```
# practiging fitting plots with various functionsimport numpy as np
import numpy as np
import LT.box as B

file_name = 'examples_data.py' #set name of data file
f = B.get_file(file_name) #open and read the file
# get the data
# current

#extract the data from the file and save them into their respective variables
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')
# The following examples fitl to fit fits C vs A, using linear fit,
#polynomial fits, in either the whole ranges (fit1, fit2) or a subrange (fit 3, 4
B.plot_exp(A, C, db) #plot the data with error bars (db)
B.pl.show() #display the plot

#You should uncomment the next line to see what it does to the x-axi
#Can you set the y-axis title now?
B.pl.xlabel("x (unit)") # sets the x-axis label
B.pl.ylabel("y (unit)")

fit1 = B.linefit(A, C, db) #does the linear fit of the entire dataset
B.plot_line(fit1.xpl, fit1.ypl) #plots the fitted line

#the following two lines selecting the ranges
r1 = B.in_between(4.0, 18.0, C) #. in window should be changed to in _between
r2 = B.in_between(2.0, 12.0, C) #creates a variable with the data points of C
#between 1 and 12

#only fit the selected ranges as specified by r1 or r2
fit3 = B.linefit(A[r1], C[r1], db[r1]) #linear fit using data from range 1
B.plot_line(fit3.xpl, fit3.ypl) #plot it

#the fit below use second order-- defined by the "2" in the argument
#polynomial; you should change 2 to other integers and see how the
#fits are different
fit4 = B.polyfit(A[r2], C[r2], db[r2], 2)
B.plot_line(fit4.xpl, fit4.ypl)

#for polynomial 5 (example) within range r2
fit5 = B.polyfit(A[r2], C[r2], db[r2], 5)
B.plot_line(fit5.xpl, fit5.ypl)


speed = fit4.parameters[1].value #for speed = fit4.par[1]
#coefficient value (slope)
```

```
d_speed = fit4.parameters[1].err #or d_speed = fit4.sig_par[1]
#uncertainty value in the coefficient

#printing out the fitting parameters with proper significant digits
print("\nspeed is %3E +/- %3E m/s \n" % (speed, d_speed))
# Print results in scientific notation
```

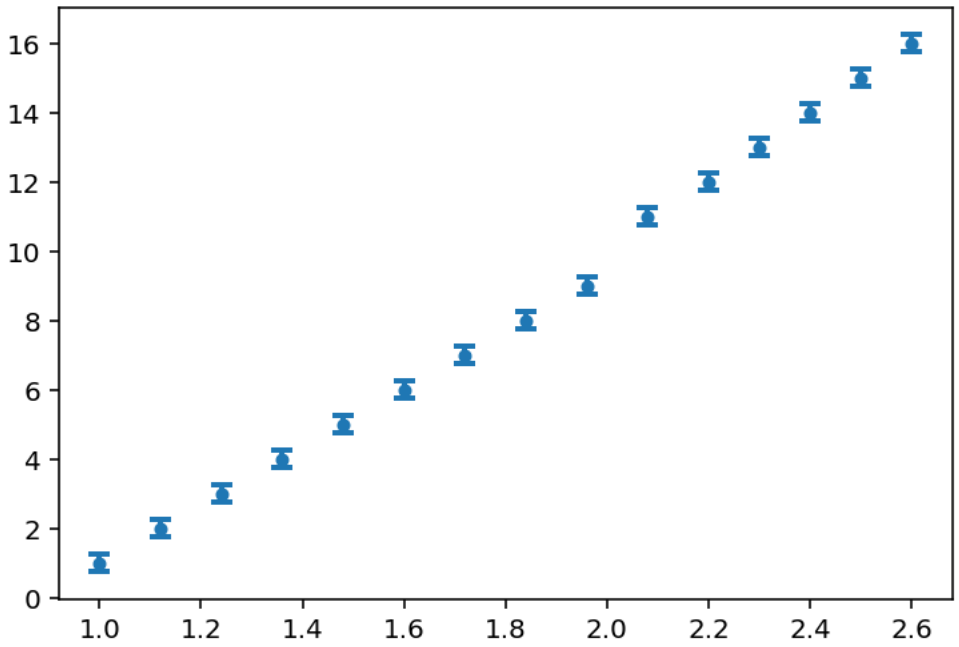
```
In [26]: # practiging fitting plots with various functionsimport numpy
as np

In [27]: import numpy as np

In [28]: import LT.box as B

In [28]:

In [29]: file_name = 'examples_data.py' #set name of data file

In [30]: f = B.get_file(file_name) #open and read the file

In [31]: # get the data

In [32]: # current

In [32]:

In [33]: #extract the data from the file and save them into their
respective variables

In [34]: A = B.get_data(f,'A')

In [35]: b = B.get_data(f,'b')

In [36]: db = B.get_data(f, 'db')

In [37]: C = B.get_data(f,'C')

In [38]: D = B.get_data(f, 'D')

In [39]: # The following examples fitl to fit fits C vs A, using linear
fit,

In [40]: #polynomial fits, in either the whole ranges (fit1, fit2) or a
subrange (fit 3, 4)

In [41]: B.plot_exp(A, C, db) #plot the data with error bars (db)
Out[41]: <ErrorbarContainer object of 3 artists>
```

