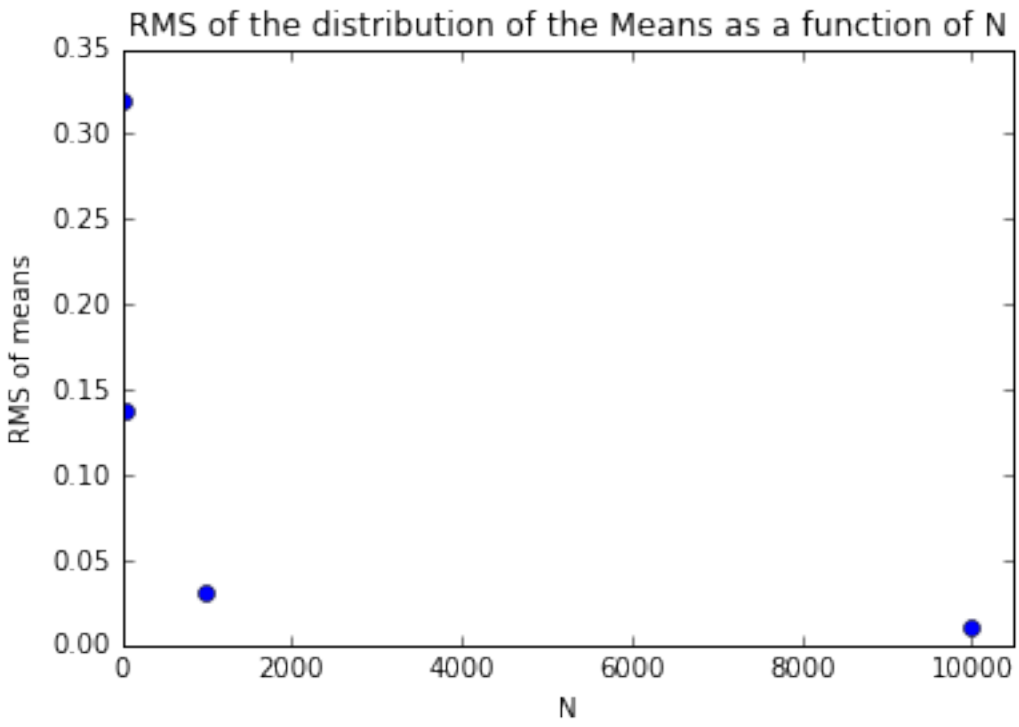

standard normal distribution means M=1000, N=100
250
200
150
100
50
0
−0.4
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3


standard normal distribution means M=1000, N=10
250
200
150
100
50
0
−1.0
−0.5
0.0
0.5
1.0


standard normal distribution means M=1000, N=50
250
200
150
100
50
0
−0.4
−0.2
0.0
0.2
0.4


standard normal distribution means M=1000, N=1000
300
250
200
150
100
50
0
−0.15
−0.10
−0.05
0.00
0.05
0.10


standard normal distribution means M=1000, N=10000
300
250
200
150
100
50
0
−0.04
−0.03
−0.02
−0.01
0.00
0.01
0.02
0.03
0.04



