Least squares fit to a straight line

Derivation of the least-squares formulae for slope and intercept

Suppose we want to fit a dependent data set y_i to an independent set x_i according to the model

$$y = mx + b$$

In the following, we will assume that all of the y values *have the same uncertainty* measured by a standard deviation σ_y.

For trial values of slope m and intercept b we'd like to minimize the difference between the predicted value y from the measured value y_i for all of the values of x_i. We call these differences the "residuals" and define a measure χ^2 for the goodness of fit for a given choice of parameters:

$$\chi^2 = \sum_{i=1}^{N} \frac{1}{\sigma_{yi}^2}[y_i - (mx_i + b)]^2$$

Note that this is a weighted sum of the squares of the residuals divided by the squares of data uncertainties σ_{yi} that depend on which data value y_i is used.

We get the best fit for a minimum of the above quantity (hence the name "least squares"), which requires the derivatives with respect to m and b to vanish.

$$\frac{\partial}{\partial m}\chi^2 = 0.$$

$$\frac{\partial}{\partial b}\chi^2 = 0.$$

Applying these derivatives to the definition of χ^2:

$$\sum_{i=1}^{N} \frac{1}{\sigma_{yi}^2} 2[y_i - (mx_i + b)](-x_i) = 0.$$

$$\sum_{i=1}^{N} \frac{1}{\sigma_{yi}^2} 2[y_i - (mx_i + b)](-1) = 0.$$

Rearranging, we have two equations in two unknowns m and b:

$$\left(\sum_{i=1}^{N} \frac{x_i^2}{\sigma_{yi}^2}\right) m + \left(\sum_{i=1}^{N} \frac{x_i}{\sigma_{yi}^2}\right) b = \sum_{i=1}^{N} \frac{x_i y_i}{\sigma_{yi}^2}.$$

$$\left(\sum_{i=1}^{N} \frac{x_i}{\sigma_{yi}^2}\right) m + \left(\sum_{i=1}^{N} \frac{1}{\sigma_{yi}^2}\right) b = \sum_{i=1}^{N} \frac{y_i}{\sigma_{yi}^2}.$$

Define weighted averages as follows:

$$< x >\equiv \frac{\tilde{\sigma_y}^2}{N} \sum_{i=1}^{N} \frac{x_i}{\sigma_{yi}^2} \left(= \frac{1}{N} \sum_{i=1}^{N} x_i \text{if all } \sigma_{yi} \text{ are equal}\right).$$

$$< y >\equiv \frac{\tilde{\sigma_y}^2}{N} \sum_{i=1}^{N} \frac{y_i}{\sigma_{yi}^2} \left(= \frac{1}{N} \sum_{i=1}^{N} y_i \text{if all } \sigma_{yi} \text{ are equal}\right).$$

$$< x^2 >\equiv \frac{\tilde{\sigma_y}^2}{N} \sum_{i=1}^{N} \frac{x_i^2}{\sigma_{yi}^2} \left(= \frac{1}{N} \sum_{i=1}^{N} x_i^2 \text{if all } \sigma_{yi} \text{ are equal}\right).$$

$$< xy >\equiv \frac{\tilde{\sigma_y}^2}{N} \sum_{i=1}^{N} \frac{x_i y_i}{\sigma_{yi}^2} \left(= \frac{1}{N} \sum_{i=1}^{N} x_i y_i \text{if all } \sigma_{yi} \text{ are equal}\right).$$

Here

$$\frac{1}{\tilde{\sigma_y}^2} \equiv \frac{1}{N} \sum_{i=1}^{N} \frac{1}{\sigma_{yi}^2} \left(= \frac{1}{\sigma_y^2} \text{ if all } \sigma_{yi} \text{ are equal to the same value } \sigma_y\right).$$

$$\tilde{\sigma_y}^2 \equiv \frac{N}{\sum_{i=1}^{N} \frac{1}{\sigma_{yi}^2}} \left(= \sigma_y^2 \text{ if all } \sigma_{yi} \text{ are equal to the same value } \sigma_y\right).$$

After multiplying through by a common factor of $\tilde{\sigma_y}^2$, the equations for slope and intercept become

$$< x^2 > m+ < x > b =< xy >.$$

$$< x > m + b =< y >.$$

Using this notation, solve the second of the simultaneous equations for b:

$$b =< y > -m < x >.$$

This seems to be an intutively plausible result.

