```
# practicing fitting histograms
# As is true in many programming languages, a goal can be achieved
# by many methods
# The below example shows different approaches to histogram fitting
# including both the PDF Poisson example and class Gaussian work

import LT.box as B
import numpy as np
import LT_Fit.parameters as P  # get the parameter module
import LT_Fit.gen_fit as G     # load the genfit module
import scipy.special as ms     # need this for the factorial function
                              # called in myfunl(x) for Poisson fitting

# Pdf Poisson Fitting from Data File
file_name = 'histoFitting.data'
f = B.get_file(file_name)

# get the data from column 'A'
A = B.get_data(f, 'A')

# here histo is a method inside LT.box. There are other ways of
# making histograms as well
h2 = B.histo(A, (0.5, 10.5), 10)
# this means A is the data being histogramed, (0.5, 10.5) is the range,
# and 10 is number of bins, obviously the bin width is 1.

h2.plot()
# histograms can be fitted with the build in function for the
# Lt.box.histo object
# for example h2.fit() or h2.fit(2.0, 8.0) will fit the range between 2.0, and 8.
# However, this is limited
# the example below provides a way to fit histograms with any user
# defined function, including linear, polynomial, gaussian, poisson, etc.....


# put the bin centers and bin contents to two arrays
hx_poisson = h2.bin_center
hy_poisson = h2.bin_content
dy_poisson = np.sqrt(hy_poisson)
print("hy is:\n", hy_poisson)
print("dy is:\n", dy_poisson)

# Define parameters for Poisson function
mu = P.Parameter(2., 'mu')
norm = P.Parameter(10., 'norm')

# The function defined here is a pure poisson function. The initial
# "guess" of the parameters are given above
def myfunl(x):
   value = norm()*mu()**x*np.exp(-mu())/ms.gamma(x+1.0)
   return value
```

```
# you need to provide the independent variable x, and the yvalue at x
# in the forms of arrays. in this case, it's the bin_center (hx_poisson, as defin
# being x, and bin_content (hy_poisson) being y
# the way the fitting is done is the same as if you are simply fitting a Y vs X p

fit_poisson = G.genfit(myfunl, [mu, norm], x=hx_poisson, y=hy_poisson)

B.plot_line(fit_poisson.xpl, fit_poisson.ypl, color='red')
h2.plot()
B.pl.title('Poissin fit of the histofitting.data')
B.pl.show()



print("Poisson fit completed")

# From class: Gaussian Fitting from random ages

# Create an array with random ages from class work
age = ([23,19,25,20,18,17,26,30,17,21,22,25,27,7,37,47,19,24,25,20,22,
        30,20,26,27,20,])