4 Fourth problem

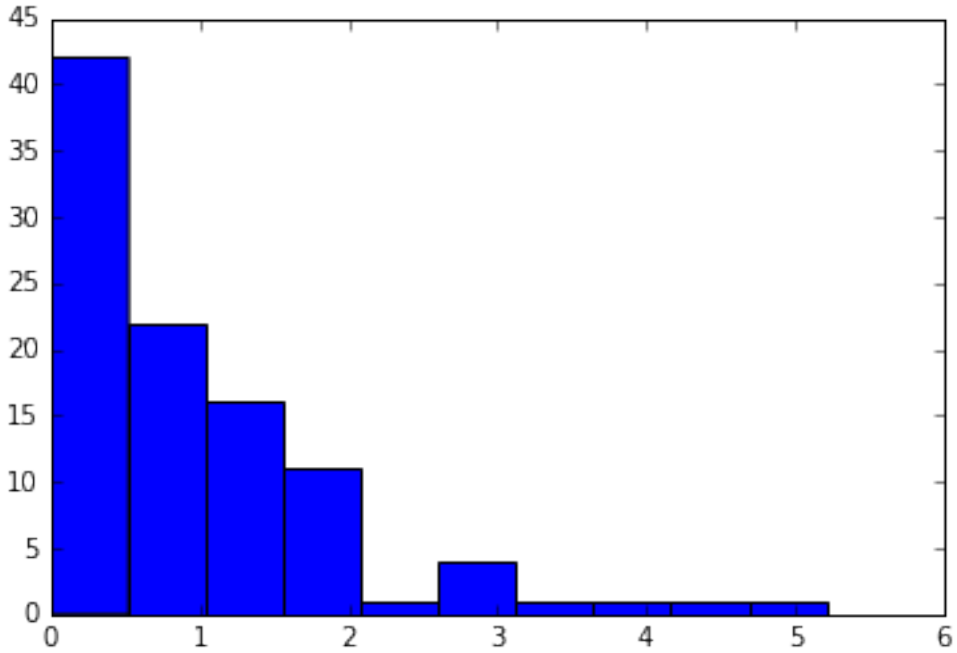
4.1 Code solution to problem

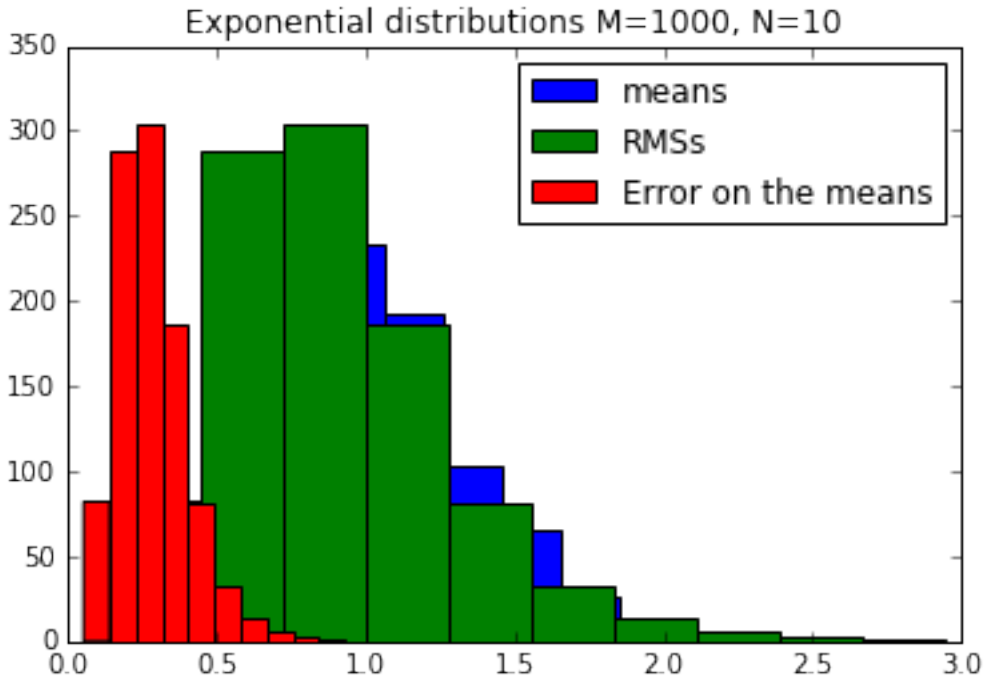
```
import numpy as np
import scipy as sp
from scipy import stats
import matplotlib.pyplot as plt
%matplotlib inline

#1. I expect the mean of the exponential distribution to be 1,
#and for the

#2.
a = np.random.exponential(size=100)
#3.
plt.hist(a)
#4.
print 'M=0, N=100'
print 'mean: ', np.mean(a)
print 'standard deviation: ', np.std(a)
print 'error on the mean: ',np.mean(a)-1
```