Substitute this value for b into the first equation to get:

$$< x^2 > m+ < x > [< y. -m < x >] =< xy >.$$

Solve for m:

$$m = \frac{< xy > - < x >< y >}{< x^2 > - < x >^2}.$$

Once we have m, we simply use this in the expression above to calculate b:

$$b =< y > -m < x >.$$

Uncertainties in the slope and intercept estimates

*The following results can be derived through propagation of errors in the expressions for m and b above - after explicily writing out the averages as sums and taking derivatives with respect to the uncertain quantities y_i.

(1) The estimated uncertainty in the slope is given by

$$\sigma_{m\_est} = \frac{\tilde{\sigma_y}}{\sqrt{N}} \sqrt{\frac{1}{< x^2 > - < x >^2}}.$$

(2) The estimated uncertainty in the intercept is given by

$$\sigma_{b\_est} = \frac{\tilde{\sigma_y}}{\sqrt{N}} \sqrt{\frac{< x^2 >}{< x^2 > - < x >^2}} = \sigma_{m\_est} \sqrt{< x^2 >}.$$

An interesting feature of these results is that if, from one experimental run to the next, the same set of values of the independent variable x are used, then the uncertainties in slope and intercept will only vary because of variations in the set of *uncertainties* of y, not because of variations in the y values themselves. This is demonstrated in the numerical exploration below when doing weighted fits with specified uncertainties in y.

Again, if all of the uncertainties in y are assumed to be the same value σ_y, then we replace $\tilde{\sigma_y}$ with σ_y in the above expressions. If the uncertainty σ_y is not known *a priori* then it can be estimated by

$$\sigma_{y\_est} \approx \sqrt{\frac{1}{N-2} \sum_{i=1}^{N} [y_i - (mx_i + b)]^2},$$

where m and b are the estimated values of slope and intercept. When this estimate is used, the uncertainties in slope and intercept will vary from run to run as the set of values of y used in the estimate varies.

References

https://www.asc.ohio-state.edu/gan.1/teaching/spring04/Chapter7.pdf (https://www.asc.ohio-state.edu/gan.1/teaching/spring04/Chapter7.pdf)

Reference: see P. R. Bevington & D. K. Robiinson, *Data Reduction and Error Analsyis for the Physical Sciences, 3rd ed.,* (McGraw-Hill, 2003) Chapter 6.

Code to do linear regression in several ways

The following approaches are used to do a linear regression for a small data set.

If you want to use the code, edit the variables:

```
my-name
xdata (an array)
ydata (an array)
sigma_ydata (an array of uncertainty estimates for variable y)
```

The code then calculates fitting parameters m and b for the linear model $y = mx + b$ along with uncertainties in the slope using the following methods:

- explicit calculation using formulas from the above derivations but without weighting by the individual point variance estimates
- same as above but with weighting
- unweighted fit using the module linregress from the scipy.stats library
- unweighted and weighted fits using the module curve_fit from the scipy.optimize module

You should see that the unweighted results all agree and separately how the weighted results agree.

The best fit lines (weighted and unweighted) are then plotted through the data points, which have error-bars.

In [15]:
```
'''
multLS.py
Perferms least squares fit multiple ways, both unweighted and weight
Randall Tagg 07-Oct-2020
'''

import math
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import linregress
from scipy.optimize import curve_fit
import datetime

################# EDIT NAME AND DATA ARRAYS HERE ##############
```

