```
# -*- coding: utf-8 -*-

# This Python-program was developed using "Enthought Canopy v. 1.4.1
# environment, on a MacBook Pro running OS X 10.10
#
# written by T. Ihn, D-PHYS ETH Zurich, 4 Nov 2014

#
# Commands from libraries
#
from csv import reader

from numpy import array
from numpy import exp
from numpy import log
from numpy import sqrt
from numpy import multiply
from numpy import sum
from numpy import arange
from numpy import zeros
from numpy import meshgrid
from numpy.linalg import inv

from scipy.misc import factorial

from matplotlib import rcParams

from matplotlib.pyplot import figure
from matplotlib.pyplot import axes
from matplotlib.pyplot import bar
from matplotlib.pyplot import contourf
from matplotlib.pyplot import contour
from matplotlib.pyplot import colorbar
from matplotlib.pyplot import clabel
from matplotlib.pyplot import plot
from matplotlib.pyplot import axis
from matplotlib.pyplot import xticks
from matplotlib.pyplot import yticks
from matplotlib.pyplot import tick_params
from matplotlib.pyplot import xlabel
from matplotlib.pyplot import ylabel
from matplotlib.pyplot import title
from matplotlib.pyplot import show
```

```
from scipy.optimize import fmin

#
# read data from a csv-file
#
readerout = reader(open("histdata.csv","rb"),delimiter=',');
x = list(readerout);
data = array(x).astype('float')
xi = data[:,0]
ci = data[:,1]

#
# Plot data
#
figure(
    num=1,
    figsize=(7,5.08),
    dpi=80,
    facecolor='white')
rcParams['axes.linewidth']=2
axes([0.15,0.18,0.82,0.70],
    axisbg='lightgray')
bar(xi-0.25,ci,width=0.5)
axis([-10.5,10.5,0,55])
xticks([-10,-5,0,5,10],
    fontsize=24,
    fontname='Times New Roman')
yticks([0,10,20,30,40,50],
    fontsize=24,
    fontname='Times New Roman')
tick_params(width=2,length=10)
xlabel(r'$(E-E_0)/\Delta$',
    fontsize=24)
ylabel(r'Counts',
    fontsize=24)
title('Energy resolved counting data',fontsize=24)
show()

#
# Function calculating the expected number of counts in a bin given
#
def mu(A,B,x):
```

```
    return (A*exp(-x**2) + B)/2.

#
# Function calculating the negative log-likelihood for given A,B
#
def L((A,B),xi,ci):
    mui = mu(A,B,xi)
    return -sum(multiply(ci,log(mui)) - mui)

#
# Minimization of the negative log-likelihood
#
parms,Lmin,_,_,_ = fmin(L,[100,20],args=(xi,ci),full_output=True)
A0 = parms[0]
B0 = parms[1]

#
# Plot the negative log-likelihood function
#
AA = arange(75,110,35./51.)
BB = arange(18,22.2,4.2/51.)
LL = zeros(shape=(len(AA),len(BB)))
X,Y = meshgrid(AA,BB)
for n in range(0,len(AA)):
    for m in range(0,len(BB)):
        LL[m,n] = 2*(L([AA[n],BB[m]],xi,ci)-Lmin)

figure(
    num=2,
    figsize=(7,5.08),
    dpi=80,
    facecolor='white')
rcParams['axes.linewidth']=2
axes([0.15,0.18,0.82,0.70],
    axisbg='lightgray')
CS = contourf(X,Y,LL,10)
# We draw two contour lines into the plot, the smaller of which is s
CS2 = contour(CS,levels=[0.2,1,2,3],colors=('w','r','w','w'),linewid
clabel(CS2, inline=1, fmt='%1.1f', fontsize=16)
plot(A0,B0,'wo')
title('Negative log-likelihood',fontsize=24)
xlabel(r'$A$',
    fontsize=24)
```

```
ylabel(r'$B$',
    fontsize=24)
cbar = colorbar(CS)
cbar.ax.set_ylabel(r'$2(L(A,B)-L_\mathrm{min})$',fontsize=24)
show()

# We determine the standard error of the parameters from the contour
cc = CS2.collections[1].get_paths()[0]
CC = cc.vertices
sAm = A0-min(CC[:,0])
sAp = max(CC[:,0])-A0
sBm = B0-min(CC[:,1])
sBp = max(CC[:,1])-B0

print('A = %1.1f +%1.1f/-%1.1f'% (A0,sAp,sAm))
print('B = %1.1f +%1.1f/-%1.1f'% (B0,sBp,sBm))

#
# Plot the result of the fit
#
x = arange(-10.5,10.5,0.1)
figure(
    num=3,
    figsize=(7,5.08),
    dpi=80,
    facecolor='white')
rcParams['axes.linewidth']=2
axes([0.15,0.18,0.82,0.70],
    axisbg='lightgray')
bar(xi-0.25,ci,width=0.5)
plot(x,mu(A0,B0,x),'r',linewidth=2)
axis([-10.5,10.5,0,60])
xticks([-10,-5,0,5,10],
    fontsize=24,
    fontname='Times New Roman')
yticks([0,10,20,30,40,50,60],
    fontsize=24,
    fontname='Times New Roman')
tick_params(width=2,length=10)
xlabel(r'$(E-E_0)/\Delta$',
    fontsize=24)
ylabel(r'Counts',
    fontsize=24)
```

```
title('Energy resolved counting data',fontsize=24)
show()

#
# Determine the elements of the Hesse matrix numerically
#
h = 1.;
k = 0.1;
d2LdA2 = (L([A0+h,B0],xi,ci) - 2*L([A0,B0],xi,ci) + L([A0-h,B0],xi,c
d2LdB2 = (L([A0,B0+k],xi,ci) - 2*L([A0,B0],xi,ci) + L([A0,B0-k],xi,c
d2LdAdB = (L([A0+h,B0+k],xi,ci) - L([A0+h,B0-k],xi,ci) - L([A0-h,B0+

# invert the Hesse matrix
cov = inv([[d2LdA2,d2LdAdB],[d2LdAdB,d2LdB2]])
sigmaA = sqrt(cov[0,0])
sigmaB = sqrt(cov[1,1])
rho = cov[0,1]/sigmaA/sigmaB
print('\nFrom Hesse matrix inversion:')
print('A = %1.1f +/-%1.1f'% (A0,sigmaA))
print('B = %1.1f +/-%1.1f'% (B0,sigmaB))
print('rho = %1.2f'% (rho))
```