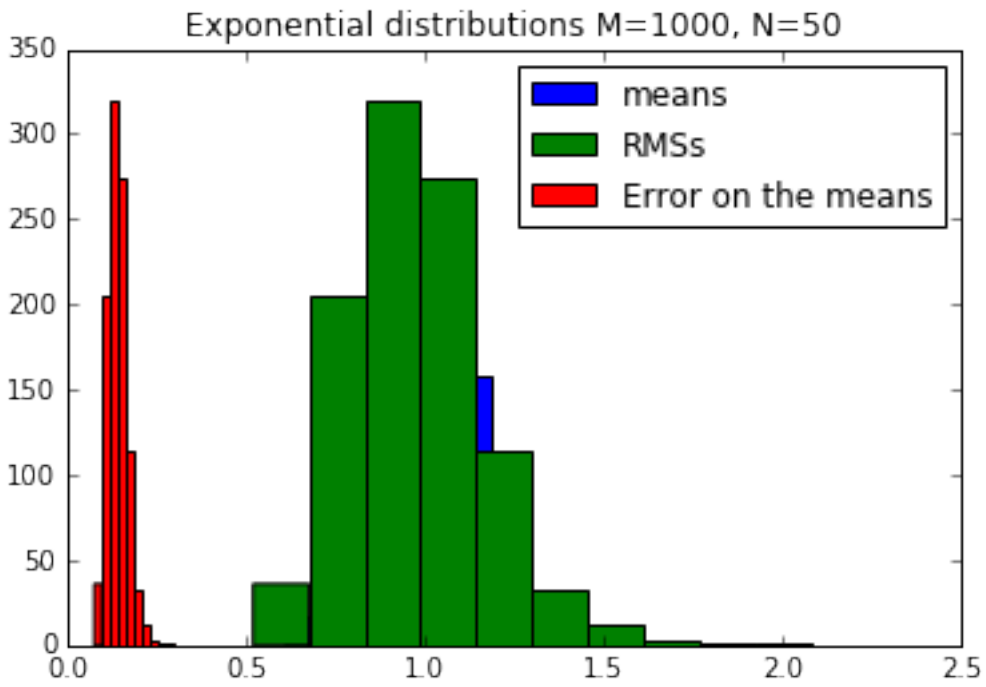
```
#5-6 combined:
Ns=[10,50,100,1000,10000]
RMSmeansExp=[]
for elt in Ns:
    N=elt
    i=0
    means=[]
    stds=[]
    errs=[]
    while i<1000:
        x=np.random.exponential(size=N)
        means.append(np.mean(x))
        stds.append(sp.std(x))
        errs.append(sp.std(x)/np.sqrt(N))
        i+=1
    RMSmeansExp.append(np.std(means))
    plt.figure()
    plt.hist(means, label='means')
    plt.title('Exponential distributions M=1000, N=%i' %N)

    plt.subplot()
    plt.hist(stds, label='RMSs')

    plt.subplot()
    plt.hist(errs, label='Error on the means')
    plt.legend()
plt.figure()
plt.plot(Ns, (RMSmeansExp), 'o')
plt.xlabel('N')
plt.xlim(0,10500)
plt.ylabel('RMS of means')
plt.title('RMS of the distribution of the Means as a function of N')
```