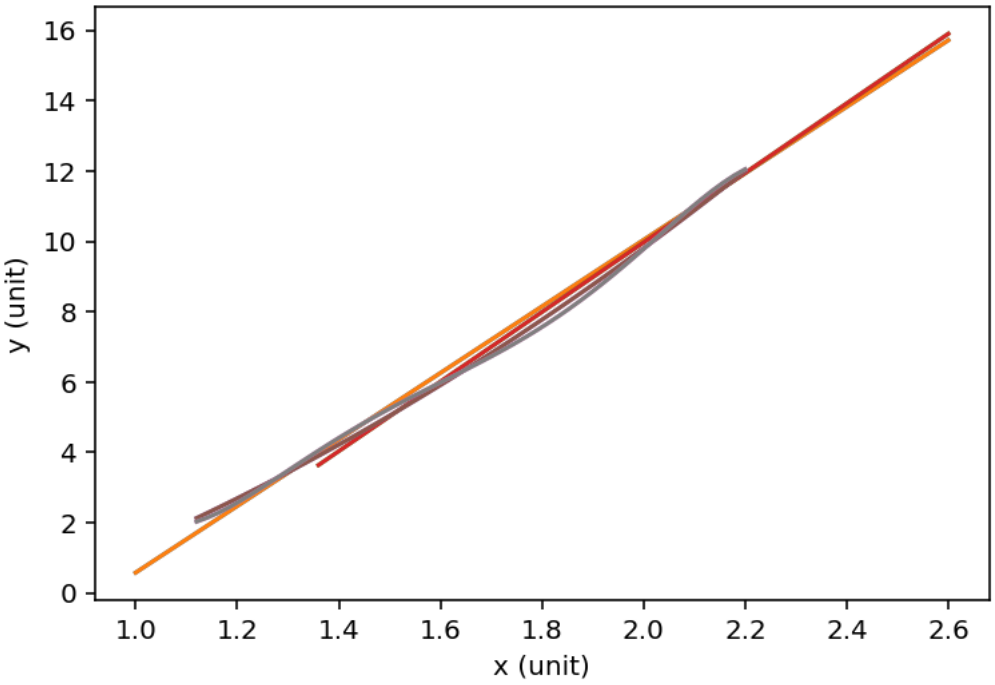
Important
Figures are displayed in the Plots pane by default. To make them also appear inline in the console, you need to uncheck "Mute inline plotting" under the options menu of Plots.

```
In [42]: B.pl.show() #display the plot

In [42]:

In [43]: #You should uncomment the next line to see what it does to the
x-axi

In [44]: #Can you set the y-axis title now?

In [45]: B.pl.xlabel("x (unit)") # sets the x-axis label
Out[45]: Text(0.5, 0, 'x (unit)')

In [46]: B.pl.ylabel("y (unit)")
Out[46]: Text(0, 0.5, 'y (unit)')

In [46]:

In [47]: fit1 = B.linefit(A, C, db) #does the linear fit of the entire
dataset
chisq/dof =  1.7887470965978935
offset  =  -8.886820603907685  +/-  0.24439543707385852
slope =  9.463587921847273  +/-  0.1290421055659006

In [48]: B.plot_line(fit1.xpl, fit1.ypl) #plots the fitted line
Out[48]: [<matplotlib.lines.Line2D at 0x15f60a110>]

In [48]:

In [49]: #the following two lines selecting the ranges

In [50]: r1 = B.in_between(4.0, 18.0, C) #. in window should be changed
to in _between

In [51]: r2 = B.in_between(2.0, 12.0, C) #creates a variable with the
data points of C

In [52]: #between 1 and 12

In [52]:

In [53]: #only fit the selected ranges as specified by r1 or r2

In [54]: fit3 = B.linefit(A[r1], C[r1], db[r1]) #linear fit using data
from range 1
chisq/dof =  1.2245662034703695
offset  =  -9.81841745266039  +/-  0.375043497129335
slope =  9.892720858233144  +/-  0.18371099586622422

In [55]: B.plot_line(fit3.xpl, fit3.ypl) #plot it
Out[55]: [<matplotlib.lines.Line2D at 0x15f73e050>]
```

```
In [55]:

In [56]: #the fit below use second order-- defined by the "2" in the
argument

In [57]: #polynomial; you should change 2 to other integers and see how
the

In [58]: #fits are different

In [59]: fit4 = B.polyfit(A[r2], C[r2], db[r2], 2)
chisq/dof =  0.7757575757575722
parameter [ 0 ] =  -2.9259259259254775  +/-  2.029820013068078
parameter [ 1 ] =  2.1548821548819577  +/-  2.5188754443260843
parameter [ 2 ] =  2.104377104377827  +/-  0.7555453817525889

In [60]: B.plot_line(fit4.xpl, fit4.ypl)
Out[60]: [<matplotlib.lines.Line2D at 0x15f7f09d0>]

In [60]:

In [61]: #for polynomial 5 (example) within range r2

In [62]: fit5 = B.polyfit(A[r2], C[r2], db[r2], 5)
chisq/dof =  0.6414918427924733
parameter [ 0 ] =  415.9043355282652  +/-  399.0864134652094
parameter [ 1 ] =  -1363.0645292471067  +/-  1266.7850482018996
parameter [ 2 ] =  1749.8791172516821  +/-  1586.3094883400727
parameter [ 3 ] =  -1098.668405712301  +/-  979.9029609901401
parameter [ 4 ] =  339.3010443124799  +/-  298.73046810568184
parameter [ 5 ] =  -41.218238158485356  +/-  35.97382499319897

In [63]: B.plot_line(fit5.xpl, fit5.ypl)
Out[63]: [<matplotlib.lines.Line2D at 0x1680d0ed0>]

In [63]:

In [63]:

In [64]: speed = fit4.parameters[1].value #for speed = fit4.par[1]

In [65]: #coefficient value (slope)

In [65]:

In [66]: d_speed = fit4.parameters[1].err #or d_speed = fit4.sig_par[1]

In [67]: #uncertainty value in the coefficient

In [67]:
```

```
In [68]: #printing out the fitting parameters with proper significant
digits

In [69]: print("\nspeed is %3E +/- %3E m/s \n" % (speed, d_speed))

speed is 2.154882E+00 +/- 2.518875E+00 m/s


In [70]: # Print results in scientific notation

In [70]:

In [70]:

In [70]:

In [71]:
```


