ENERGY_RESOLUTION1

September 8, 2020

1 LABORATORIO DI FISICA MEDICA:

ESERCIZIO DI STIMA DELLA RISOLUZIONE ENERGETICA DI RIVELATORI A SCINTILLAZIONE

[76]:
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
# Bellurie per LaTeX
#from matplotlib import rcParams
#matplotlib.rc('font',**{'family':'serif','serif':['Palatino']})
#plt.rc('text', usetex=True)
#plt.rcParams['text.latex.preview'] = True
from scipy.optimize import curve_fit
from scipy import stats
```

1.1 CARICAMENTO DEI DATI

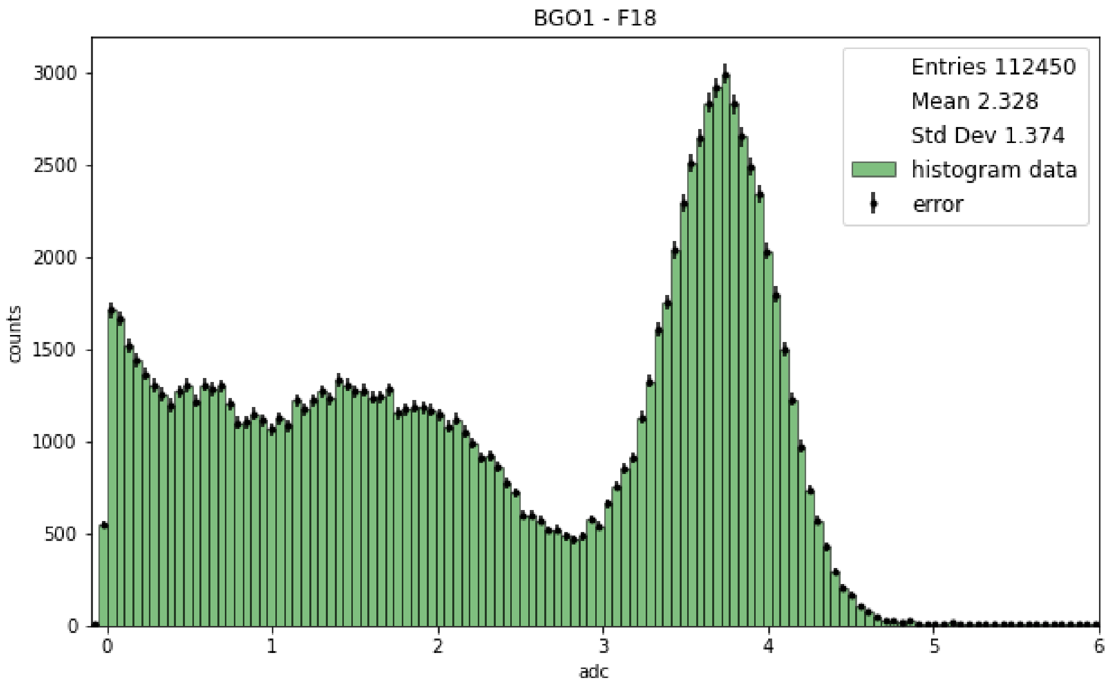
[77]:
```
# BGO - Fluoro18
BGO1F18 = np.loadtxt('DATA/dataBGO1F18.txt')
BGO2F18 = np.loadtxt('DATA/dataBGO2F18.txt')
BGO3F18 = np.loadtxt('DATA/dataBGO3F18.txt')

# LSO - Fluoro18
LSO1F18 = np.loadtxt('DATA/dataLSO1F18.txt')
LSO2F18 = np.loadtxt('DATA/dataLSO2F18.txt')
LSO3F18 = np.loadtxt('DATA/dataLSO3F18.txt')

# LSO - Bario133
LSO1Ba133 = np.loadtxt('DATA/dataLSO1Ba133.txt')
LSO2Ba133 = np.loadtxt('DATA/dataLSO2Ba133.txt')

# NaI - Tecnezio99m
NaI1Tc99m = np.loadtxt('DATA/dataNaI1Tc99m.txt')
NaI2Tc99m = np.loadtxt('DATA/dataNaI2Tc99m.txt')
NaI3Tc99m = np.loadtxt('DATA/dataNaI3Tc99m.txt')
```

1.2 Istogramma (semplice)

Per ognuno dei files

1.2.1 BGO1 - ^{18}F

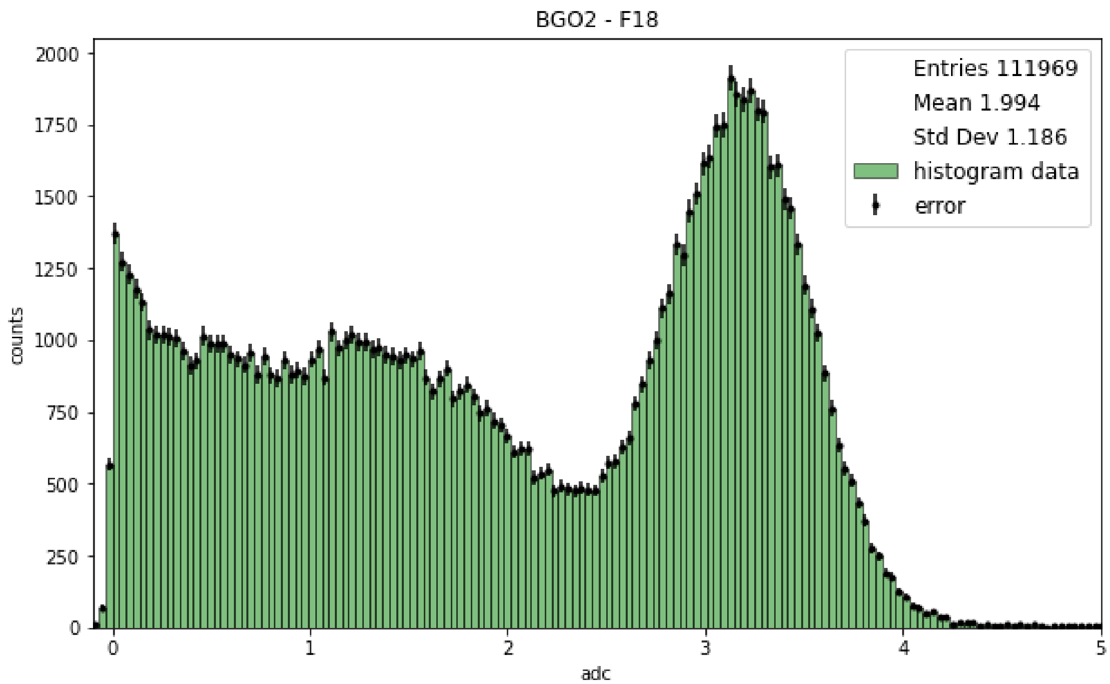
[115]:
```
# BGO1 - F18

# Entries, Media, Standard deviation, Errore sulla media
entries    = len(BGO1F18)
mean, std  = np.mean(BGO1F18), np.std(BGO1F18)
mean_error = std/np.sqrt(len(BGO1F18))

# Creazione dell'istogramma (con barre di errore)
fig                          = plt.figure(1, figsize = (10,6))
bin_heights, bin_borders, _ = plt.
 ↪hist(BGO1F18,bins=200,facecolor='g',ec='black',alpha=0.5,label='histogram␣
 ↪data',density=False)
bin_centers                  = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black', label='error')

# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('BGO1 - F18')
plt.xlim(-0.1, 6)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Entries %.
 ↪i'   %entries)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Mean %.3f'␣
 ↪    %mean)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Std Dev %.
 ↪3f' %std)
plt.legend(fancybox=True, shadow=False, loc='best', prop={"size":12}, numpoints␣
 ↪= 1)
plt.show()
```



1.2.2 BGO2 - ^{18}F

[116]:
```
# BGO2 - F18

# Entries, Media, Standard deviation, Errore sulla media
entries    = len(BGO2F18)
mean, std  = np.mean(BGO2F18), np.std(BGO2F18)
mean_error = std/np.sqrt(len(BGO2F18))

# Creazione dell'istogramma (con barre di errore)
fig                               = plt.figure(figsize = (10,6))
bin_heights, bin_borders, _ = plt.
 ↪hist(BGO2F18,bins=200,facecolor='g',ec='black',alpha=0.5,label='histogram␣
 ↪data',density=False)
bin_centers                       = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                     = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black', label='error')

# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('BGO2 - F18')
plt.xlim(-0.1, 5)
```

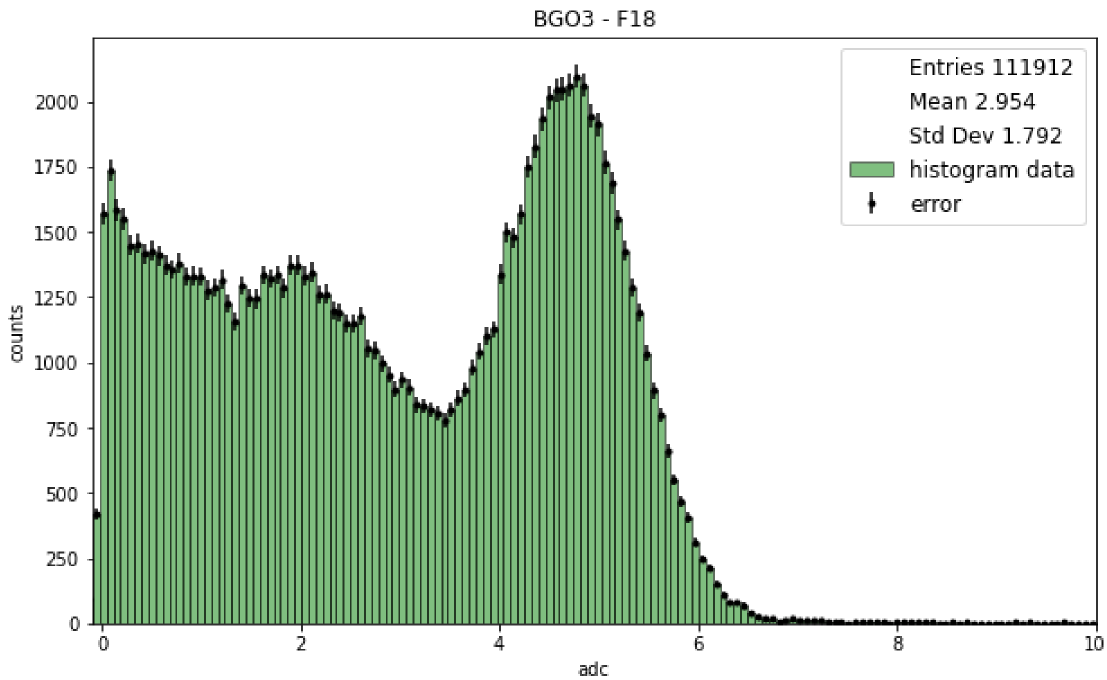
```
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Entries %.
↪i'   %entries)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Mean %.3f'␣
↪    %mean)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Std Dev %.
↪3f' %std)
plt.legend(fancybox=True, shadow=False, loc='best', prop={"size":12}, numpoints␣
↪= 1)
plt.show()
```



1.2.3 BGO3 - ^{18}F

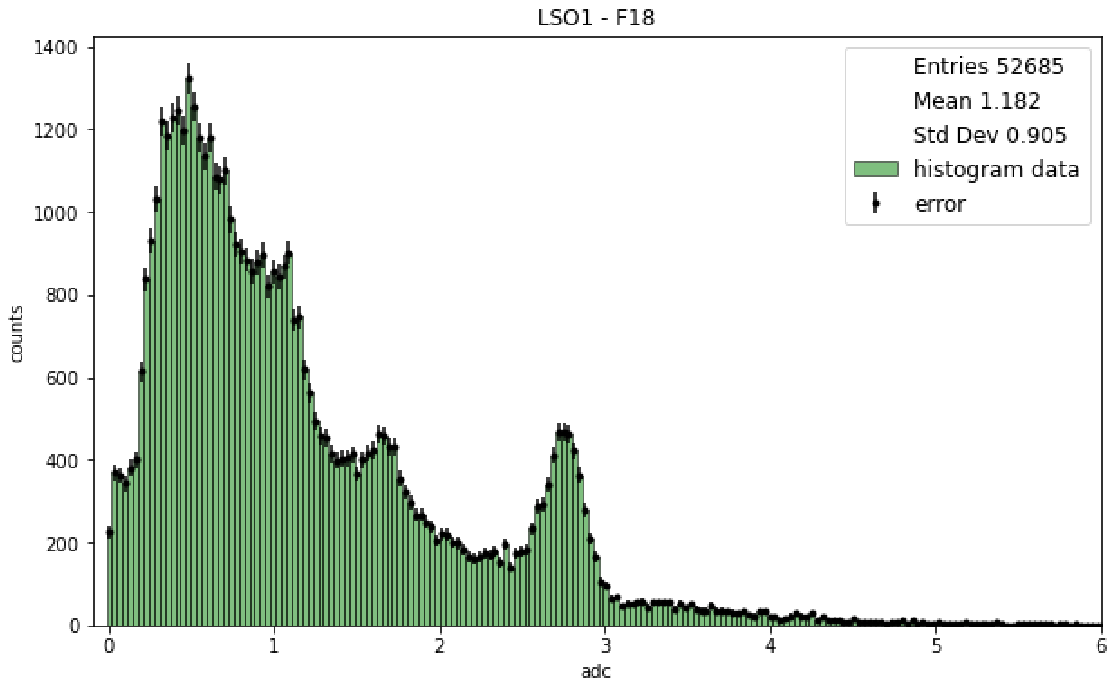
[117]:
```
# BGO3 - F18

# Entries, Media, Standard deviation, Errore sulla media
entries     = len(BGO3F18)
mean, std   = np.mean(BGO3F18), np.std(BGO3F18)
mean_error = std/np.sqrt(len(BGO3F18))

# Creazione dell'istogramma (con barre di errore)
fig                             = plt.figure(figsize=(10,6))
bin_heights, bin_borders, _ = plt.
↪hist(BGO3F18,bins=200,facecolor='g',ec='black',alpha=0.5,label='histogram␣
↪data',density=False)
```

```
bin_centers                  = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black', label='error')

# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('BGO3 - F18')
plt.xlim(-0.1, 10)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Entries %.
 ↪i'   %entries)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Mean %.3f'␣
 ↪    %mean)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Std Dev %.
 ↪3f' %std)
plt.legend(fancybox=True, shadow=False, loc='best', prop={"size":12}, numpoints␣
 ↪= 1)
plt.show()
```



1.2.4 LSO1 - ^{18}F

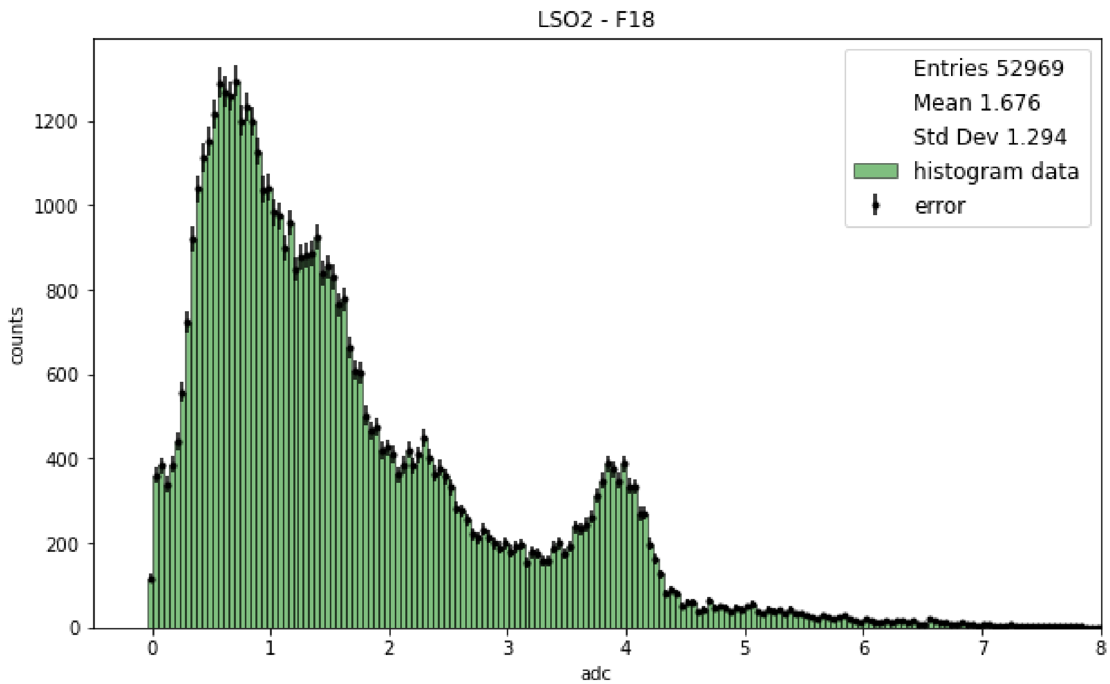
[118]:
```
# LSO1 - F18