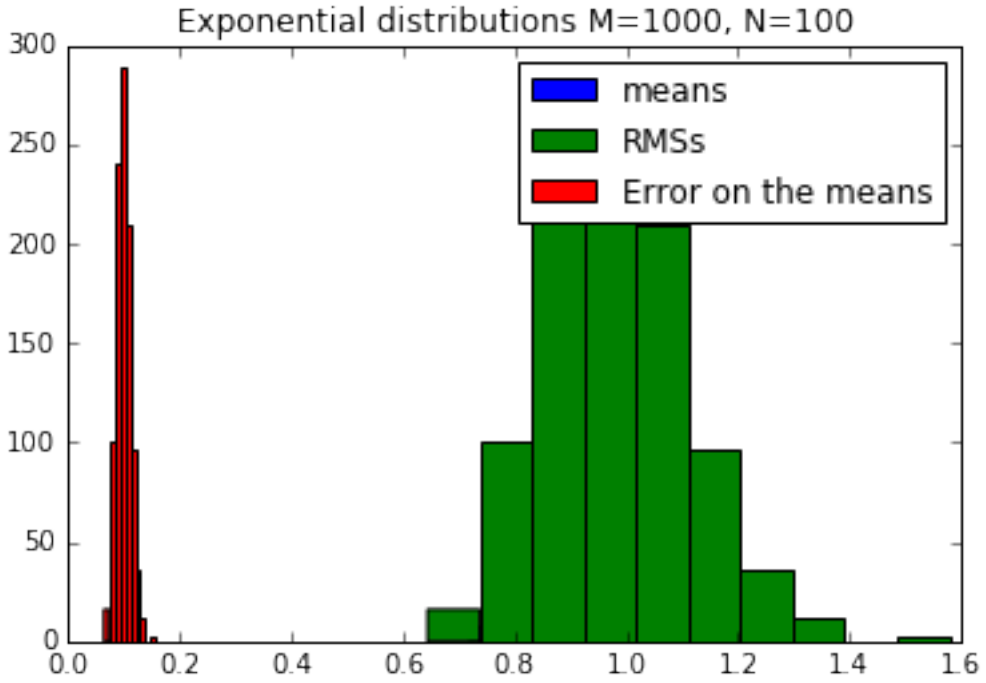
4.2 Output of code

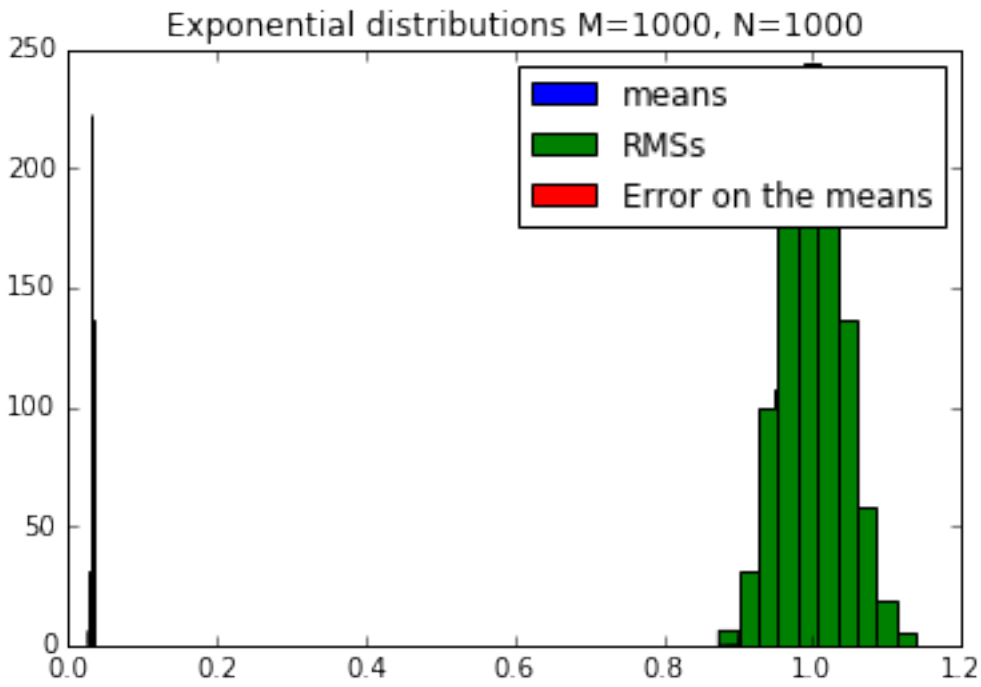
```
M=0, N=100
mean:  0.963302376995
standard deviation:  0.966064557611
error on the mean:  -0.0366976230052
```