# variable to create a histogram including the array age within the range
# (5, 50) separated in different number of 'bins'
# You can try different bin numbers by uncommenting:

# 5 bins version
# a1=B.histo(age, (5,50), 5)
# Plot the histogram
# a1.plot()

# Separate in 10 bins instead of 5
# a2=B.histo(age, (5,50), 10)
# plot
# a2.plot()

# Separate in 40 bins instead
a3 = B.histo(age, (5,40), 40)
# plot
a3.plot()
B.pl.title('Gaussian fit of the smaller age dataset')

#### fitting
# put the bin centers and bin contents to two arrays
hx = a3.bin_center  # Note: originally had hx=a3.bin_content which was wrong
hy = a3.bin_content
unc = np.sqrt(np.maximum(hy, 1.0))  # uncertainty calculation

# Alternative fitting methods (from class):
# fit histogram polynomial
```

```
# fith = B.polyfit(hx, hy, order=10)
# B.plot_line(fith.xpl, fith.ypl)
# regular fit
# fith = a3.fit()

# Extended age dataset for better statistics
age2 = ([23,19,25,20,18,17,26,30,17,21,22,25,27,7,37,47,19,24,25,20,22,
         30,20,26,27,20,47,45,39,45,23,36,44,36,24,33,56,32,49,35,35,45,
         54,49,56,75,42,63,45,34,36,48,48,38,24,52,34,24,62,54,32,36,45,34])

# Separate in 60 bins for the larger dataset
a4 = B.histo(age2, (5,60), 60)
# plot
a4.plot()
B.pl.title('Gaussian fit of the larger age dataset')

## fitting the extended dataset
# put the bin centers and bin contents to two arrays
hx2 = a4.bin_center
hy2 = a4.bin_content
unc2 = np.sqrt(np.maximum(hy2, 1.0))  # uncertainty

# fit histogram with Gaussian function
# Define parameters with better initial guesses based on your histogram
p1 = P.Parameter(5, 'p1')   # amplitude - should match peak height (~5)
p2 = P.Parameter(45, 'p2')  # mean/center - peak appears around age 45-50
p3 = P.Parameter(12, 'p3')  # standard deviation - data looks spread out

# define the gaussian function
def gauss(x):
    return p1() * np.exp(-0.5 * ((x - p2()) / p3())**2)

# create fitted with gauss variable - use the larger dataset (hx2, hy2)
fit = G.genfit(gauss, [p1, p2, p3], x=hx2, y=hy2, y_err=unc2)

# Perform the fit
fit.fit()

# Print results manually since show_results() doesn't exist
print("Gaussian fit results:")
print(f"Amplitude (p1): {p1():.3f}")
print(f"Mean (p2): {p2():.3f}")
print(f"Standard deviation (p3): {p3():.3f}")
print(f"Chi-squared: {fit.chi2:.3f}")

# Plot the fitted Gaussian curve using the same x data
B.plot_line(hx2, gauss(hx2), color='blue')

print("Gaussian fit completed!")
print(f"Final parameters - Amplitude: {p1():.2f}, Mean: {p2():.2f}, Std Dev: {p3(
```

```
B.pl.show()
```

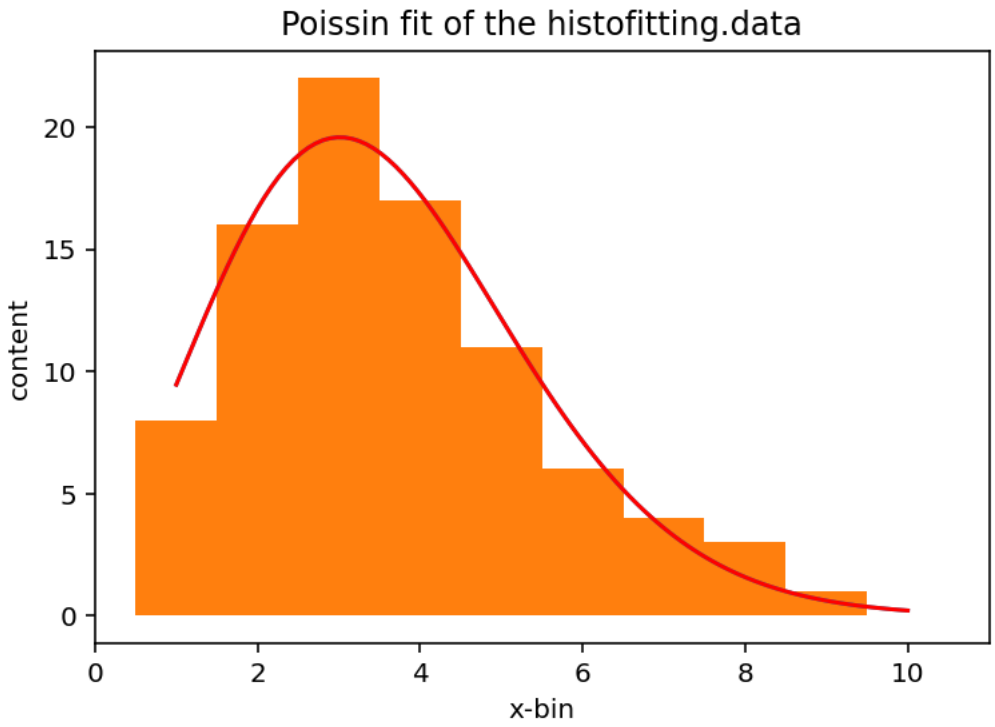
```
Python 3.11.13 | packaged by conda-forge | (main, Jun  4 2025, 14:52:34)
[Clang 18.1.8 ]
Type "copyright", "credits" or "license" for more information.