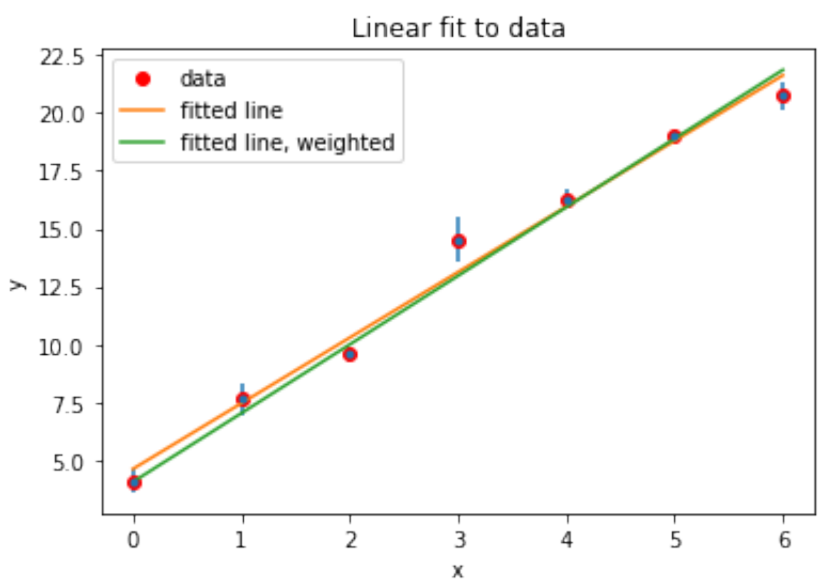
```
my_name = "Randall Tagg"

#Enter x data
xdata = np.asarray([0.,1.,2.,3.,4.,5.,6.])
Ndata = xdata.size

#Enter y data
ydata = np.asarray([4.111,7.632,9.563,14.530,16.269,19.003,20.752])

#Enter standard deviations of y data
sigma_ydata = np.asarray([0.5,0.7,0.3,1.0,0.4,0.2,0.6])
#Uncomment the following two lines and edit if needed for
# uniform weighting
#sigmaScaleFactor = 1.
#sigma_ydata = sigmaScaleFactor*np.asarray([1.,1.,1.,1.,1.,1.,1.])


# Create a time stamp for each run
time_stamp_format = "%Y-%m-%d %H:%M:%S"
print(my_name,datetime.datetime.now().strftime(time_stamp_format),'\
print('x ',xdata)
print('y',ydata)
print('sigma_ydata',sigma_ydata)

######################################################


def flinear(xx,mm,bb):  # define the linear function (for use with t
    return mm*xx + bb


######################################################


# Explicitly compute the fit parameters and their uncertainties
# with no weighting

print('\nCompute fit parameters explicitly from derived equations')
xav = 0
yav = 0
xyav = 0
x2av = 0
for i in range(0,Ndata):
    xav = xav+xdata[i]/Ndata
    yav = yav+ydata[i]/Ndata
    x2av = x2av+xdata[i]*xdata[i]/Ndata
    xyav = xyav+xdata[i]*ydata[i]/Ndata

m_fit = (xyav-xav*yav)/(x2av-xav**2)
b_fit = yav - m_fit*xav

ypredict = np.zeros(Ndata)
variance_fit = 0
```

```python
for i in range(0,Ndata):
    ypredict[i] = flinear(xdata[i],m_fit,b_fit)
    # In the following, 2 is subtracted from Ndata because 2 degrees
    # freedom (slope and intercept) were used to estimage sigma_y
    variance_fit = variance_fit + (ydata[i]-ypredict[i])**2
sigma_y_estimate = math.sqrt(variance_fit/(Ndata-2))
sigma_m_fit = sigma_y_estimate/math.sqrt(Ndata*(x2av-xav**2))
sigma_b_fit = sigma_m_fit*math.sqrt(x2av)

print('Unweighted   m_fit = {0:0.3f} \u00B1 {1:0.3f}    b_fit = {2:0
      .format(m_fit,sigma_m_fit,b_fit,sigma_b_fit))


#####################################################


# Now explicitly compute the fit parameters and their uncertainties
# using variance weighting

xav = 0
yav = 0
xyav = 0
x2av = 0
sum_one_over_sigma_y_squared = 0
for i in range(0,Ndata):
    xav = xav+xdata[i]/(sigma_ydata[i])**2
    yav = yav+ydata[i]/(sigma_ydata[i])**2
    x2av = x2av+xdata[i]*xdata[i]/(sigma_ydata[i])**2
    xyav = xyav+xdata[i]*ydata[i]/(sigma_ydata[i])**2
    sum_one_over_sigma_y_squared = sum_one_over_sigma_y_squared \
    + 1/(sigma_ydata[i])**2

xav = xav/sum_one_over_sigma_y_squared
yav = yav/sum_one_over_sigma_y_squared
x2av = x2av/sum_one_over_sigma_y_squared
xyav = xyav/sum_one_over_sigma_y_squared
sigma_y_tilde = math.sqrt(Ndata/sum_one_over_sigma_y_squared)

m_fit_w = (xyav-xav*yav)/(x2av-xav**2)
b_fit_w = yav - m_fit_w*xav

sigma_m_fit_w = sigma_y_tilde/math.sqrt(Ndata*(x2av-xav**2))
sigma_b_fit_w = sigma_m_fit_w*math.sqrt(x2av)

print('Weighted   m_fit_w = {0:0.3f} \u00B1 {1:0.3f}  b_fit_w = {2:0