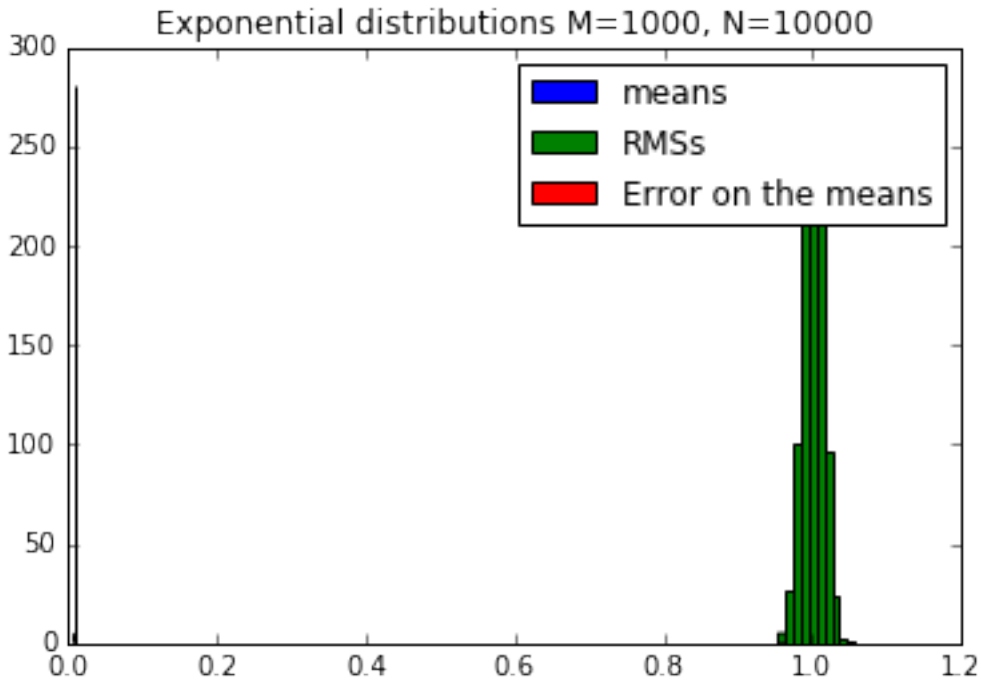

45
40
35
30
25
20
15
10
5
0
0
1
2
3
4
5
6

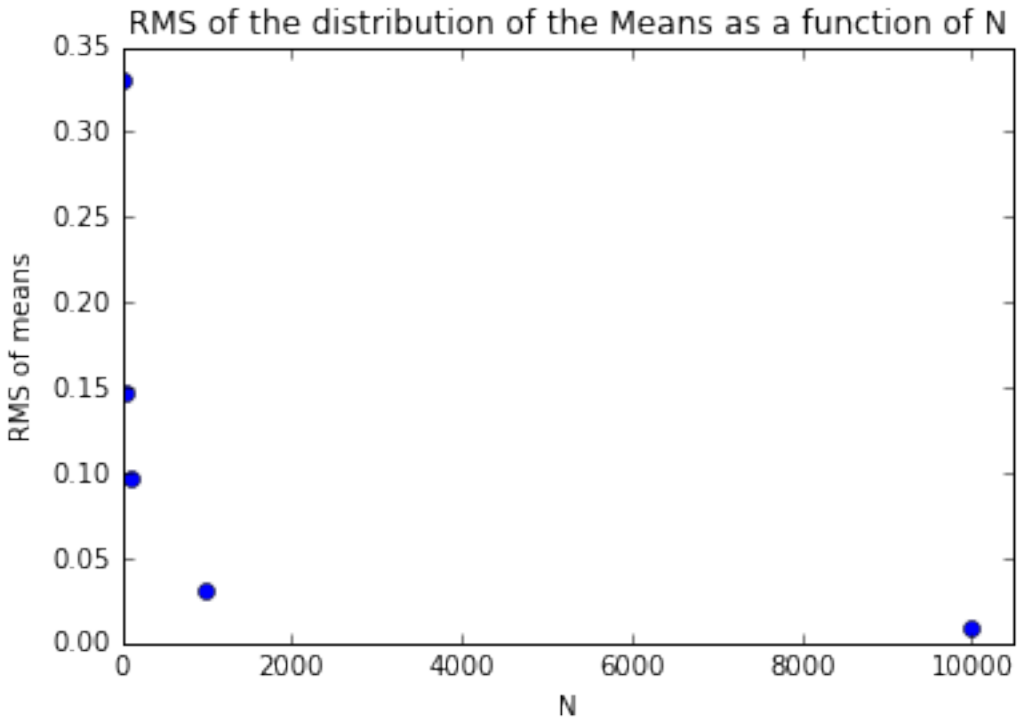

Exponential distributions M=1000, N=10
means
RMSs
Error on the means
350
300
250
200
150
100
50
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0


Exponential distributions M=1000, N=50
means
RMSs
Error on the means
350
300
250
200
150
100
50
0
0.0
0.5
1.0
1.5
2.0
2.5


Exponential distributions M=1000, N=100
means
RMSs
Error on the means
300
250
200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6


Exponential distributions M=1000, N=1000
means
RMSs
Error on the means
250
200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2


Exponential distributions M=1000, N=10000
means
RMSs
Error on the means
300
250
200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2