# Entries, Media, Standard deviation, Errore sulla media
entries    = len(LSO1F18)
mean, std  = np.mean(LSO1F18), np.std(LSO1F18)
mean_error = std/np.sqrt(len(LSO1F18))

# Creazione dell'istogramma (con barre di errore)
fig                          = plt.figure(figsize=(10,6))
bin_heights, bin_borders, _ = plt.
 ↪hist(LSO1F18,bins=200,facecolor='g',ec='black',alpha=0.5,label='histogram␣
 ↪data',density=False)
bin_centers                  = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black', label='error')

# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('LSO1 - F18')
plt.xlim(-0.1, 6)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Entries %.
 ↪i'   %entries)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Mean %.3f'␣
 ↪    %mean)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Std Dev %.
 ↪3f' %std)
plt.legend(fancybox=True, shadow=False, loc='best', prop={"size":12}, numpoints␣
 ↪= 1)
plt.show()
```



1.2.5 LSO2 - ^{18}F

[119]:
```
# LSO2 - F18

# Entries, Media, Standard deviation, Errore sulla media
entries     = len(LSO2F18)
mean, std   = np.mean(LSO2F18), np.std(LSO2F18)
mean_error = std/np.sqrt(len(LSO2F18))

# Creazione dell'istogramma (con barre di errore)
fig                              = plt.figure(figsize=(10,6))
bin_heights, bin_borders, _ = plt.
 ↪hist(LSO2F18,bins=200,facecolor='g',ec='black',alpha=0.5,label='histogram␣
 ↪data',density=False)
bin_centers                      = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                    = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black', label='error')

# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('LSO2 - F18')
plt.xlim(-0.5, 8)
```

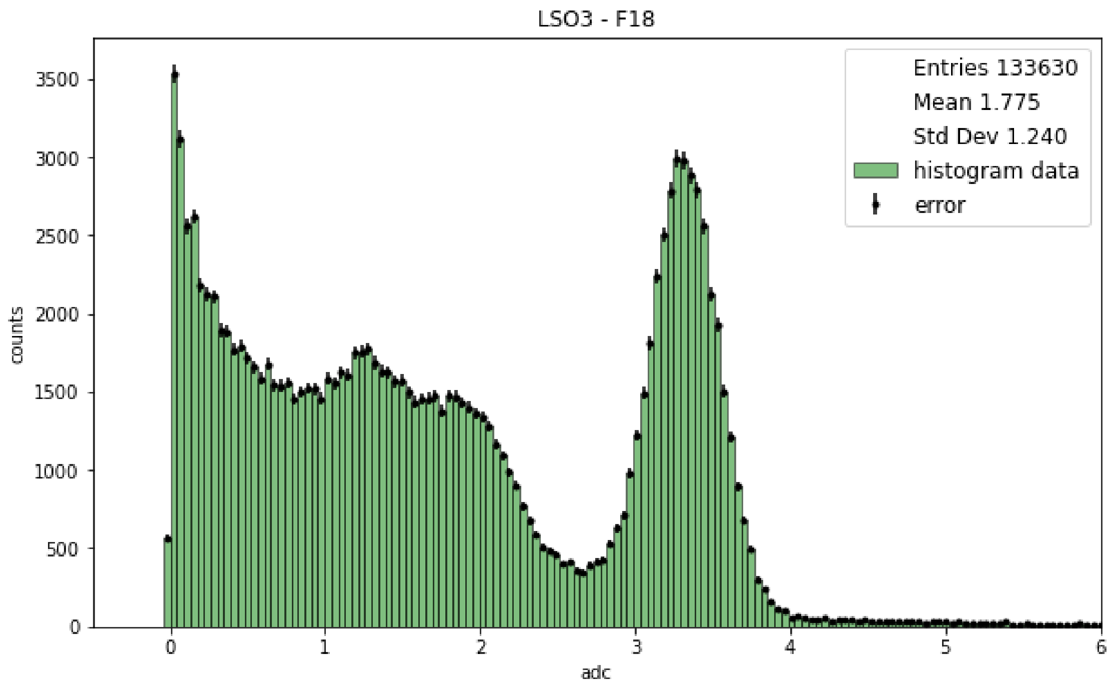
```
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Entries %.
    ↪i'   %entries)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Mean %.3f'␣
    ↪    %mean)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Std Dev %.
    ↪3f' %std)
plt.legend(fancybox=True, shadow=False, loc='best', prop={"size":12}, numpoints␣
    ↪= 1)
plt.show()
```



1.2.6 LSO3 - ^{18}F

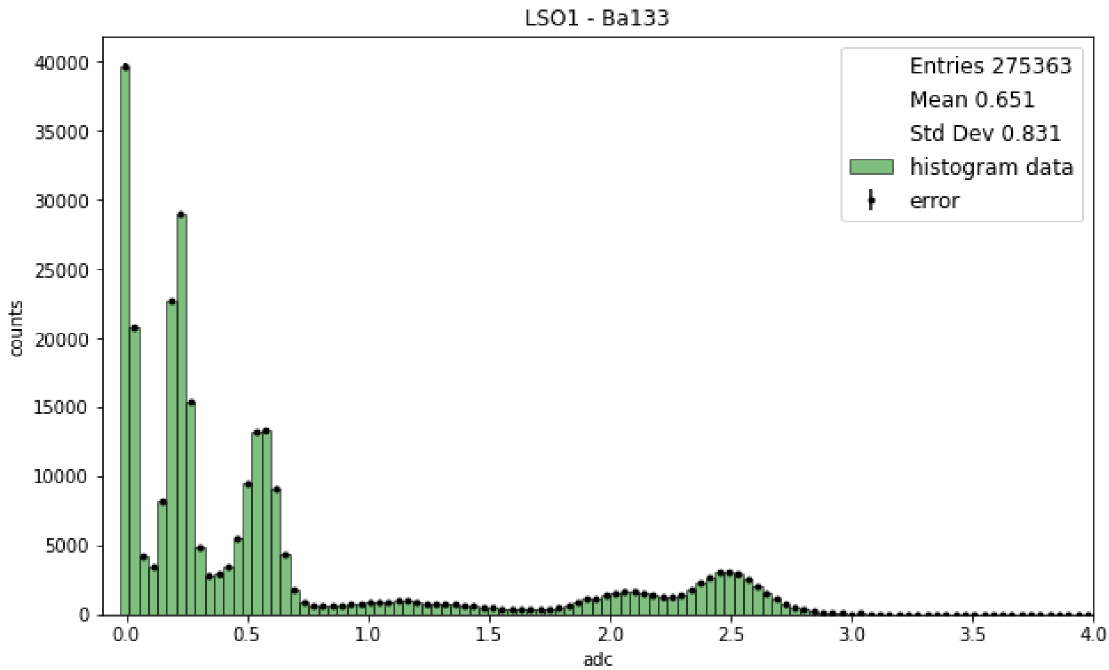
[120]:
```
# LSO3 - F18

# Entries, Media, Standard deviation, Errore sulla media
entries     = len(LSO3F18)
mean, std   = np.mean(LSO3F18), np.std(LSO3F18)
mean_error = std/np.sqrt(len(LSO3F18))

# Creazione dell'istogramma (con barre di errore)
fig                            = plt.figure(figsize=(10,6))
bin_heights, bin_borders, _ = plt.
    ↪hist(LSO3F18,bins=200,facecolor='g',ec='black',alpha=0.5,label='histogram␣
    ↪data',density=False)
```

```
bin_centers                    = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                  = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black', label='error')

# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('LS03 - F18')
plt.xlim(-0.5, 6)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Entries %.
 ↪i'   %entries)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Mean %.3f'␣
 ↪    %mean)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Std Dev %.
 ↪3f' %std)
plt.legend(fancybox=True, shadow=False, loc='best', prop={"size":12}, numpoints␣
 ↪= 1)
plt.show()
```



1.2.7 LSO1 - ^{133}Ba

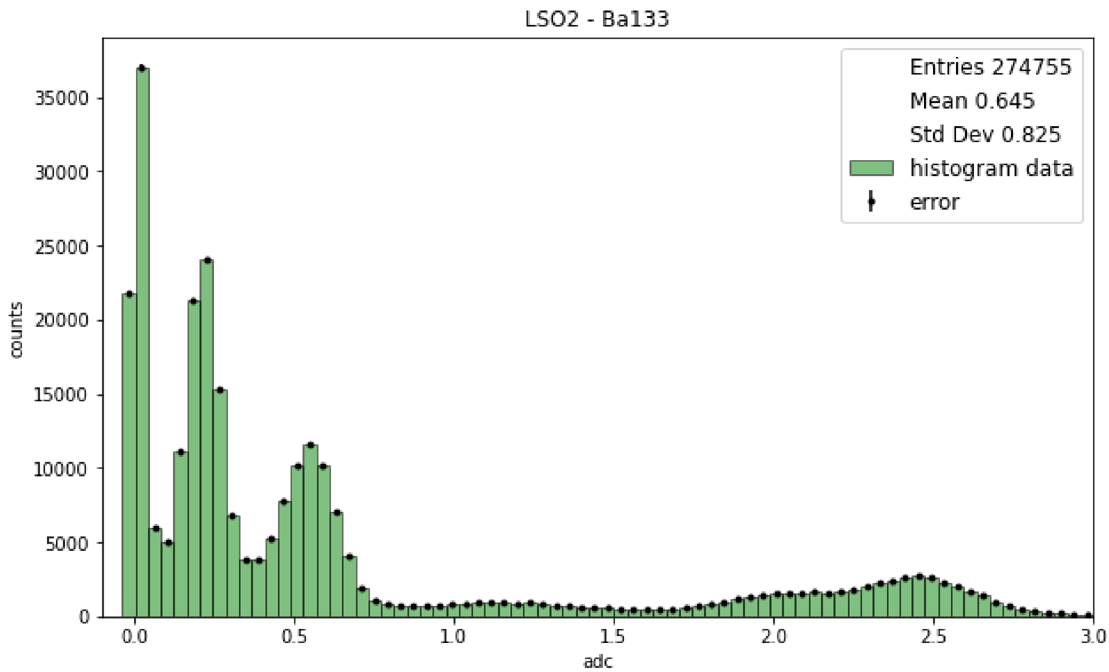
[114]:
```
# LSO1 - Ba133

# Entries, Media, Standard deviation, Errore sulla media
entries     = len(LSO1Ba133)
mean, std   = np.mean(LSO1Ba133), np.std(LSO1Ba133)
mean_error = std/np.sqrt(len(LSO1Ba133))

# Creazione dell'istogramma (con barre di errore)
fig                          = plt.figure(figsize=(10,6))
bin_heights, bin_borders, _ = plt.
 ↪hist(LSO1Ba133,bins=200,facecolor='g',ec='black',alpha=0.5,label='histogram␣
 ↪data',density=False)
bin_centers                  = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black', label='error')

# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('LSO1 - Ba133')
plt.xlim(-0.1,4)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Entries %.
 ↪i'   %entries)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Mean %.3f'␣
 ↪    %mean)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Std Dev %.
 ↪3f' %std)
plt.legend(fancybox=True, shadow=False, loc='best', prop={"size":12}, numpoints␣
 ↪= 1)
plt.show()
```



1.2.8 LSO2 - ^{133}Ba

[113]:
```
#LSO2 - Ba133

# Entries, Media, Standard deviation, Errore sulla media
entries    = len(LSO2Ba133)
mean, std  = np.mean(LSO2Ba133), np.std(LSO2Ba133)
mean_error = std/np.sqrt(len(LSO2Ba133))

# Creazione dell'istogramma (con barre di errore)
fig                            = plt.figure(figsize=(10,6))
bin_heights, bin_borders, _ = plt.
 ↪hist(LSO2Ba133,bins=200,facecolor='g',ec='black',alpha=0.5,label='histogram␣
 ↪data',density=False)
bin_centers                    = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                  = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black', label='error')

# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('LSO2 - Ba133')
plt.xlim(-0.1, 3)
```

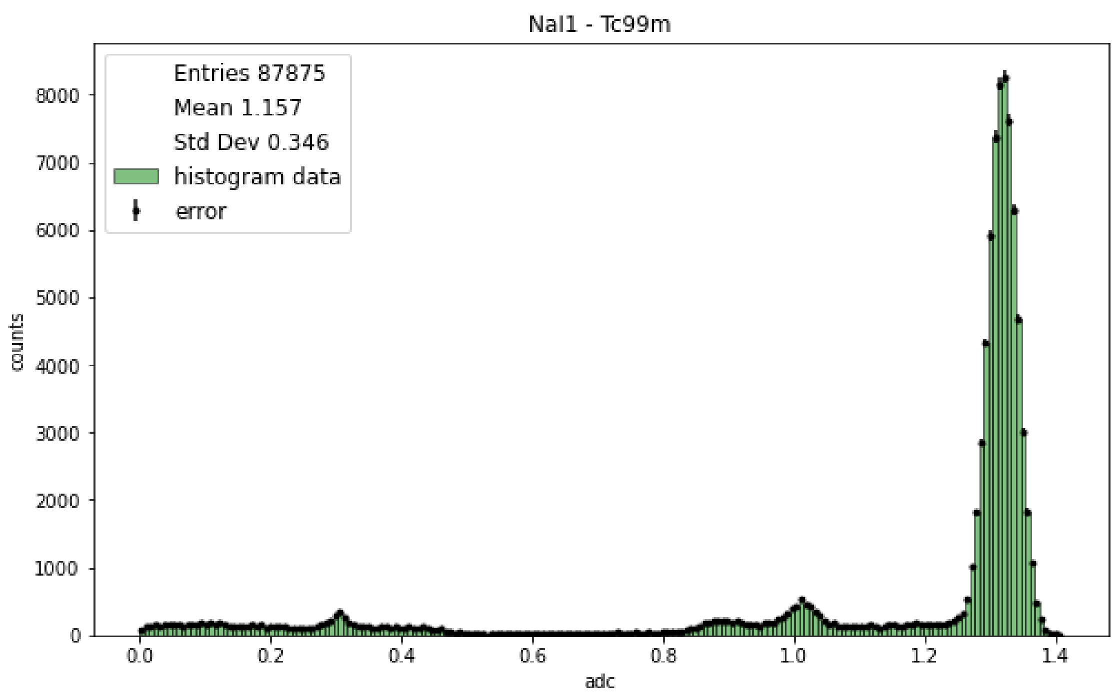
```
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Entries %.
↪i'   %entries)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Mean %.3f'␣
↪    %mean)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Std Dev %.
↪3f' %std)
plt.legend(fancybox=True, shadow=False, loc='best', prop={"size":12}, numpoints␣
↪= 1)
plt.show()
```



1.2.9 NaI1 - ^{99m}Tc

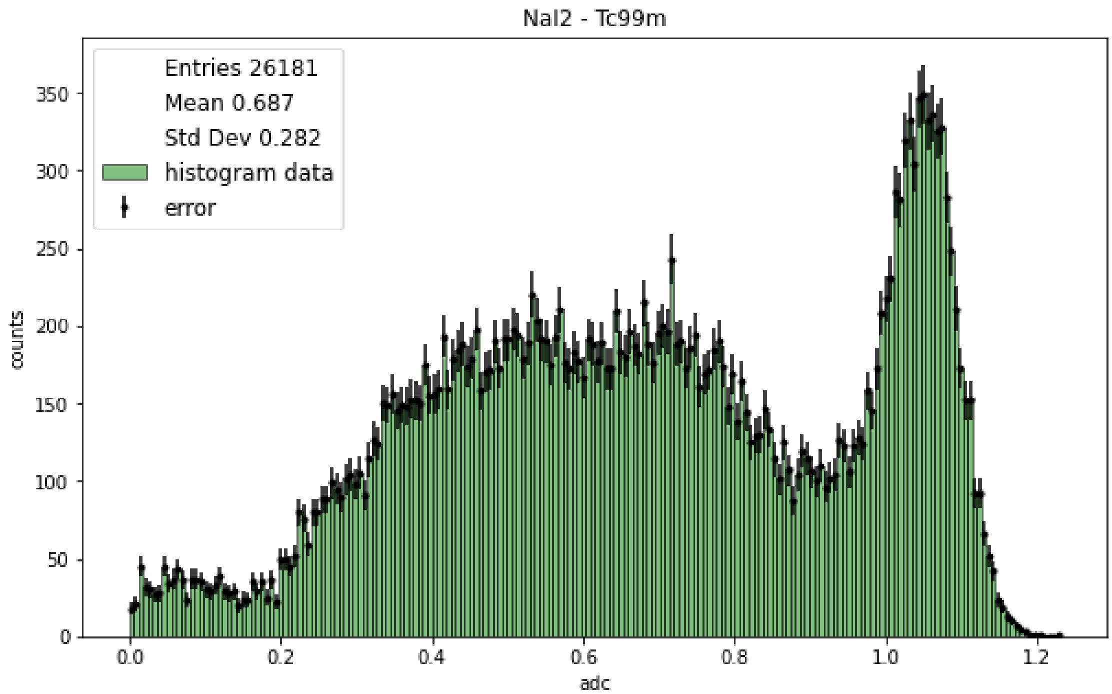
[112]:
```
# NaI1 - 99mTc