IPython 8.37.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: %runfile '/Users/catacisneros/Library/CloudStorage/OneDrive-
Personal/FIU/Fall 2025/Intermediate Physics Lab/SamplePrograms2/
untitled5.py' --wdir
hy is:
 [ 8 16 22 17 11  6  4  3  1  0]
dy is:
 [2.82842712 4.         4.69041576 4.12310563 3.31662479 2.44948974
 2.         1.73205081 1.         0.        ]
gen_fit kwargs =  {}
gen_fit.fit kwargs =  {}
Calculate numerical parameter derivatives with diff_step = 0.001
------------------------------------------------------------------------
fit results :
------------------------------------------------------------------------
chisquare =  13.501798022083067
red. chisquare =  1.6877247527603834
parameters:
parameter  0  :  mu = np.float64(3.524076582318037) +/-
np.float64(0.09837057896347122)
parameter  1  :  norm = np.float64(91.04106302717189) +/-
np.float64(3.3853510194359666)
```

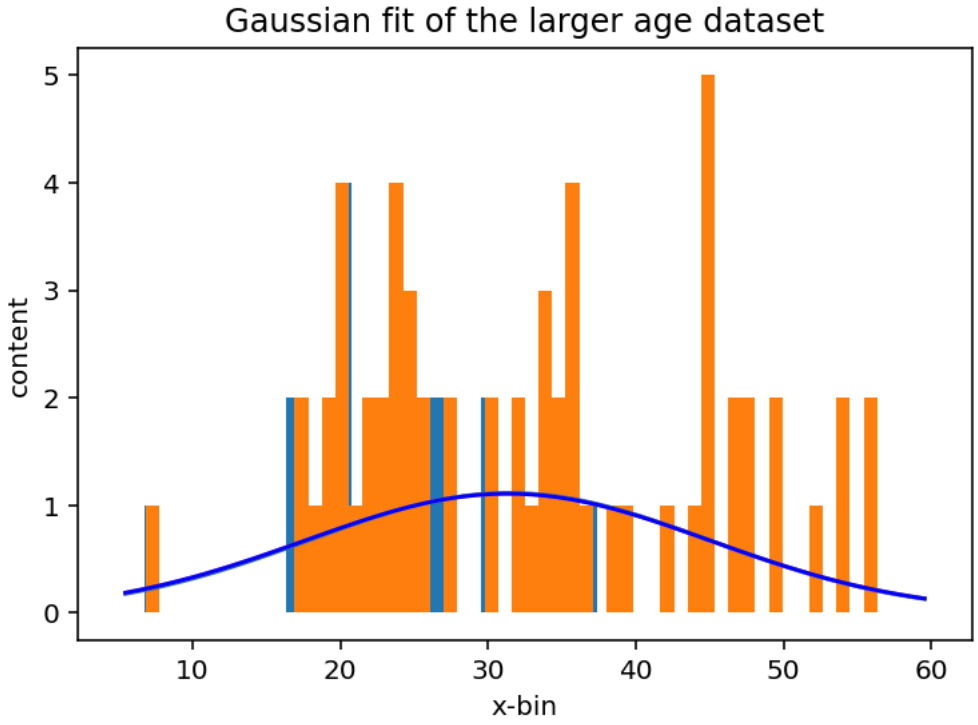
Important

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.