      .format(m_fit_w,sigma_m_fit_w,b_fit_w,sigma_b_fit_w ))


#####################################################


# Now use scipy.stats linregress module instead.
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.l
print()
```

```
print()
m_stats, b_stats, r_value, p_value, std_err = linregress(xdata,ydata
print('Now use numpy.stats linregress module p_value = {0:0.2e} r_va
print('m_stats = {0:0.3f} \u00B1 {1:0.3f}    b_stats={2:0.3f}'.forma


#######################################################


# Finally, use scipy.optimize curve_fit module, first unweighted and
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimiz
# See also
# https://scipython.com/book/chapter-8-scipy/examples/weighted-and-n
print('\nFinally use numpy.optimize curve_fit module')
popt, pcov = curve_fit(flinear, xdata, ydata)
m_opt,b_opt = popt
sigma_m_opt,sigma_b_opt = np.sqrt(np.diag(pcov))
print('m_opt = {0:0.3f} \u00B1 {1:0.3f}    b_opt = {2:0.3f} \u00B1 {
      .format(m_opt,sigma_m_opt,b_opt,sigma_b_opt))
popt_w, pcov_w = curve_fit(flinear, xdata, ydata, p0=None, sigma=sig
m_opt_w,b_opt_w = popt_w
sigma_m_opt_w,sigma_b_opt_w = np.sqrt(np.diag(pcov_w))
print('m_opt_w = {0:0.3f} \u00B1 {1:0.3f}    b_opt_w = {2:0.3f} \u00
      .format(m_opt_w,sigma_m_opt_w,b_opt_w,sigma_b_opt_w))


#######################################################


print()
print('sigma_y using fit = {0:0.3f}'.format(sigma_y_estimate))
print('sigma_y_tilde = {0:.3f}'.format(sigma_y_tilde))


#######################################################


# This is needed for plotting the line fitted using variance weighti
ypredict_w = np.zeros(Ndata)
for i in range(0,Ndata):
    ypredict_w[i] = flinear(xdata[i],m_opt_w,b_opt_w)
    # ypredict_w[i] = m_opt_w*xdata[i]+b_opt_w


#######################################################


# Now plot the data and best fit lines for unweighted and weighted c
datapoints, = plt.plot(xdata,ydata,'r.',markersize=12,label='data')
plt.errorbar(xdata,ydata,yerr=sigma_ydata,marker='.',linestyle = 'no
fittedline, = plt.plot(xdata,ypredict,label='fitted line')
fittedlinew, = plt.plot(xdata,ypredict_w,label='fitted line, weighte
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear fit to data')
```

```
plt.title('Linear fit to data')
plt.legend(handles=[datapoints,fittedline,fittedlinew])
plt.show()
```

```
Randall Tagg 2021-12-04 16:30:38

x  [0. 1. 2. 3. 4. 5. 6.]
y [ 4.111  7.632  9.563 14.53  16.269 19.003 20.752]
sigma_ydata [0.5 0.7 0.3 1.  0.4 0.2 0.6]

Compute fit parameters explicitly from derived equations
Unweighted   m_fit = 2.835 ± 0.163    b_fit = 4.619 ± 0.586
Weighted   m_fit_w = 2.967 ± 0.079  b_fit_w = 4.059 ± 0.325

Now use numpy.stats linregress module p_value = 1.14e-05 r_value=0.992
m_stats = 2.835 ± 0.163    b_stats=4.619

Finally use numpy.optimize curve_fit module
m_opt = 2.835 ± 0.163    b_opt = 4.619 ± 0.586
m_opt_w = 2.967 ± 0.079    b_opt_w = 4.059 ± 0.325

sigma_y using fit = 0.861
sigma_y_tilde = 0.366
```