5 Problem 5

5.1 Code solution

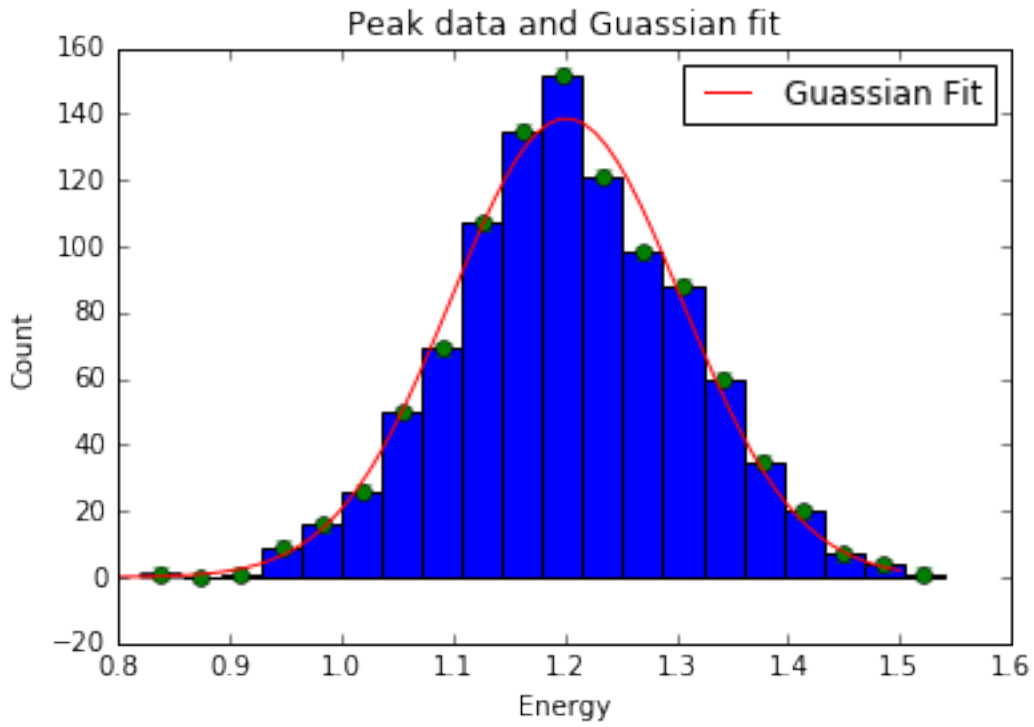
```
Gamma-ray peak
import scipy.optimize as fitter
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
%matplotlib inline

#Loading and plotting data and printing characteristic info.:
peak=np.loadtxt('peak.dat')
plt.hist(peak, bins=20)
mean=np.mean(peak)
std=np.std(peak)
N=len(peak)
meansig=std/np.sqrt(N)
stdsig=std/np.sqrt(2*N)
print 'mean: {0:5.4f}'.format(mean), '+/- {0:5.4f}'.format(meansig)
print 'standard devation: {0:5.4f}'.format(std), '+/- {0:5.4f}'.format(stdsig)
#creating variables with bin values and locations of centers of bins
y,edges=np.histogram(peak, bins=20)
print 'y: ', y
```

```
x=[]
for i in range(0,len(edges)-1):
    x.append((edges[i]+edges[i+1])/2.)
#Definging a Guassian model:
def model(x,a,b,c):
    return a*np.exp((-((x-b)*(x-b))/(2*c*c)))
#Using scipy model fitting function given chosen initial guesses:
par0=np.array([150,1.2,0.1])
par,cov=fitter.curve_fit(model,x,y,par0,sigma=None)
#Extracting results from the output:
a = par[0]
ea = np.sqrt(cov[0,0])
print 'a={0:6.3f}+/-{1:5.3f}'.format(a,ea)
b = par[1]
eb = np.sqrt(cov[1,1])
print 'b={0:6.3f}+/-{1:5.3f}'.format(b,eb)
c=par[2]
ec=np.sqrt(cov[2,2])
print 'c={0:6.3f}+/-{1:5.3f}'.format(c,ec)
#Computing Chi2 and reduced Chi2 values:
sig=2
sigma = np.ones(20)*sig
print 'model: ',model(x, par[0],par[1],par[2])
chi_squared = np.sum(((model(x, par[0],par[1],par[2])-y)/sigma)**2)
reduced_chi_squared = (chi_squared)/(len(x)-len(par))
print 'chi^2 = {0:5.2f}'.format(chi_squared)
print 'chi^2/d.f.={0:5.2f}'.format(reduced_chi_squared)
#Plotting the model:
plt.title('Peak data and Guassian fit')
plt.xlabel('Energy')
plt.ylabel('Count')
plt.errorbar(x, y, xerr=0, yerr=sigma, fmt='o')
plt.xlim(.8,1.6)
xfit = np.linspace(.5,1.5,100)
plt.plot(xfit,model(xfit,par[0],par[1],par[2]),'r-', label='Guassian Fit')
plt.legend()
plt.show()
#The Guassian appears to fit well past 1 sigma from the mean, and not as well
#towards the center of the distrbution.
```

