```
#This example shows you how to fit a plot with various user defined
#function, in different ranges and with different colors

#you can comment out various regions of this file to test different
# parts
import numpy as np
import LT.box as B
#need the next two modules to fit a general user defined function
import LT_Fit.parameters as P  # get the parameter module
import LT_Fit.gen_fit as G     # load the genfit module

#need scipy.misc to calculate factorial function
import scipy.misc as ms
file_name = 'examples.data'
f = B.get_file(file_name)
# get the data
A = B.get_data(f, 'A')
b = B.get_data(f, 'b')
db = B.get_data(f, 'db')
C = B.get_data(f, 'C')
D = B.get_data(f, 'D')

#plot the data
B.plot_exp(C, D, db)
B.pl.show()
#initialize the parameters
c1 = P.Parameter(1., 'pol0')     # parameter c1, called 'pol0',
                                 #initialized to 1.
#Of course, you could have called the parameter c1 as 'c1" as
#well. What's inside the '' is what will be printed out

c2 = P.Parameter(1., 'pol1')    # parameter c2, called 'pol1',
                                # intialized to 1.
c3 = P.Parameter(1., 'pol2')  # parameter c3, called 'pol2',
                              #initialized to 1.
height = P.Parameter(8., 'height')
mean = P.Parameter(6., 'mean')
sigma = P.Parameter(1., 'sigma')

#define the functions that takes the parameters (to be fitted), and
#the variable (x) the first function defined below is a 2nd order
#polynomial + Gaussian
def myfun(x):
```

```
        value = c1()+c2()*x+c3()*x**2+height()*np.exp(-((x-mean())/
        (2*sigma()))**2)
        return value

#fit the D vs C plot using the function you just defined, called
#myfun Notice here c1, c2, c3, height, mean, sigma are of the same
#names as defined earlier.
fit5 = G.genfit(myfun, [c1, c2, c3, height, mean, sigma], x = C,
y = D, y_err = db)

B.plot_line(fit5.xpl, fit5.ypl, color='blue')

#We can also fit the same plot, same myfun, but in a sub-range
#defined by r3 below

r3 = B.in_window(2.0, 12.0, C)

fit6 = G.genfit(myfun, [c1, c2, c3, height, mean, sigma], x = C[r3],
y = D[r3], y_err = db[r3])

B.plot_line(fit6.xpl, fit6.ypl, color='magenta')

#We can define another function myfun1, to fit the same plot again

#mu = P.parameter(6., 'mu')
mu = P.Parameter(6., 'mu')
norm = P.Parameter(10., 'norm')

#second function defined here is a pure poisson function
def myfun1(x):
        value = norm()*mu()**(x)*np.exp(-mu())/ms.factorial(x)
        return value

fit7 = G.genfit(myfun1, [mu, norm], x = C[r3], y = D[r3],
y_err = db[r3])
B.plot_line(fit7.xpl, fit7.ypl, color='red')


c1 = P.Parameter(1., 'pol0')      # parameter c1, called 'pol0',
                                  #initialized to 1.
c2 = P.Parameter(1., 'pol1')    # parameter c2, called 'pol1',
                                 # intialized to 1.
c3 = P.Parameter(1., 'pol2')   # parameter c3, called 'pol2',
```

```
                                #initialized to 1.
mu = P.Parameter(7., 'mu')
norm = P.Parameter(20., 'norm')

#Third function defined here is a 2nd order polynomial + poisson
#function
def myfun2(x):
        pol2 = c1() + c2()*x + c3()*x**2
        value = pol2 +norm()*mu()**(x)*np.exp(-mu())/ms.factorial(x)
        return value

r4 = B.in_window(4.0, 16.0, C)

fit8 = G.genfit(myfun2, [c1, c2, c3, mu, norm], x = C[r4],
y = D[r4], y_err = db[r4])
B.plot_line(fit8.xpl, fit8.ypl, color='brown')

#You can also set the x^2 term to be 0, and fit it again
c1 = P.Parameter(1., 'pol0')     # parameter c1, called 'pol0',
                                 #initialized to 1.
c2 = P.Parameter(1., 'pol1')    # parameter c2, called 'pol1',
                                 #intialized to 1.
c3 = P.Parameter(1., 'pol2')   # parameter c3, called 'pol2',
                                 #initialized to 1.
mu = P.Parameter(7., 'mu')
norm = P.Parameter(20., 'norm')
c2.set(0.)

#Noteice c2 is set, and is no longer in the list of paramters below
#that are to be fitted
fit9 = G.genfit(myfun2, [c1, c3, mu, norm], x = C[r4], y = D[r4],
y_err = db[r4])
B.plot_line(fit9.xpl, fit9.ypl, color='green')

# You can get the value and uncertainty of the fitted parameter,
#and do furthe analysis,
#for example
muvalue = mu.value
muerr=mu.err
print "mu has the value of %.3f +- %.3f\n" %(muvalue, muerr)
```