Code to read in data from a file and fit to a straight line

The following will do a weighted fit using the curve_fit module from the scipy.optimize library for data read from a comma delimoted (.csv) file. There should be two or three columns:

- xdata
- ydata
- uncertainty estimates for ydata If the third column is missing then uniform weighting will be used and an estimate of the y variance will be made using the residuals (i.e. the differences between best fit and data).

The first line of the file should contain the variable names. The second line of the file should contain the units.

Here is an example:

```
time,position,sigma
s,m,m
0,15,1.2
1,17.1,0.5
2,18.9,1.1
3,21.3,1.2
4,22.7,1.1
```

This renders in table form as

time	position	sigma
s	m	m
0	15	1.2
1	17.1	0.5
2	18.9	1.1
3	21.3	1.2
4	22.7	1.1

Edit the variables in the code below before running:

```
myname = (your name for the output time stamp)
myDataFile = (name of the data file, which should have extension
.csv)
plotTitle = (a text string)
```

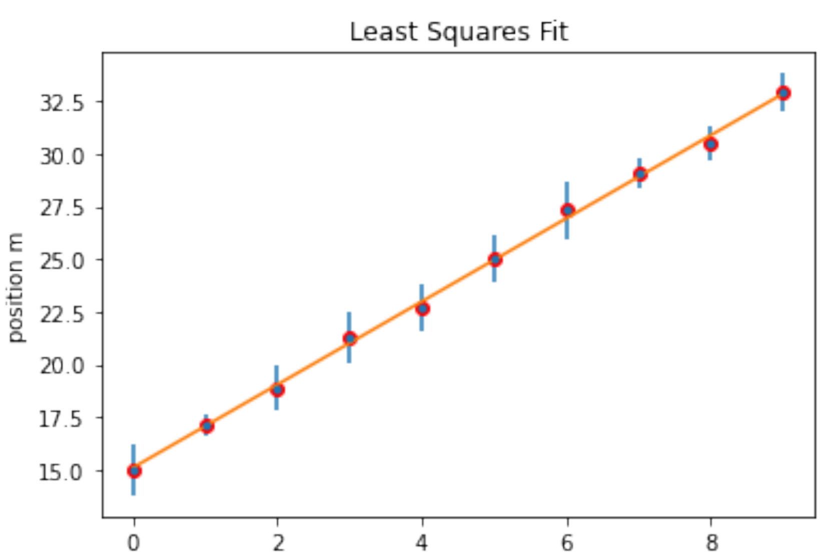
It is likely also to be necessary to edit the data formatting in the print statements. (The example uses the format 0.3f for all values.)