# Entries, Media, Standard deviation, Errore sulla media
entries    = len(NaI1Tc99m)
mean, std  = np.mean(NaI1Tc99m), np.std(NaI1Tc99m)
mean_error = std/np.sqrt(len(NaI1Tc99m))

# Creazione dell'istogramma (con barre di errore)
fig                            = plt.figure(figsize=(10,6))
bin_heights, bin_borders, _ = plt.
↪hist(NaI1Tc99m,bins=200,facecolor='g',ec='black',alpha=0.5,label='histogram␣
↪data',density=False)
```

```
bin_centers                   = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                 = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black', label='error')

# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('NaI1 - Tc99m')
#plt.xlim(-0.5, 10)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Entries %.
 ↪i'   %entries)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Mean %.3f'␣
 ↪    %mean)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Std Dev %.
 ↪3f' %std)
plt.legend(fancybox=True, shadow=False, loc='best', prop={"size":12}, numpoints␣
 ↪= 1)
plt.show()
```



1.2.10 NaI2 - ^{99m}Tc

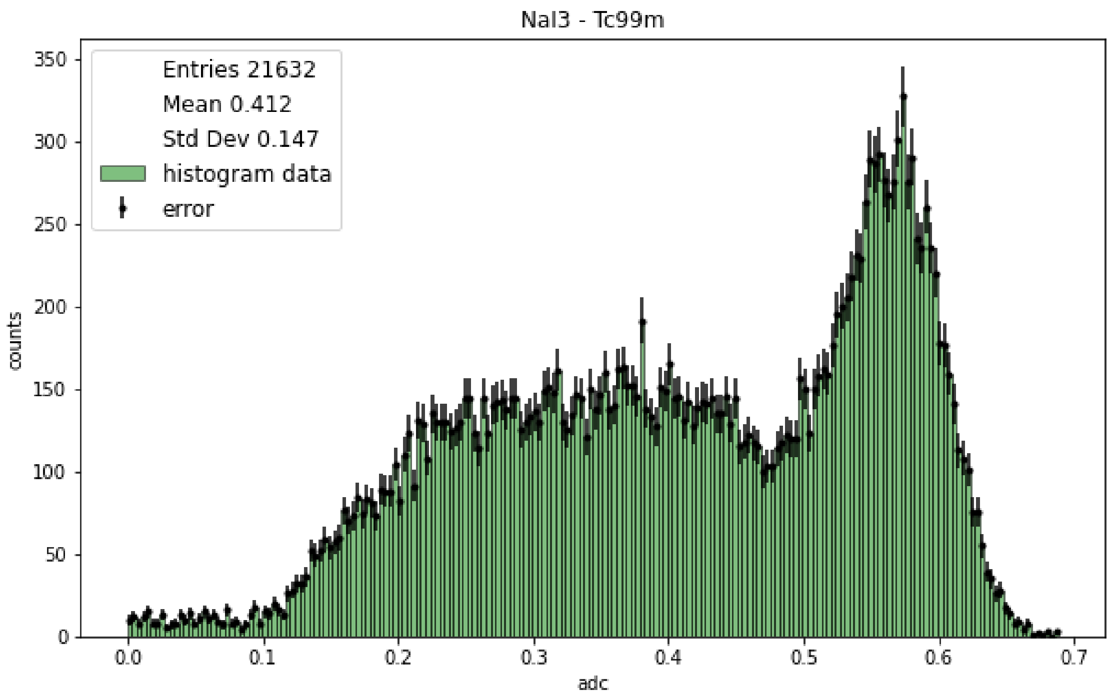
[111]:
```
# NaI2 - 99mTc

# Entries, Media, Standard deviation, Errore sulla media
entries    = len(NaI2Tc99m)
mean, std  = np.mean(NaI2Tc99m), np.std(NaI2Tc99m)
mean_error = std/np.sqrt(len(NaI2Tc99m))

# Creazione dell'istogramma (con barre di errore)
fig                           = plt.figure(figsize=(10,6))
bin_heights, bin_borders, _ = plt.
 ↪hist(NaI2Tc99m,bins=200,facecolor='g',ec='black',alpha=0.5,label='histogram␣
 ↪data',density=False)
bin_centers                   = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                 = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black', label='error')

# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('NaI2 - Tc99m')
#plt.xlim(-0.5, 10)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Entries %.
 ↪i'   %entries)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Mean %.3f'␣
 ↪    %mean)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Std Dev %.
 ↪3f' %std)
plt.legend(fancybox=True, shadow=False, loc='best', prop={"size":12}, numpoints␣
 ↪= 1)
plt.show()
```



1.2.11 NaI3 - ^{99m}Tc

[110]:
```
# NaI3 - 99mTc

# Entries, Media, Standard deviation, Errore sulla media
entries     = len(NaI3Tc99m)
mean, std   = np.mean(NaI3Tc99m), np.std(NaI3Tc99m)
mean_error = std/np.sqrt(len(NaI3Tc99m))

# Creazione dell'istogramma (con barre di errore)
fig                              = plt.figure(figsize=(10,6))
bin_heights, bin_borders, _ = plt.
 ↪hist(NaI3Tc99m,bins=200,facecolor='g',ec='black',alpha=0.5,label='histogram␣
 ↪data',density=False)
bin_centers                      = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                    = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black', label='error')

# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('NaI3 - Tc99m')
#plt.xlim(-0.5, 10)
```

```
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Entries %.
↪i'   %entries)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Mean %.3f'␣
↪    %mean)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'Std Dev %.
↪3f' %std)
plt.legend(fancybox=True, shadow=False, loc='best', prop={"size":12}, numpoints␣
↪= 1)
plt.show()
```



1.3 MODELLO: GAUSS + LINEARE

1.3.1 BGO1 - ^{18}F

[243]:
```
# BGO1 - F18

# Entries, Media, Standard deviation, Errore sulla media
entries     = len(BGO1F18)
mean, std  = np.mean(BGO1F18), np.std(BGO1F18)
mean_error = std/np.sqrt(len(BGO1F18))

# Creazione dell'istogramma (con barre di errore)
fig                            = plt.figure(1, figsize = (10,6))
```

```
bin_heights, bin_borders, _ = plt.
 ↪hist(BGO1F18,bins=200,facecolor='g',ec='black',alpha=0.5 ,density=False) #␣
 ↪label='histogram data'
bin_centers                    = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                  = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black') # label='error'