5.2 Code output

```
mean: 1.2027 +/- 0.0033
standard devation: 0.1038 +/- 0.0023
y:  [  1   0   1   9  16  26  50  69 107 135 152 121  98  88  60  35  20   7
```

```
   4   1]
a=138.621+/-3.421
b= 1.201+/-0.003
c= 0.103+/-0.003
model:  [   0.28233146    0.90687098    2.57934212    6.49605382   14.48663427
   28.60635974   50.01898946   77.44340662  106.17229422  128.8887978
  138.54675607  131.87266057  111.14511838   82.94748737   54.81422713
   32.07455007   16.61900692    7.6247706     3.09760269    1.11429931]
chi^2 = 168.53
chi^2/d.f.= 9.91
```



6 Sixth Problem

6.1 Code solution

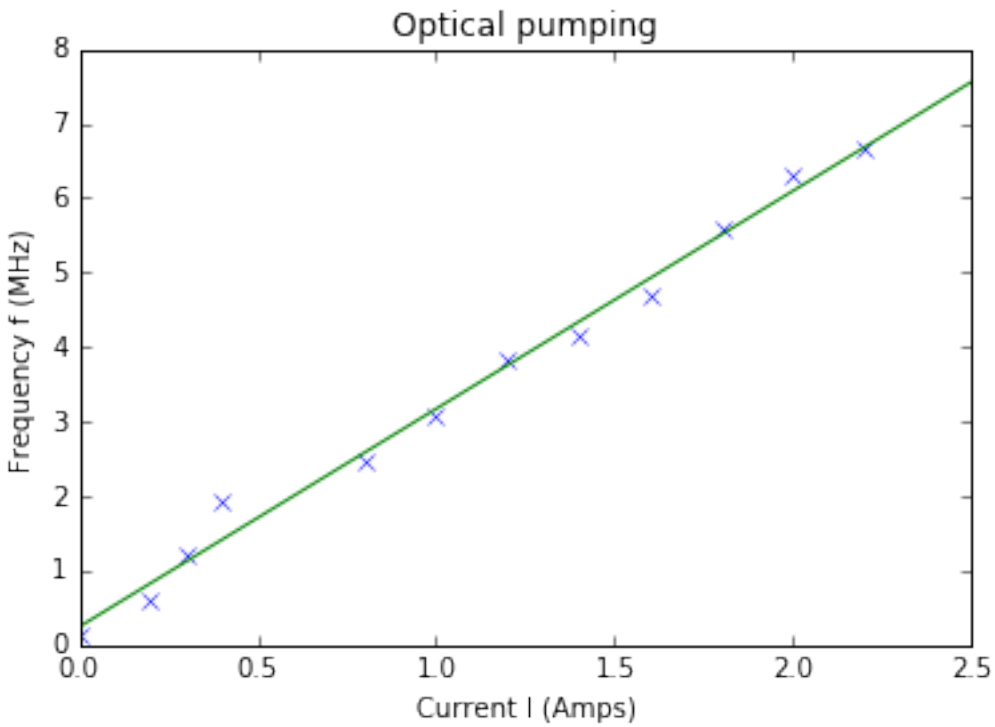
```
import scipy.optimize as fitter
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from scipy.stats import chi2
%matplotlib inline
I=[0.0,.2,.3,.4,.8,1.0,1.2,1.4,1.6,1.8,2.0,2.2]
f=(0.14,0.6,1.21,1.94,2.47,3.07,3.83,4.16,4.68,5.60,6.31,6.67)
plt.plot(I,f,'x',label='Data')
```

```
def model2(x, a, b):
    return a+b*x
par0    = np.array([0.0, 3.0])
par, cov = fitter.curve_fit(model2, I,f, par0,absolute_sigma=True)
#absolut_sigma True to get equal weighting.
a = par[0]
ea = np.sqrt(cov[0,0])
print 'a={0:6.1f}+/-{1:5.1f}'.format(a,ea)
b = par[1]
eb = np.sqrt(cov[1,1])
print 'b={0:6.1f}+/-{1:5.1f}'.format(b,eb)
xfit = np.linspace(0,2.5,12)
plt.plot( xfit, model2(xfit,a,b) , '-')
plt.title('Optical pumping')
plt.ylabel('Frequency f (MHz)')
plt.xlabel('Current I (Amps)')
plt.show()
#2
#Computing Chi2 and reduced Chi2 values:
sig=0.01
print 'sigma: ', sig
sigma = np.ones(12)*sig
chi_squared =0
for i in range(0,len(I)):
    chi_squared+=(((model2(float(I[i]), float(par[0]),float(par[1]))-f[i])/sigma[i])**2)
reduced_chi_squared = (chi_squared)/(len(xfit)-len(par))
print 'chi^2 = {0:5.2f}'.format(chi_squared)
print 'chi^2/d.f.={0:5.2f}'.format(reduced_chi_squared)
p=1-chi2.cdf(chi_squared,2)#probability that a randon chi2 would be greater than our chi
print 'probability that the straight line we found is an adequate description: ', p