In [83]:

```
%matplotlib inline
'''Code to read in data including uncertainties and do linear least s

import numpy as np
import matplotlib.pyplot as plt
import math,datetime
from scipy.optimize import curve_fit

def flinear(xx,mm,bb):  # define the linear function (for use with th
    return mm*xx + bb

# *****************************USER EDITED PORTION *****************

#Put in the name of the desired data file in the following line
#and edit the x and y labels to suit the paritcular data
my_name = "J.J. Datasmith"
myDataFile = 'testData.csv'
plotTitle = 'Least Squares Fit'

# ******************************************************************

#Read in the header
f = open(myDataFile)
varnames = f.readline()
units = f.readline()

#Read in the data
mydata = np.loadtxt(myDataFile,delimiter=',',skiprows=2)

if (mydata.shape[1] == 3):
    difvar = True
    xname,yname,signame = varnames.split(',')
    xunit,yunit,sigunit =  units.split(',')
else:
    difvar = False
    xname,yname = varnames.split(',')
    yname = yname.rstrip(yname[-1]) # remove the line feed
    xunit,yunit =  units.split(',')
    yunit = yunit.rstrip(yunit[-1]) # remove the line feed

#print(difvar)
#print(mydata.shape[1])

x = mydata[:,0]   # x data
y = mydata[:,1]   # y data
if (difvar):
    sigma_y = mydata[:,2] # uncertainties in y data
else:
    sigma_y = np.ones(Ndata) # equal weighting

# Use numpy.optimize curve_fit module to make a least squares fit of
# data to a straight line, using variance weighting (hence the _w des
popt_w, pcov_w = curve_fit(flinear, x, y, p0=None, sigma=sigma_y, abs
m,b = popt_w
```

```
sigma_m,sigma_b = np.sqrt(np.diag(pcov_w))

#Compute y values predicted from best fit line
Ndata = y.size
ypredicted = np.zeros(Ndata)
variance_fit = 0
tempsig = 0
for i in range(0,Ndata):
    ypredicted[i] = flinear(x[i],m,b)
    variance_fit = variance_fit + (y[i]-ypredicted[i])**2
    if(difvar): tempsig = tempsig + 1/sigma_y[i]**2
# In the following, 2 is subtracted from Ndata because 2 degrees of
# freedom (slope and intercept) were used to estimage sigma_y
sigma_y_estimate = math.sqrt(variance_fit/(Ndata-2))
if(difvar): sigma_y_tilde = math.sqrt(Ndata/tempsig)

# Create a time stamp for each run
time_stamp_format = "%Y-%m-%d %H:%M:%S"
print(my_name,datetime.datetime.now().strftime(time_stamp_format),'\n

plt.plot(x,y,'r.',markersize=12,label='data')
if(difvar): plt.errorbar(x,y,yerr=sigma_y,marker='.',linestyle = 'non
plt.plot(x,ypredicted,label='best fit line')
xlabel = xname + ' ' + xunit
plt.xlabel(xlabel)
ylabel = yname + ' ' + yunit
plt.ylabel(ylabel)
plt.title(plotTitle)

plt.show()

print('Slope m = {0:0.3f} \u00B1 {1:0.3f}'.format(m,sigma_m),yunit,'/
print('Intercept b = {0:0.3f} \u00B1 {1:0.3f}'.format(b,sigma_b),yuni
print('Fit estimated y std dev = {0:0.3f}'.format(sigma_y_estimate),y
if(difvar): print('Inverse avg y std dev = {0:0.3f}'.format(sigma_y_t
```

```
J.J. Datasmith 2023-03-03 19:22:04
```



time s

```
Slope m = 1.969 ± 0.089 m / s
Intercept b = 15.113 ± 0.466 m
Fit estimated y std dev = 0.255 m
Inverse avg y std dev = 0.870 m
```

Code to explore the effects size of randomness and other details

The followiing creates a synthetic data set with adjustable uncertainties in order to explore how these affect the fit.