```
 Poisson fit completed
gen_fit kwargs =  {}
gen_fit.fit kwargs =  {}
Calculate numerical parameter derivatives with diff_step = 0.001
------------------------------------------------------------------------
fit results :
------------------------------------------------------------------------
chisquare =  37.07868301784079
red. chisquare =  0.6505032108393121
parameters:
parameter  0  :  p1 = np.float64(1.1145767857481084) +/-
np.float64(0.28736004648365304)
parameter  1  :  p2 = np.float64(31.441298847697603) +/-
np.float64(3.823989349585074)
parameter  2  :  p3 = np.float64(13.538174741487351) +/-
```

```
np.float64(4.009566818569393)
gen_fit.fit kwargs =  {}
Calculate numerical parameter derivatives with diff_step = 0.001
------------------------------------------------------------------------
fit results :
------------------------------------------------------------------------
chisquare =  37.0769009812381
red. chisquare =  0.6504719470392648
parameters:
parameter  0  :  p1 = np.float64(1.108311408809262) +/-
np.float64(0.28632522762532225)
parameter  1  :  p2 = np.float64(31.3247842979659) +/-
np.float64(3.8695787477841117)
parameter  2  :  p3 = np.float64(13.705217356433284) +/-
np.float64(4.088348489135589)
Gaussian fit results:
Amplitude (p1): 1.108
Mean (p2): 31.325
Standard deviation (p3): 13.705
Chi-squared: 37.077
Gaussian fit completed!
Final parameters - Amplitude: 1.11, Mean: 31.32, Std Dev: 13.71

In [2]:
```


Poissin fit of the histofitting.data
content
x-bin
0
5
10
15
20
0
2
4
6
8
10


Gaussian fit of the larger age dataset
content
x-bin
0
1
2
3
4
5
10
20
30
40
50
60

```
import LT.box as B
import numpy as np

# Loop and Array Examples - Building on Class Work
# This combines the PDF examples with practical applications

# PART 1: Quick Array Definition (from PDF)
# a lot of times when you are dealing with a few data points, you don't
# have to read them from a data file. You can quickly define using the
# np.array() --

x=np.array([1,2,3,4]) # pay attention to the parenthesis outside,
# and the bracket inside

y=np.array([2.1,4.0,5.9,7.5])

deltay=np.array([0.1,0.1,0.2,0.2])

B.plot_exp(x,y,dy=deltay)

# Now you made a plot very quickly without creating a data file first and
# then reading from it


################################
#What if you have multiple data files which you want to analyze similarly
# Then you will benefit from using loops
# For example, we have three data files ( yellow.dat, etc) with similar data
#definitions We can do the following

filenames=np.array(['yellow.dat', 'green.dat', 'blue.dat'])
range_low=np.array([1,2,3])
range_high=np.array([8,9,10])

#since there are three data files, the argument to the function range() below
# should be range[0,3]
for fileindex in range(0,3):
# Instead of range(0,3), you could also just use range(3)
# the filediex is an integer starting from 0, ending with 2
# veryify that by uncommenting the following statement
    #print ('fileindex is', fileindex)
    f = B.get_file(filenames[fileindex])
    #print (filenames[fileindex])
    A = B.get_data(f, 'A')
    #print (A)
    b = B.get_data(f, 'b')
    db = B.get_data(f, 'db')
    #B.pl.figure()    # this creats a new figure window for the plot
                     # if you comment the above out, then all graphs will
                     # will be in the same window
```

```
    B.plot_exp(A, b, dy=db)
    myrange=B.in_between(range_low[fileindex], range_high[fileindex],A)
    fit = B.linefit(A[myrange], b[myrange], db[myrange])
    B.plot_line(fit.xpl,fit.ypl)
    #the above define the fitting window in termes of variable A, using
    # the previously defined two arrays: range_low and range_high
    #don't forget that the [fileindex] argument; see comments befow

#What if you want to fit the graphs, but with a different range
# then you need to define the ranges (both lower and upper limits)
# in array as well, before the loop

# similarly, if you were to fit those data with different intial
# parameters, they should be pre-defined in arrays before the loop
# then access the corresponding array members in side the loop

# Notece all the statements in the loop should be indented; if there is no
#indent, then python assume the loop has ended

