```
In [232]: #This example shows you how to fit a plot with various user
defined

In [233]: #function, in different ranges and with different colors

In [234]: #you can comment out various regions of this file to test
different

In [235]: # parts

In [236]: import numpy as np

In [237]: import LT.box as B

In [238]: #need the next two modules to fit a general user defined
function

In [239]: import LT_Fit.parameters as P      # get the parameter module

In [240]: import LT_Fit.gen_fit as G         # load the genfit module

In [240]:

In [241]: #need scipy.misc to calculate factorial function

In [242]: #import scipy.misc as ms

In [243]: import scipy.special as sp

In [243]:

In [244]: file_name = 'examples_data.py' #save data name

In [245]: f = B.get_file(file_name) #open data

In [246]: # get the data, extract the columns and save them in their
corresponding variables

In [247]: A = B.get_data(f, 'A')

In [248]: b = B.get_data(f, 'b')

In [249]: db = B.get_data(f, 'db')

In [250]: C = B.get_data(f, 'C')

In [251]: D = B.get_data(f, 'D')

In [251]:
```

```
In [252]: #plot the data

In [253]: B.plot_exp(C, D, db)
Out[253]: <ErrorbarContainer object of 3 artists>

In [254]: B.pl.show()

In [254]:

In [255]: #initialize the parameters

In [256]: c1 = P.Parameter(1., 'pol0')      # parameter c1, called
'pol0',

In [257]: #initialized to 1.

In [258]: #Of course, you could have called the parameter c1 as 'c1' as

In [259]: #well. What's inside the '' is what will be printed out

In [259]:

In [260]: c2 = P.Parameter(1., 'pol1')      # parameter c2, called
'pol1',

In [261]: # initialized to 1.

In [262]: c3 = P.Parameter(1., 'pol2')      # parameter c3, called
'pol2',

In [263]: #initialized to 1.

In [264]: height = P.Parameter(3., 'height')

In [265]: mean = P.Parameter(5., 'mean')

In [266]: sigma = P.Parameter(1., 'sigma')

In [266]:

In [267]: #define the functions that takes the parameters (to be
fitted), and

In [268]: #the variable (x) the first function defined below is a 2nd
order

In [269]: #polynomial + Gaussian

In [270]: def myfun(x):
     ...:     value = c1() + c2()*x + c3()*x**2 + height()*np.exp(-((x-
mean())**2)/(2*sigma()**2))
```

```
     ...:     return value  # Check if this Gaussian function is defined
correctly. IF not, correct it!
     ...:

In [271]: #fit the D vs C plot using the function you just defined,
called

In [272]: #myfun Notice here c1, c2, c3, height, mean, sigma are of the
same

In [273]: #names as defined earlier.

In [274]: fit5 = G.genfit(myfun, [c1, c2, c3, height, mean, sigma], x =
C, y = D, y_err = db)
gen_fit kwargs =  {}
gen_fit.fit kwargs =  {}
Calculate numerical parameter derivatives with diff_step = 0.001
------------------------------------------------------------------------
fit results :
------------------------------------------------------------------------
chisquare =  16.864325479182067
red. chisquare =  1.873813942131341
parameters:
parameter  0  :  pol0 = np.float64(1.2882029600245064) +/-
np.float64(0.24350258648094902)
parameter  1  :  pol1 = np.float64(0.009042570264055031) +/-
np.float64(0.08082205029533764)
parameter  2  :  pol2 = np.float64(-0.006114591681902849) +/-
np.float64(0.005008505335969559)
parameter  3  :  height = np.float64(5.487291915274562) +/-
np.float64(0.25687330662690866)
parameter  4  :  mean = np.float64(6.06398917411691) +/-
np.float64(0.06649191264202162)
parameter  5  :  sigma = np.float64(1.53248048614974) +/-
np.float64(0.0857722548726211)

In [274]:

In [275]: B.plot_line(fit5.xpl, fit5.ypl, color='blue')
Out[275]: [<matplotlib.lines.Line2D at 0x168b503d0>]

In [275]:

In [276]: #We can also fit the same plot, same myfun, but in a sub-range

In [277]: #defined by r3 below

In [278]: r3 = B.in_between(2.0, 12.0, C)

In [278]:
```

```
In [279]: fit6 = G.genfit(myfun, [c1, c2, c3, height, mean, sigma], x =
C[r3], y = D[r3], y_err = db[r3])
gen_fit kwargs =  {}
gen_fit.fit kwargs =  {}
Calculate numerical parameter derivatives with diff_step = 0.001
------------------------------------------------------------------------
fit results :
------------------------------------------------------------------------
chisquare =  10.024716792797241
red. chisquare =  2.5061791981993102
parameters:
parameter  0  :  pol0 = np.float64(1.4384421763508077) +/-
np.float64(0.6180009449581974)
parameter  1  :  pol1 = np.float64(0.20001894691962246) +/-
np.float64(0.3240329245056917)
parameter  2  :  pol2 = np.float64(-0.023119511623679055) +/-
np.float64(0.02405310310360854)
parameter  3  :  height = np.float64(4.942420920428553) +/-
np.float64(0.5184108174284167)
parameter  4  :  mean = np.float64(6.099540998155622) +/-
np.float64(0.07369169551322495)
parameter  5  :  sigma = np.float64(1.3300423295647972) +/-
np.float64(0.13950595058811247)

In [279]:

In [280]: B.plot_line(fit6.xpl, fit6.ypl, color='magenta')
Out[280]: [<matplotlib.lines.Line2D at 0x168329d90>]

In [280]:

In [281]: #We can define another function myfun1, to fit the same plot
again

In [281]:

In [282]: mu = P.Parameter(6., 'mu')

In [283]: norm = P.Parameter(10., 'norm')

In [283]:

In [284]: #second function defined here is a pure poisson function

In [285]: def myfun1(x):
     ...:     value = norm()*mu()**(x)*np.exp(-mu())/sp.factorial(x)
     ...:     return value

In [286]: fit7 = G.genfit(myfun1, [mu, norm], x = C[r3], y = D[r3],
y_err = db[r3])
gen_fit kwargs =  {}
```

```
gen_fit.fit kwargs =  {}
Calculate numerical parameter derivatives with diff_step = 0.001
------------------------------------------------------------------------
fit results :
------------------------------------------------------------------------
chisquare =  75.57973968005273
red. chisquare =  9.44746746000659
parameters:
parameter  0  :  mu = np.float64(6.2440521381677385) +/-
np.float64(0.08014407845440903)
parameter  1  :  norm = np.float64(34.92103671624136) +/-
np.float64(0.763678410590509)

In [287]: B.plot_line(fit7.xpl, fit7.ypl, color='red')
Out[287]: [<matplotlib.lines.Line2D at 0x168ef6e90>]

In [287]:

In [288]: c1 = P.Parameter(1., 'pol0')      # parameter c1, called
'pol0',

In [289]: #initialized to 1.