In [32]:

```
'''
exploreLSv2.py
Code to explore least squares fitting: an artificial data set
is generated using a random number generator that uses a Gaussian
distribution of errors with standard deviation specified by the
parameter sigma. The effect of changing the size of sigma on the
quality of fit can be explored. Both unweighted and weighted fitting
is done.
Randall Tagg 07-Oct-2020
'''

import math
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import linregress
from scipy.optimize import curve_fit
import datetime

# Create a time stamp for each run
my_name = "Randall Tagg"
time_stamp_format = "%Y-%m-%d %H:%M:%S"
print(my_name,datetime.datetime.now().strftime(time_stamp_format),'\r

def flinear(xx,mm,bb):  # define the linear function (for use with th
    return mm*xx + bb

#Actual slope and intercept for exact line
m=3.
b=4.


#Create synthetic data for exploring how fits depend on the distribut

xdata = np.asarray([0.,1.,2.,3.,4.,5.,6.])
Ndata = xdata.size

# Uncertainties to use in generating data using a normal distributior
sigma_scale_factor = 2
```

```
sigma_scale_factor = 2.
sigma_ydata = sigma_scale_factor*np.asarray([0.5,0.7,0.3,1.0,0.4,0.2,

#sigma_ydata = 2*np.asarray([0.5,0.5,0.5,0.5,0.5,0.5,0.5]) # use this

yexact = np.zeros(Ndata)
ydata = np.zeros(Ndata)
ydiff = np.zeros(Ndata)
ydiff_sum = 0
#Synthesize the y data
for i in range(0,Ndata):
    yexact[i] = flinear(xdata[i],m,b)
    ydata[i] = np.random.normal(yexact[i], sigma_ydata[i], 1) # creat
    ydiff[i] = ydata[i]-yexact[i]
    ydiff_sum = ydiff_sum + ydiff[i]
ydiff_mean = ydiff_sum/Ndata
#Comment out the following line if using emperical mean for ydiff; ot
ydiff_mean = 0 # override previous result to force assumed zero mean

yvariance_sum = 0
for i in range(0,Ndata):
    yvariance_sum = yvariance_sum + (ydiff[i]-ydiff_mean)**2
sigma_y_sample = math.sqrt(yvariance_sum/(Ndata-1))

print('x ',xdata)
print('Exact y ',yexact)
print('Random y [{0:.3f} {1:.3f} {2:.3f} {3:.3f} {4:.3f} {5:.3f} {6:.
      .format(ydata[0],ydata[1],ydata[2],ydata[3],ydata[4],
              ydata[5],ydata[6]))
print('sigma_ydata [{0:.3f} {1:.3f} {2:.3f} {3:.3f} {4:.3f} {5:.3f} {
      .format(sigma_ydata[0],sigma_ydata[1],sigma_ydata[2],sigma_ydat
              sigma_ydata[5],sigma_ydata[6]))

print('\nm_exact = {0:.3f}   b_exact = {1:.3f}'.format(m, b))


# Explicitly compute the fit parameters and their uncertainties with

print('\nCompute fit parameters explicitly from derived equations')
xav = 0
yav = 0
xyav = 0
x2av = 0
for i in range(0,Ndata):
    xav = xav+xdata[i]/Ndata
    yav = yav+ydata[i]/Ndata
    x2av = x2av+xdata[i]*xdata[i]/Ndata
    xyav = xyav+xdata[i]*ydata[i]/Ndata

m_fit = (xyav-xav*yav)/(x2av-xav**2)
b_fit = yav - m_fit*xav

ypredict = np.zeros(Ndata)
variance_fit = 0
for i in range(0,Ndata):
```

```
for i in range(0,Ndata):
    ypredict[i] = flinear(xdata[i],m_fit,b_fit)
    # In the following, 2 is subtracted from Ndata because 2 degrees
    # freedom (slope and intercept) were used to estimage sigma_y
    variance_fit = variance_fit + (ydata[i]-ypredict[i])**2
sigma_y_estimate = math.sqrt(variance_fit/(Ndata-2))
sigma_m_fit = sigma_y_estimate/math.sqrt(Ndata*(x2av-xav**2))
sigma_b_fit = sigma_m_fit*math.sqrt(x2av)

print('m_fit = {0:0.3f} \u00B1 {1:0.3f}    b_fit = {2:0.3f} \u00B1 {3
      .format(m_fit,sigma_m_fit,b_fit,sigma_b_fit))


# Now explicitly computer the fit parameters and their uncertainties

xav = 0
yav = 0
xyav = 0
x2av = 0
sum_one_over_sigma_y_squared = 0
for i in range(0,Ndata):
    xav = xav+xdata[i]/(sigma_ydata[i])**2
    yav = yav+ydata[i]/(sigma_ydata[i])**2