#If you did everything correctly, you shoudl see three graphs,
#almost straight lines, plotted on top of each other
# what if you want each graph to be plotted on a different canvas?


#### So far we addressed 1. quick defintion of arrays with few data points
#######################2. using loop to access multiple files and perform
########################### similar analyses


##### Next we have a simple example of quickly assigning values to
#array members using loops
####### Let's say three arrays of 20 mebers (index from 0 the 19)
####### newx, newy, newdy
### there are different ways of doing this; blow is a simple version
### first you need to create the empty array,
newx=np.empty(20)    #if you use newx=np.array([]), and then try to access
                     # the members of newx[index],even if index is in range
                     # it won't work; since those array members dont' exist yet
newy=np.empty(20)

newdy=np.empty(20)

for array_index in range(20):
    newx[array_index]=array_index ### don't forget the [array_index]
                                  #  part when accessing the array members
    # assume that newx is angle in radian
    newy[array_index]=np.cos(array_index)

    #newdy[array_index]=np.sqrt(newy[array_index])
    newdy[array_index]=0.2
B.pl.figure()
```

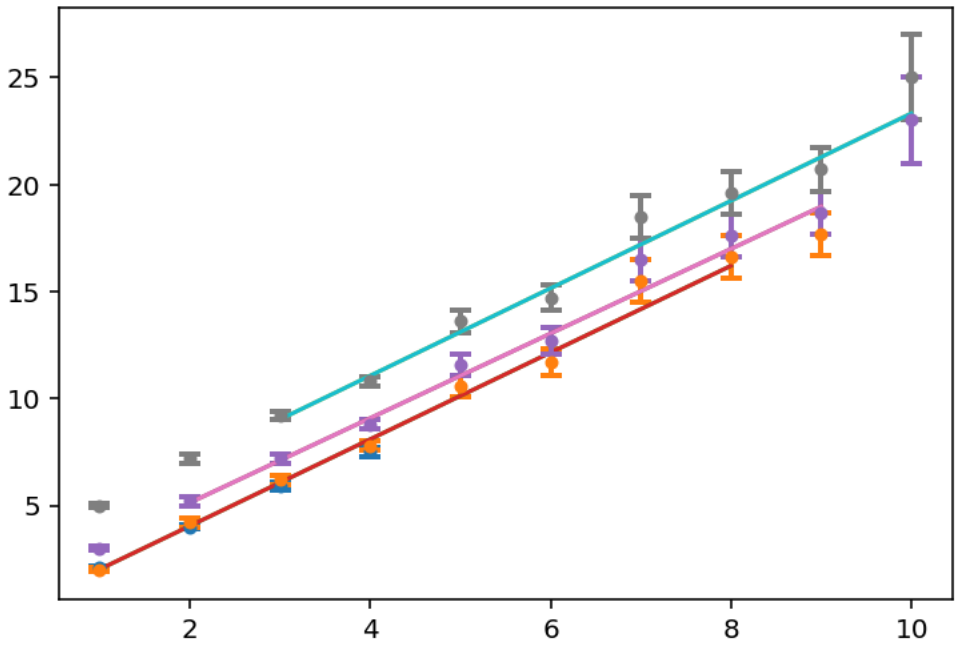
```
B.plot_exp(newx, newy, dy=newdy)

# Let's try to fit this graph with 8th order polynomial (even though
#we know it is basically y=cos(x)

r1=B.in_between(2,15,newx)
fit2 = B.polyfit(newx[r1], newy[r1], yerr=newdy[r1], order=8)
B.plot_line(fit2.xpl, fit2.ypl)

#you could comment out the line below if you just want to check
#if the 8th order polynomial worked
WiderX=np.linspace(0,20,100) #This linespace function creates an array with
                             #100 members from 0 to 20
WiderY=fit2.poly(WiderX)+1 #WE are accessing the fitting results/8th order
#polynomial functions from 3  lines above using fit2.poly()

# The "+1" above create an offset of 1 so you can separate the two lines

B.plot_line(WiderX, WiderY)
