NORTHERN ILLINOIS UNIVERSITY

PHYSICS DEPARTMENT

Physics 374 – Junior Physics Lab Spring 2024

Python Tutorial #6

**Python Nonlinear Least Squares Fitting**

In this tutorial, we will learn how to how to use the least squares fitting routines in the *scipy* module to do nonlinear fits to data. To use *scipy*, we must install its module using *pip* (see Python Tutorial #1)—open up Windows PowerShell and type:

py -m pip install scipy

A good tutorial for *scipy* can be found at:

<https://python4mpia.github.io/fitting_data/least-squares-fitting.html> and <https://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.optimize.curve_fit.html>.

We will use the curve\_fit function in the *scipy* module to fit functions. It applies the Levenburg-Marquardt gradient method using what is called a *greedy algorithm* to minimize chi-squared.

To introduce us to *scipy*, let’s fit the data in Bevington, Table 7.2, with a straight line. To use the curve\_fit function in the *scipy* module, we must import *scipy* into to the program:

import scipy.optimize as opt

In the code below, the six voltage and temperature data points are contained in the xdata and ydata arrays. Since this is a numerical least squares fitting method, we have to provide initial guesses of the parameter values. The guesses of the parameters are contained in the array called guess. If the initial guesses are far from the optimal parameter values that miminize , then the numerical calculations may never converge to the correct solutions. Thus, it is important to make good initial guesses for the parameters. The array sigma contains the uncertainties of the measured values (in this case, the voltages). Using the *numpy* routine fill, we can fill the array sigma with equal uncertainties of 0.05 volts for all measurements. The function calfun(x, a1, a2) contains the curve we wish to fit the data to, which, in this case, is the equation of a straight line. The least squares fitting routine, curve\_fit, applies the Levenburg-Marquardt gradient technique to fit the data. The output from curve\_fit is the final parameter file par and the error (or covariance) matrix error. Open up Microsoft Visual Studio and copy and paste the following code into a project called *Least\_Squares*:

import numpy as np # the alias for "numpy" will be "np"

import matplotlib.pyplot as plt # the alias for "matplotlib.pyplot" will be "plt"

import scipy.optimize as opt # the alias for "scipy.optimize" will be "opt"

xdata = np.array([0.0, 20.0, 40.0, 60.0, 80.0, 100.0]) # Temperature data

ydata = np.array([-0.849, -0.196, 0.734, 1.541, 2.456, 3.318]) # Voltage data

guess = np.array([0.01, -0.1]) # initial guesses for parameters a1 & a2

 # for the linear fit y = a1 + a2\*x

sigma = np.empty([6]) # create an empty array of six elements

sigma.fill(0.05) # uncertainties all voltages measurements are 0.05 volts

 # "fill" sets all elements of the array to be the same number

def calfun(x, a1, a2): # defining the function y = a1 + a2\*x

 return a1 + a2\*x

#

# par --> contains the final parameter values of the fit

# error --> contains the error, or covariance, matrix

# absolute\_sigma=False --> Sigma contains relative weights of the data points.

# The covariance matrix will be based on estimated

# errors in the data

# absolute\_sigma=True --> Sigma contains standard deviation errors of the data

# points. The covariance matrix will be based on these

# values.

#

par,error = opt.curve\_fit(calfun, xdata, ydata, guess, sigma, absolute\_sigma=True)

uncertainties = np.sqrt(np.diagonal(error)) # take square root of diagonal matrix

#

# Printing outputs

#

print(par) # print the parameter array # elements

print()

print(error) # print the error (covariance) matrix

print()

print(uncertainties) # print the uncertainties of each parameter

print() # "\u00B1" is the +/- symbol ; ":.8f" = write result to 8 decimal places

print("parameter a1 = {:.8f} \u00B1 {:.8f}".format(par[0],uncertainties[0]))

print("parameter a2 = {:.8f} \u00B1 {:.8f}".format(par[1],uncertainties[1]))

#

# Plotting outputs

#

plt.plot(xdata,ydata,"ko",label="data",ms=2) # make a scatter plot for the data

a1, a2 = par # set a1 and a2 parameters # "ms" sets the size of the circle

x = np.linspace(min(xdata), max(xdata), 100) # array of 100 x-axis points

yfit = calfun(x,a1,a2) # array of yfit values for each value of x array

plt.plot(x,yfit,":",label="fit") # plot the fitting curve with a dotted curve “:”

plt.errorbar(xdata,ydata,yerr=sigma,capsize=3,fmt='ko',ms=1) # plot error bars

plt.legend() # show the legend

plt.title("Least Squares Linear Fit") # plot title of the plot

# "\N{DEGREE SIGN}" is the degee symbol for degrees Celcius

plt.xlabel("Temperature (\N{DEGREE SIGN}C)") # plot the x-axis label

plt.ylabel("Voltage (V)") # plot the y-axis label

plt.show() # command to draw the plot on the screen

The output is shown below:



Notice that the parameter array and the error matrix agree with Bevington’s results in Table 7.2 (page 125).

**Homework**

none