# FIT GAUSSIANO
# Definisco gli estremi di intervallo su cui eseguire il fit gaussiano
a = 2.9
b = 4.7
# Creazione dei vettori per il fit sul fotopicco => elimino la parte che non␣
 ↪interessa
bin_centers_new = []
bin_heights_new = []

for i in range(len(bin_heights)):
    if bin_centers[i] < b and bin_centers[i] > a:
        bin_centers_new.append(bin_centers[i])
        bin_heights_new.append(bin_heights[i])
    else:
        pass

bin_centers_new   = np.array(bin_centers_new)
bin_heights_new   = np.array(bin_heights_new)
sigma_heights_new = np.sqrt(bin_heights_new)

#================================================
# Funzione di fit
#================================================
def fit_gauss(x, A, mu, sigma):
    return A*np.exp(-(x-mu)**2/(2*sigma**2))

# Parametri iniziali
param0      = [entries, mean, std]
# Best Parameters
popt, pcov = curve_fit(fit_gauss, bin_centers_new, bin_heights_new, param0,␣
 ↪sigma = sigma_heights_new)

# Best-Parameters
A            = popt[0]
sigma_A      = np.sqrt(pcov[0,0])
mu           = popt[1]
sigma_mu     = np.sqrt(pcov[1,1])
sigma        = popt[2]
```

```
sigma_sigma = np.sqrt(pcov[2,2])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
x = np.linspace(min(bin_centers),max(bin_centers),500)
#x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss(x, *popt), '-', color="blue", label='Gauss')


#================================================
# CHI2 TEST
#================================================
chi2    = sum(((bin_heights_new - fit_gauss(bin_centers_new, *popt)) / sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2*np.sqrt(2*np.log(2))*sigma
sigma_FWHM = 2*np.sqrt(2*np.log(2))*sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error propagation


# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('BGO1 - F18')
plt.xlim(-0.1, 6.5)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$    (%.2f $\pm$ %.2f)'          %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
```

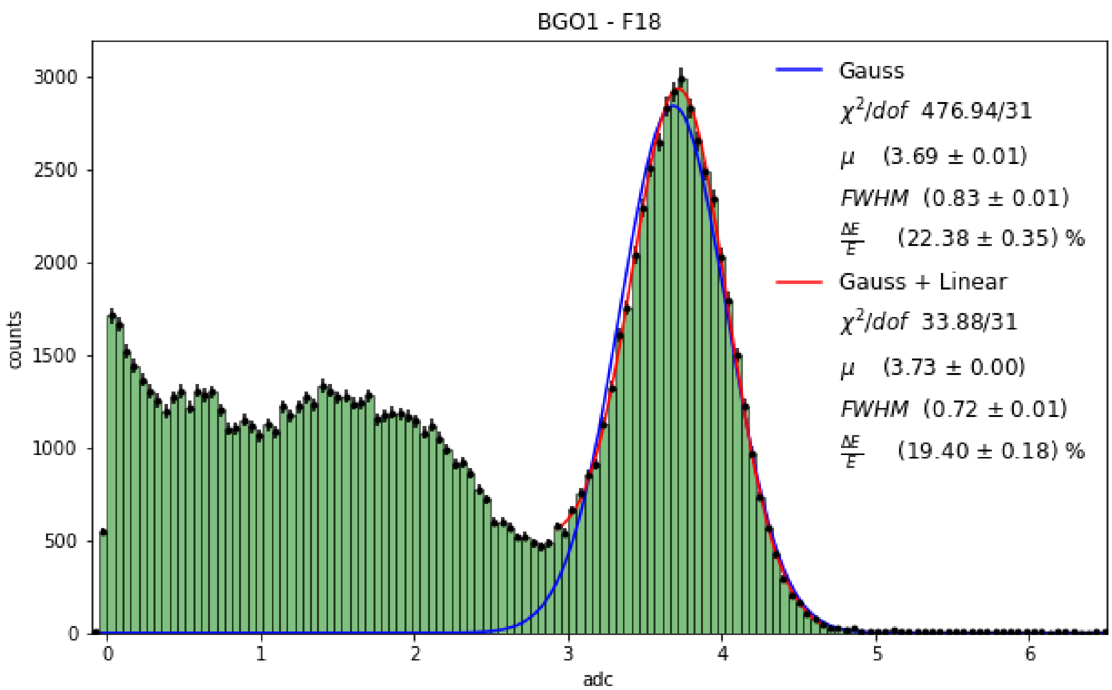
```
plt.legend(frameon=False ,fancybox=True, shadow=False, loc='best', prop={"size":
 ↪12}, numpoints = 1)

#plt.savefig('FIGURE/BGO1F18_gauss.pdf', format='pdf',bbox_inches="tight",␣
 ↪dpi=100)
#plt.show()


#==================================================
# STAMPA RISULTATI DEL FIT
#==================================================
print('==================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#--------------------------------\n')
print('# Fit Gaussiano:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('\n--------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n--------------------------------\n')
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n==================================================')

#==================================================
# Funzione di fit
#==================================================
def fit_gauss_linear(x, A, mu, sigma, a, b):
    fotopicco = A*np.exp(-(x-mu)**2/(2*sigma**2))   # Modello gaussiano per il␣
 ↪fotopicco
    fondo     = a * x + b                           # Modello lineare per il␣
 ↪fondo
    return fotopicco + fondo

# Parametri iniziali
#param0     = [entries, (a+b)/2, std]
# Best Parameters
```

```
#popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new,␣
↪param0, sigma = sigma_heights_new)
popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new,␣
↪sigma = sigma_heights_new)
# Best-Parameters
A           = popt[0]
sigma_A     = np.sqrt(pcov[0,0])
mu          = popt[1]
sigma_mu    = np.sqrt(pcov[1,1])
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])
a           = popt[3]
sigma_a     = np.sqrt(pcov[3,3])
b           = popt[4]
sigma_b     = np.sqrt(pcov[4,4])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
#x = np.linspace(min(bin_centers),max(bin_centers),500)
x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss_linear(x, *popt), '-', color="red", label='Gauss +␣
↪Linear')


#==================================================
# CHI2 TEST
#==================================================
chi2    = sum(((bin_heights_new - fit_gauss_linear(bin_centers_new, *popt)) /␣
↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2*np.sqrt(2*np.log(2))*sigma
sigma_FWHM = 2*np.sqrt(2*np.log(2))*sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error␣
↪propagation
```

```python
# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('BGO1 - F18')
plt.xlim(-0.1, 6.5)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$   (%.2f $\pm$ %.2f)'           %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
plt.legend(frameon=False, fancybox=True, shadow=False, loc='best', prop={"size":12}, numpoints = 1)

#=================================================
# STAMPA RISULTATI DEL FIT
#=================================================
print('================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#---------------------------------\n')
print('# Fit Gaussiano + Lineare:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('a        (%.3f ± %.3f)' %(popt[3],np.sqrt(pcov[3,3])))
print('b        (%.3f ± %.3f)' %(popt[4],np.sqrt(pcov[4,4])))
print('\n---------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n---------------------------------\n')
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n================================================')

plt.savefig('FIGURE/BGO1F18_gauss_gauss_linear.pdf', format='pdf',bbox_inches="tight", dpi=100)
plt.show()
```

```
==================================================

Entries   112450
Mean      2.328
Std Dev   1.374

#---------------------------------

# Fit Gaussiano:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)

# Parametri del fit:
A        (2840.809 ± 63.610)
mu       (3.686 ± 0.007)
sigma    (0.350 ± 0.005)

---------------------------------

# Chi square test:
Chi2      476.940
dof       31
Chi2/dof  15.385
pvalue    0.000

---------------------------------

FWHM      (0.825 ± 0.013)
R         (22.383 ± 0.348)%

==================================================
==================================================

Entries   112450
Mean      2.328
Std Dev   1.374

#---------------------------------

# Fit Gaussiano + Lineare:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b

# Parametri del fit:
A        (2672.169 ± 20.278)
mu       (3.728 ± 0.003)
sigma    (0.307 ± 0.003)
a        (-263.079 ± 11.188)
b        (1240.796 ± 52.326)
```

```
---------------------------------

# Chi square test:
Chi2       33.885
dof        31
Chi2/dof   1.093
pvalue     0.330

---------------------------------

FWHM       (0.723 ± 0.007)
R          (19.397 ± 0.179)%

=================================================
```



1.3.2 BGO2 - ^{18}F

[264]:
```
# BGO2 - F18

# Entries, Media, Standard deviation, Errore sulla media
entries    = len(BGO2F18)
mean, std  = np.mean(BGO1F18), np.std(BGO2F18)
mean_error = std/np.sqrt(len(BGO2F18))