#You can see that the fit function overlaps with teh original fit2 line in
#the range of 2 to 15, where you defined the orginal fit2. However,
#it differs dramatically outside that range
# what did you learn from this exercise?
#Loops are faster than copy-pasting teh same code. Arrays let you handle
# different settings for each file automatically.
#Polynomial fits can look good but can looks bad outside the data range
```

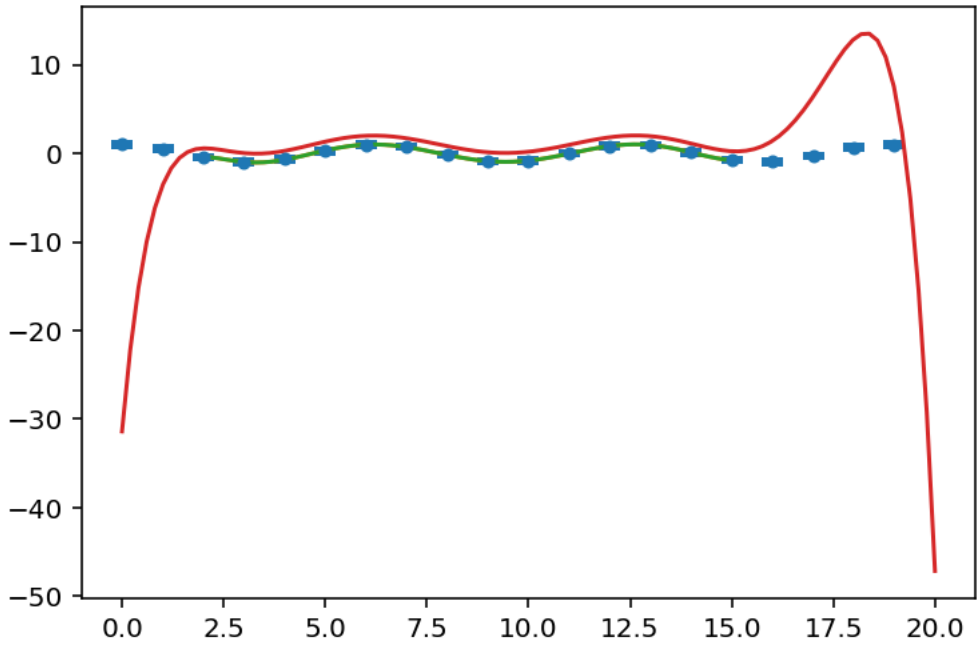
```
Python 3.11.13 | packaged by conda-forge | (main, Jun  4 2025, 14:52:34)
[Clang 18.1.8 ]
Type "copyright", "credits" or "license" for more information.

IPython 8.37.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: %runfile '/Users/catacisneros/Library/CloudStorage/OneDrive-
Personal/FIU/Fall 2025/Intermediate Physics Lab/SamplePrograms2/
untitled6.py' --wdir
chisq/dof =  1.1071299398256451
offset  =  -0.006249256630853744  +/-  0.13007337079675216
slope =  2.0260578968930485  +/-  0.05227135543340129
chisq/dof =  1.08030003295684
offset  =  1.176074169655135  +/-  0.2904883536291863
slope =  1.9770343868351352  +/-  0.07995666956719942
chisq/dof =  1.1596386473745905
offset  =  2.8938269413155204  +/-  0.41903566189941205
slope =  2.042995099115601  +/-  0.10047936198580754
chisq/dof =  0.08424938921481875
parameter [ 0 ] =  -32.399792195221345  +/-  22.05720929625515
parameter [ 1 ] =  51.6683923050961  +/-  31.52502453355677
parameter [ 2 ] =  -32.58488893606334  +/-  18.103613241005352
parameter [ 3 ] =  10.39281770325195  +/-  5.510967860286315
parameter [ 4 ] =  -1.84956946251974  +/-  0.9817041046086912
parameter [ 5 ] =  0.19107815696891758  +/-  0.10564824286122872
parameter [ 6 ] =  -0.011361155429523683  +/-  0.006754932056785228
parameter [ 7 ] =  0.0003599552659064643  +/-  0.00023601428552005413
parameter [ 8 ] =  -4.699433359102725e-06  +/-  3.4676937162592123e-06
```

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 [2]:
```


25
20
15
10
5
2
4
6
8
10


10
0
−10
−20
−30
−40
−50
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0