16
14
12
10
8
6
4
2
0
1.0
1.2
1.4
1.6
1.8
2.0
2.2
2.4
2.6


16
14
12
10
8
6
4
2
0
y (unit)
1.0
1.2
1.4
1.6
1.8
2.0
2.2
2.4
2.6
x (unit)

```
#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
import scipy.special as sp

file_name = 'examples_data.py' #save data name
f = B.get_file(file_name) #open data
# get the data, extract the columns and save them in their corresponding variable
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',
                                   # initialized to 1.
c3 = P.Parameter(1., 'pol2')      # parameter c3, called 'pol2',
                                   #initialized to 1.
height = P.Parameter(3., 'height')
mean = P.Parameter(5., '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)/(2*sigma
    return value  # Check if this Gaussian function is defined correctly. IF not,

#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 = d

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_between(2.0, 12.0, C)

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

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')
norm = P.Parameter(10., 'norm')

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

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

fit8 = G.genfit(myfun2, [c1, c2, c3, mu, norm], x = C[r4], y = D[r4], y_err = db[

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',
```

```
                                    #initialized 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.)

#Notice c2 is set, and is no longer in the list of parameters 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 further analysis,
#for example
muvalue = mu.value
muerr = mu.err
print ("mu has the value of %f +- %f\n" %(muvalue, muerr))
```

```
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))
```