# Creazione dell'istogramma (con barre di errore)
```

```
fig                             = plt.figure(1, figsize = (10,6))
bin_heights, bin_borders, _ = plt.
 ↪hist(BGO2F18,bins=200,facecolor='g',ec='black',alpha=0.5, density=False) #␣
 ↪label='histogram data'
bin_centers                     = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                   = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black') # label='error'


# FIT GAUSSIANO
# Definisco gli estremi di intervallo su cui eseguire il fit gaussiano
a = 2.29
b = 4

# Creazione dei vettori per il fit sul fotopicco => elimino la parte che non␣
 ↪interessa
bin_centers_new = []
bin_heights_new = []

for i in range(len(bin_heights)):
    if bin_centers[i] < b and bin_centers[i] > a:
        bin_centers_new.append(bin_centers[i])
        bin_heights_new.append(bin_heights[i])
    else:
        pass

bin_centers_new   = np.array(bin_centers_new)
bin_heights_new   = np.array(bin_heights_new)
sigma_heights_new = np.sqrt(bin_heights_new)

#=================================================
# Funzione di fit
#=================================================
def fit_gauss(x, A, mu, sigma):
    return A*np.exp(-(x-mu)**2/(2*sigma**2))

# Parametri iniziali
#param0     = [entries, mean, std]
# Best Parameters
#popt, pcov = curve_fit(fit_gauss, bin_centers_new, bin_heights_new, param0,␣
 ↪sigma = sigma_heights_new)
popt, pcov = curve_fit(fit_gauss, bin_centers_new, bin_heights_new, sigma =␣
 ↪sigma_heights_new)

# Best-Parameters
A            = popt[0]
```

```
sigma_A     = np.sqrt(pcov[0,0])
mu          = popt[1]
sigma_mu    = np.sqrt(pcov[1,1])
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
x = np.linspace(min(bin_centers),max(bin_centers),500)
#x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss(x, *popt), '-', color="blue", label='Gauss')


#=================================================
# CHI2 TEST
#=================================================
chi2    = sum(((bin_heights_new - fit_gauss(bin_centers_new, *popt)) /␣
 ↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2*np.sqrt(2*np.log(2))*sigma
sigma_FWHM = 2*np.sqrt(2*np.log(2))*sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error␣
 ↪propagation


# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('BGO2 - F18')
plt.xlim(1, 6)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
 ↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$   ␣
 ↪(%.2f $\pm$ %.2f)'          %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.
 ↪2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
```

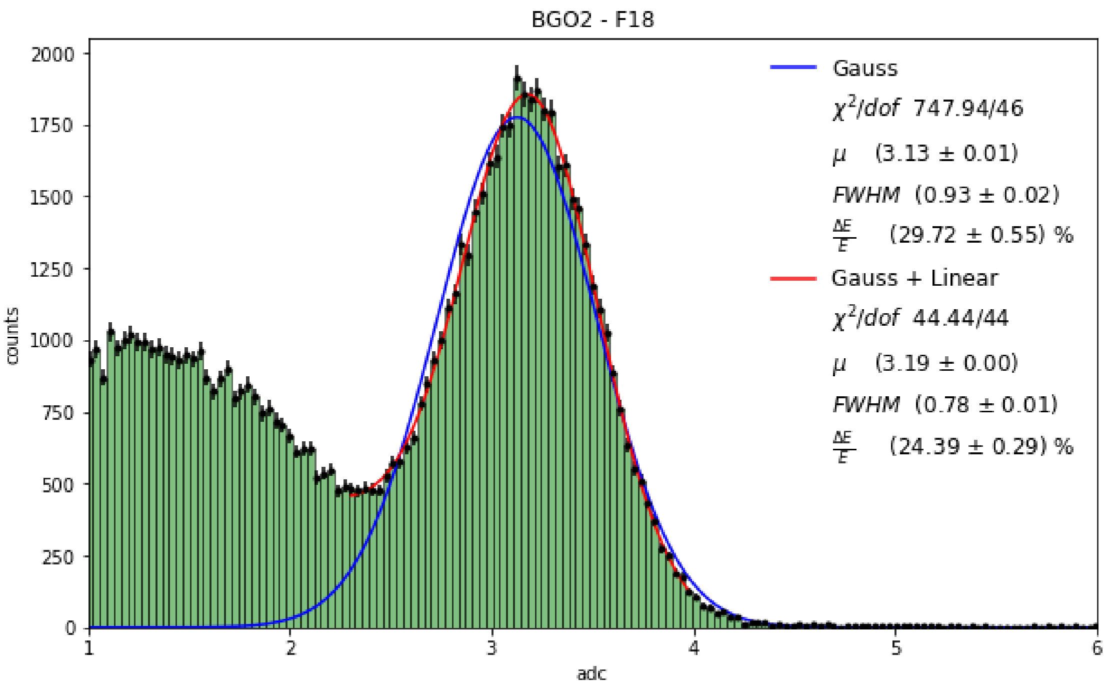
```
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
plt.legend(frameon=False, fancybox=True, shadow=False, loc='best', prop={"size":12}, numpoints = 1)

#plt.savefig('FIGURE/BGO2F18_gauss.pdf', format='pdf',bbox_inches="tight", dpi=100)
#plt.show()


#================================================
# STAMPA RISULTATI DEL FIT
#================================================
print('================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#------------------------------\n')
print('# Fit Gaussiano:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('\n-------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n-------------------------------\n')
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n================================================')


#================================================
# Funzione di fit
#================================================
def fit_gauss_linear(x, A, mu, sigma, a, b):
    fotopicco = A*np.exp(-(x-mu)**2/(2*sigma**2))   # Modello gaussiano per il fotopicco
    fondo     = a * x + b                           # Modello lineare per il fondo
    return fotopicco + fondo
```

```
# Parametri iniziali
param0     = [entries, (a+b)/2, std,1,1]
# Best Parameters
popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new,␣
↪param0, sigma = sigma_heights_new)
#popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new,␣
↪sigma = sigma_heights_new)
# Best-Parameters
A           = popt[0]
sigma_A     = np.sqrt(pcov[0,0])
mu          = popt[1]
sigma_mu    = np.sqrt(pcov[1,1])
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])
a           = popt[3]
sigma_a     = np.sqrt(pcov[3,3])
b           = popt[4]
sigma_b     = np.sqrt(pcov[4,4])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
#x = np.linspace(min(bin_centers),max(bin_centers),500)
x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss_linear(x, *popt), '-', color="red", label='Gauss +␣
↪Linear')


#================================================
# CHI2 TEST
#================================================
chi2    = sum(((bin_heights_new - fit_gauss_linear(bin_centers_new, *popt)) /␣
↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM        = 2*np.sqrt(2*np.log(2))*sigma
sigma_FWHM = 2*np.sqrt(2*np.log(2))*sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
```

```
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error␣
↪propagation


# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('BGO2 - F18')
plt.xlim(1, 6)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$    ␣
↪(%.2f $\pm$ %.2f)'          %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.
↪2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
↪label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
plt.legend(frameon=False, fancybox=True, shadow=False, loc='best', prop={"size":
↪12}, numpoints = 1)

#================================================
# STAMPA RISULTATI DEL FIT
#================================================
print('================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#---------------------------------\n')
print('# Fit Gaussiano + Lineare:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('a        (%.3f ± %.3f)' %(popt[3],np.sqrt(pcov[3,3])))
print('b        (%.3f ± %.3f)' %(popt[4],np.sqrt(pcov[4,4])))
print('\n---------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n---------------------------------\n')
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n================================================')
```

```
==================================================

Entries   111969
Mean      2.328
Std Dev   1.186

#---------------------------------

# Fit Gaussiano:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)

# Parametri del fit:
A        (1775.170 ± 41.656)
mu       (3.127 ± 0.008)
sigma    (0.395 ± 0.007)

----------------------------------

# Chi square test:
Chi2      747.938
dof       46
Chi2/dof  16.260
pvalue    0.000

----------------------------------

FWHM      (0.929 ± 0.017)
R         (29.717 ± 0.551)%

==================================================
==================================================