In [290]: c2 = P.Parameter(1., 'pol1')      # parameter c2, called
'pol1',

In [291]: # initialized to 1.

In [292]: c3 = P.Parameter(1., 'pol2')      # parameter c3, called
'pol2',

In [293]: #initialized to 1.

In [294]: mu = P.Parameter(7., 'mu')

In [295]: norm = P.Parameter(20., 'norm')

In [295]:

In [296]: #Third function defined here is a 2nd order polynomial +
poisson

In [297]: #function

In [298]: def myfun2(x):
     ...:     pol2 = c1() + c2()*x + c3()*x**2
     ...:     value = pol2 + norm()*mu()**(x)*np.exp(-mu())/
sp.factorial(x)
     ...:     return value

In [299]: r4 = B.in_between(4.0, 16.0, C)
```

```
In [299]:

In [300]: fit8 = G.genfit(myfun2, [c1, c2, c3, mu, norm], x = C[r4],
     ...:                 y = D[r4], y_err = db[r4])
gen_fit kwargs =  {}
gen_fit.fit kwargs =  {}
Calculate numerical parameter derivatives with diff_step = 0.001
------------------------------------------------------------------------
fit results :
------------------------------------------------------------------------
chisquare =  31.651775872287086
red. chisquare =  4.521682267469584
parameters:
parameter  0  :  pol0 = np.float64(-13.370845121532135) +/-
np.float64(2.658343686173159)
parameter  1  :  pol1 = np.float64(1.8888677716125142) +/-
np.float64(0.4255121242224953)
parameter  2  :  pol2 = np.float64(-0.06603939538984842) +/-
np.float64(0.01688559716149658)
parameter  3  :  mu = np.float64(5.647756479347775) +/-
np.float64(0.11317569991836551)
parameter  4  :  norm = np.float64(67.55809200192168) +/-
np.float64(5.8600960024723445)

In [301]: B.plot_line(fit8.xpl, fit8.ypl, color='brown')
Out[301]: [<matplotlib.lines.Line2D at 0x168f9ee90>]

In [301]:

In [302]: #You can also set the x^2 term to be 0, and fit it again

In [303]: c1 = P.Parameter(1., 'pol0')      # parameter c1, called
'pol0',

In [304]: #initialized to 1.

In [305]: c2 = P.Parameter(1., 'pol1')      # parameter c2, called
'pol1',

In [306]: #initialized to 1.

In [307]: c3 = P.Parameter(1., 'pol2')      # parameter c3, called
'pol2',

In [308]: #initialized to 1.

In [309]: mu = P.Parameter(7., 'mu')

In [310]: norm = P.Parameter(20., 'norm')
```

```
In [311]: c2.set(0.)

In [311]:

In [312]: #Notice c2 is set, and is no longer in the list of parameters
below

In [313]: #that are to be fitted

In [314]: fit9 = G.genfit(myfun2, [c1, c3, mu, norm], x = C[r4], y =
D[r4], y_err = db[r4])
gen_fit kwargs =  {}
gen_fit.fit kwargs =  {}
Calculate numerical parameter derivatives with diff_step = 0.001
------------------------------------------------------------------------
fit results :
------------------------------------------------------------------------
chisquare =  55.61665782655677
red. chisquare =  6.952082228319596
parameters:
parameter  0  :  pol0 = np.float64(-2.0453663146879437) +/-
np.float64(0.4785443852022825)
parameter  1  :  pol2 = np.float64(0.009534124791388566) +/-
np.float64(0.0023768983223789297)
parameter  2  :  mu = np.float64(6.285123228790032) +/-
np.float64(0.07242220286914432)
parameter  3  :  norm = np.float64(47.6046635513798) +/-
np.float64(3.026112921125579)

In [315]: B.plot_line(fit9.xpl, fit9.ypl, color='green')
Out[315]: [<matplotlib.lines.Line2D at 0x168cf1410>]

In [315]:

In [316]: # You can get the value and uncertainty of the fitted
parameter,

In [317]: #and do further analysis,

In [318]: #for example

In [319]: muvalue = mu.value

In [320]: muerr = mu.err

In [321]: print ("mu has the value of %f +- %f\n" %(muvalue, muerr))
mu has the value of 6.285123 +- 0.072422


In [322]:
```