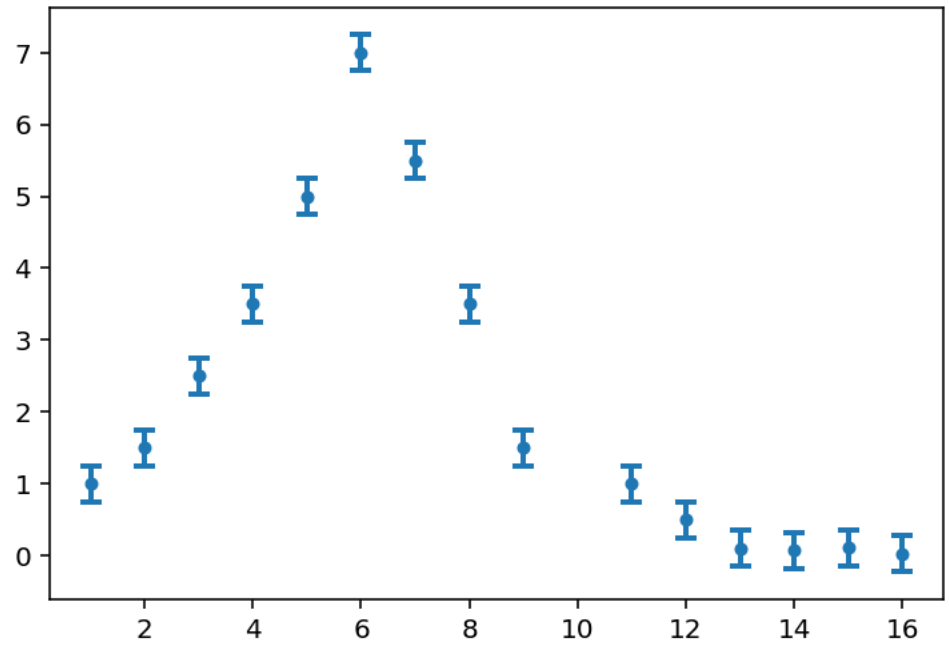
```
     ...:     return value  # Check if this Gaussian function is defined
correctly. IF not, correct it!
     ...:

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]:
```

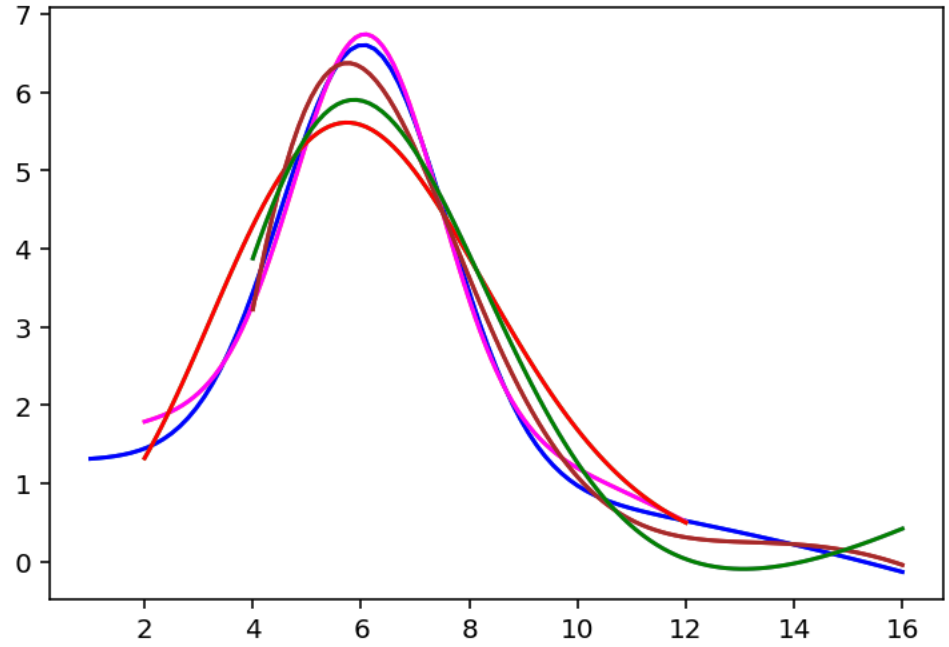
```
In [279]: fit6 = G.genfit(myfun, [c1, c2, c3, height, mean, sigma], x =
C[r3], y = D[r3], y_err = db[r3])
gen_fit kwargs =  {}
gen_fit.fit kwargs =  {}
Calculate numerical parameter derivatives with diff_step = 0.001
------------------------------------------------------------------------
fit results :
------------------------------------------------------------------------
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]:
```


7
6
5
4
3
2
1
0
2
4
6
8
10
12
14
16


7
6
5
4
3
2
1
0
2
4
6
8
10
12
14
16