    x2av = x2av+xdata[i]*xdata[i]/(sigma_ydata[i])**2
    xyav = xyav+xdata[i]*ydata[i]/(sigma_ydata[i])**2
    sum_one_over_sigma_y_squared = sum_one_over_sigma_y_squared + 1/(

xav = xav/sum_one_over_sigma_y_squared
yav = yav/sum_one_over_sigma_y_squared
x2av = x2av/sum_one_over_sigma_y_squared
xyav = xyav/sum_one_over_sigma_y_squared
sigma_y_tilde = math.sqrt(Ndata/sum_one_over_sigma_y_squared)

m_fit_w = (xyav-xav*yav)/(x2av-xav**2)
b_fit_w = yav - m_fit_w*xav

sigma_m_fit_w = sigma_y_tilde/math.sqrt(Ndata*(x2av-xav**2))
sigma_b_fit_w = sigma_m_fit_w*math.sqrt(x2av)

print('m_fit_w = {0:0.3f} \u00B1 {1:0.3f}    b_fit_w = {2:0.3f} \u00B
      .format(m_fit_w,sigma_m_fit_w,b_fit_w,sigma_b_fit_w ))



# Now use scipy.stats linregress module instead.
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.li
print()
m_stats, b_stats, r_value, p_value, std_err = linregress(xdata,ydata)
print('Now use numpy.stats linregress module p_value = {0:0.2e}  r_va
print('m_stats={0:0.3f} \u00B1 {1:0.3f}    b_stats={2:0.3f}'.format(m



# Finally, use scipy.optimize curve_fit module, first unweighted and
```

```python
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize
# See also
# https://scipython.com/book/chapter-8-scipy/examples/weighted-and-nc
print('\nFinally use numpy.optimize curve_fit module')
popt, pcov = curve_fit(flinear, xdata, ydata)
m_opt,b_opt = popt
sigma_m_opt,sigma_b_opt = np.sqrt(np.diag(pcov))
print('m_opt = {0:0.3f} \u00B1 {1:0.3f}    b_opt = {2:0.3f} \u00B1 {3
      .format(m_opt,sigma_m_opt,b_opt,sigma_b_opt))
popt_w, pcov_w = curve_fit(flinear, xdata, ydata, p0=None, sigma=sigm
m_opt_w,b_opt_w = popt_w
sigma_m_opt_w,sigma_b_opt_w = np.sqrt(np.diag(pcov_w))
print('m_opt_w = {0:0.3f} \u00B1 {1:0.3f}    b_opt_w = {2:0.3f} \u00E
      .format(m_opt_w,sigma_m_opt_w,b_opt_w,sigma_b_opt_w))



print()
print('sigma_y using diff from exact y values (zero mean diff) = {0:.
#print('sigma_y using diff from exact y values (empirical mean diff)
print('sigma_y using fit = {0:0.3f}'.format(sigma_y_estimate))
print('sigma_y_tilde = {0:.3f}'.format(sigma_y_tilde))



# This is needed for plotting the line fitted using variance weightir
ypredict_w = np.zeros(Ndata)
for i in range(0,Ndata):
    ypredict_w[i] = flinear(xdata[i],m_opt_w,b_opt_w)
    # ypredict_w[i] = m_opt_w*xdata[i]+b_opt_w

datapoints, = plt.plot(xdata,ydata,'r.',markersize=12,label='data')
plt.errorbar(xdata,ydata,yerr=sigma_ydata,marker='.',linestyle = 'nor
exactline, = plt.plot(xdata,yexact,'--',label='exact line')
fittedline, = plt.plot(xdata,ypredict,label='fitted line')
fittedlinew, = plt.plot(xdata,ypredict_w,label='fitted line, weightec
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear fit to data')
plt.legend(handles=[datapoints,exactline,fittedline,fittedlinew])
plt.show()

```

```
Randall Tagg 2021-12-04 17:32:21

x  [0. 1. 2. 3. 4. 5. 6.]
Exact y  [ 4.  7. 10. 13. 16. 19. 22.]
Random y [4.252 9.265 10.193 16.549 16.003 19.867 21.475]
sigma_ydata [1.000 1.400 0.600 2.000 0.800 0.400 1.200]

m_exact = 3.000   b_exact = 4.000

Compute fit parameters explicitly from derived equations
m_fit = 2.810 ± 0.288    b_fit = 5.513 ± 1.038
```

```
m_fit_w = 3.012 ± 0.158    b_fit_w = 4.559 ± 0.650

Now use numpy.stats linregress module p_value = 1.92e-04  r_value=0.97
5
m_stats=2.810 ± 0.288    b_stats=5.513

Finally use numpy.optimize curve_fit module
m_opt = 2.810 ± 0.288    b_opt = 5.513 ± 1.038
m_opt_w = 3.012 ± 0.158    b_opt_w = 4.559 ± 0.650

sigma_y using diff from exact y values (zero mean diff) = 1.773
sigma_y using fit = 1.524
sigma_y_tilde = 0.733
```