#So this is a very poor fit given that the uncertainty in the measurements was 0.01.
#3
sig=1
print 'sigma: ', sig
sigma = np.ones(12)*sig
chi_squared =0
for i in range(0,len(I)):
    chi_squared+=(((model2(float(I[i]), float(par[0]),float(par[1]))-f[i])/sigma[i])**2)
reduced_chi_squared = (chi_squared)/(len(xfit)-len(par))
print 'chi^2 = {0:5.2f}'.format(chi_squared)
print 'chi^2/d.f.={0:5.2f}'.format(reduced_chi_squared)
p=1-chi2.cdf(chi_squared,2) #probability that a randon chi2 would be greater than our ch
print 'probability that the straight line we found is an adequate description: ', p
#This uncertainty tells us that the fit is far more reasonable than previously thought.
#RMS of deviation errors on the parameters are found by:
```

```
perr=np.sqrt(np.diag(cov))
print r'a-uncertainty: ', round(perr[0],1)
print r'b-uncertainty: ' ,round(perr[1],1),'as calculated above by the same method.'

def sig(f):
    return 0.03+0.03*f
sigmas=[]
for elt in f:
    sigmas.append(sig(elt))
par,cov=fitter.curve_fit(model2,I, f, par0, sigma=sigmas,absolute_sigma=False)
print 'Here are the intercepts are their uncertainties assuming the uncertainty',
print '\n for each value goes like 0.03+0.03f'
a = par[0]
ea = np.sqrt(cov[0,0])
print 'a={0:6.1f}+/-{1:5.1f}'.format(a,ea)
b = par[1]
eb = np.sqrt(cov[1,1])
print 'b={0:6.1f}+/-{1:5.1f}'.format(b,eb)
```

6.2 Code output

```
a=   0.2+/-  0.5
b=   2.9+/-  0.4

sigma:  0.01
chi^2 = 5228.95
chi^2/d.f.=522.90
probability that the straight line we found is an adequate description:  0.0
sigma:  1
chi^2 =  0.52
chi^2/d.f.= 0.05
probability that the straight line we found is an adequate description:  0.769936184834
a-uncertainty:  0.5
b-uncertainty:  0.4 as calculated above by the same method.
Here are the intercepts are their uncertainties assuming the uncertainty
 for each value goes like 0.03+0.03f
a=   0.2+/-  0.1
b=   3.0+/-  0.1
```


Optical pumping
Frequency f (MHz)
Current I (Amps)
0
1
2
3
4
5
6
7
8
0.0
0.5
1.0
1.5
2.0
2.5