Entries   111969
Mean      2.328
Std Dev   1.186

#---------------------------------

# Fit Gaussiano + Lineare:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b

# Parametri del fit:
A        (1642.721 ± 12.910)
mu       (3.193 ± 0.003)
sigma    (0.331 ± 0.004)
a        (-229.146 ± 8.201)
b        (942.138 ± 28.518)
```

```
---------------------------------

# Chi square test:
Chi2       44.439
dof        44
Chi2/dof   1.010
pvalue     0.453

---------------------------------

FWHM       (0.779 ± 0.009)
R          (24.393 ± 0.288)%

=================================================
```



1.3.3 BGO3 - ^{18}F

[266]:

```
# BGO3 - F18

# Entries, Media, Standard deviation, Errore sulla media
entries    = len(BGO3F18)
mean, std  = np.mean(BGO3F18), np.std(BGO3F18)
mean_error = std/np.sqrt(len(BGO3F18))

# Creazione dell'istogramma (con barre di errore)
```

```python
fig                                = plt.figure(1, figsize = (10,6))
bin_heights, bin_borders, _ = plt.
 ↪hist(BGO3F18,bins=200,facecolor='g',ec='black',alpha=0.5, density=False) #␣
 ↪label='histogram data'
bin_centers                        = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                      = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black') # label='error'


# FIT GAUSSIANO
# Definisco gli estremi di intervallo su cui eseguire il fit gaussiano
a = 3.5
b = 6.4

# Creazione dei vettori per il fit sul fotopicco => elimino la parte che non␣
 ↪interessa
bin_centers_new = []
bin_heights_new = []

for i in range(len(bin_heights)):
    if bin_centers[i] < b and bin_centers[i] > a:
        bin_centers_new.append(bin_centers[i])
        bin_heights_new.append(bin_heights[i])
    else:
        pass

bin_centers_new    = np.array(bin_centers_new)
bin_heights_new    = np.array(bin_heights_new)
sigma_heights_new = np.sqrt(bin_heights_new)

#=================================================
# Funzione di fit
#=================================================
def fit_gauss(x, A, mu, sigma):
    return A*np.exp(-(x-mu)**2/(2*sigma**2))

# Parametri iniziali
param0     = [entries, mean, std]
# Best Parameters
popt, pcov = curve_fit(fit_gauss, bin_centers_new, bin_heights_new, param0,␣
 ↪sigma = sigma_heights_new)

# Best-Parameters
A            = popt[0]
sigma_A      = np.sqrt(pcov[0,0])
mu           = popt[1]
```

```python
sigma_mu    = np.sqrt(pcov[1,1])
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
x = np.linspace(min(bin_centers),max(bin_centers),500)
#x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss(x, *popt), '-', color="blue", label='Gauss + Linear')


#================================================
# CHI2 TEST
#================================================
chi2    = sum(((bin_heights_new - fit_gauss(bin_centers_new, *popt)) /␣
 ↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2*np.sqrt(2*np.log(2))*abs(sigma)
sigma_FWHM = 2*np.sqrt(2*np.log(2))*sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error␣
 ↪propagation



# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('BGO3 - F18')
plt.xlim(2, 9)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
 ↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$   ␣
 ↪(%.2f $\pm$ %.2f)'          %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.
 ↪2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
 ↪label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
```

```python
plt.legend(frameon=False, fancybox=True, shadow=False, loc='best', prop={"size":12}, numpoints = 1)


#===============================================
# STAMPA RISULTATI DEL FIT
#===============================================
print('=================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#--------------------------------\n')
print('# Fit Gaussiano:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('\n--------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n--------------------------------\n')
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n===============================================')


#===============================================
# Funzione di fit
#===============================================
def fit_gauss_linear(x, A, mu, sigma, a, b):
    fotopicco = A*np.exp(-(x-mu)**2/(2*sigma**2))   # Modello gaussiano per il fotopicco
    fondo     = a * x + b                           # Modello lineare per il fondo
    return fotopicco + fondo

# Parametri iniziali
#param0     = [entries, (a+b)/2, std]
# Best Parameters
#popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new, param0, sigma = sigma_heights_new)
popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new, sigma = sigma_heights_new)
```

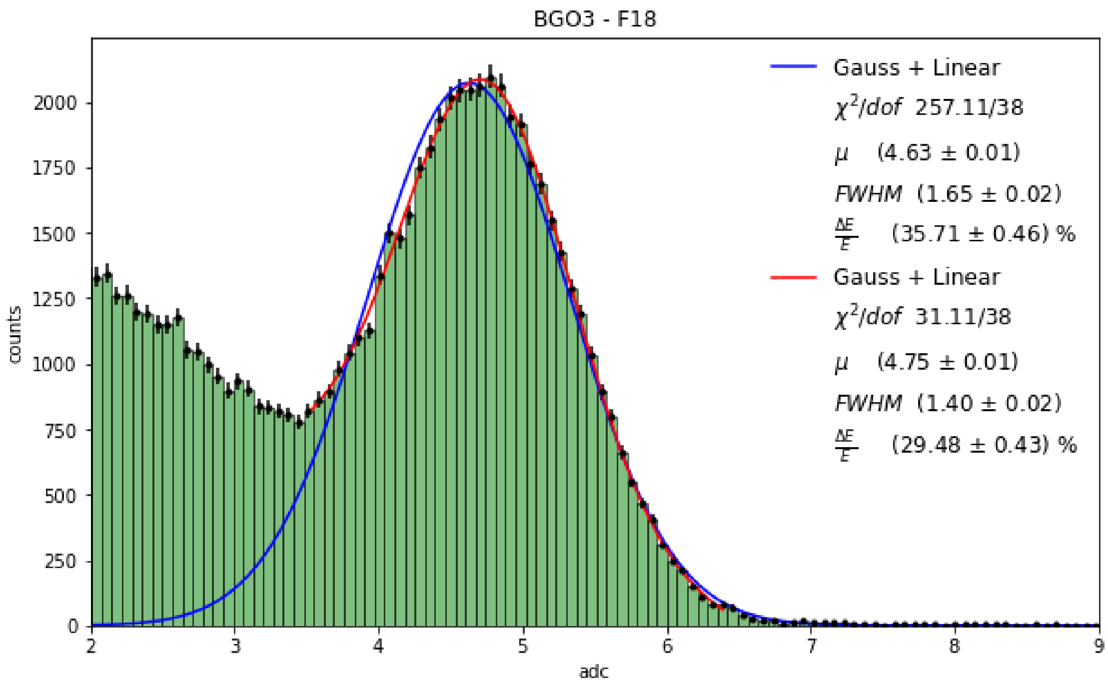
```
# Best-Parameters
A           = popt[0]
sigma_A     = np.sqrt(pcov[0,0])
mu          = popt[1]
sigma_mu    = np.sqrt(pcov[1,1])
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])
a           = popt[3]
sigma_a     = np.sqrt(pcov[3,3])
b           = popt[4]
sigma_b     = np.sqrt(pcov[4,4])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
#x = np.linspace(min(bin_centers),max(bin_centers),500)
x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss_linear(x, *popt), '-', color="red", label='Gauss +␣
↪Linear')



#================================================
# CHI2 TEST
#================================================
chi2    = sum(((bin_heights_new - fit_gauss_linear(bin_centers_new, *popt)) /␣
↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2 * np.sqrt(2 * np.log(2)) * abs(sigma)
sigma_FWHM = 2 * np.sqrt(2 * np.log(2)) * sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error␣
↪propagation



# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('BGO3 - F18')
```

```
plt.xlim(2, 9)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$   ␣
↪(%.2f $\pm$ %.2f)'           %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.
↪2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
↪label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
plt.legend(frameon=False, fancybox=True, shadow=False,loc='best', prop={"size":
↪12}, numpoints = 1)

#==================================================
# STAMPA RISULTATI DEL FIT
#==================================================
print('==================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#--------------------------------\n')
print('# Fit Gaussiano + Lineare:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('a        (%.3f ± %.3f)' %(popt[3],np.sqrt(pcov[3,3])))
print('b        (%.3f ± %.3f)' %(popt[4],np.sqrt(pcov[4,4])))
print('\n--------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n--------------------------------\n')
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n==================================================')


plt.savefig('FIGURE/BGO3F18_gauss.pdf', format='pdf',bbox_inches="tight",␣
↪dpi=100)
plt.show()
```

```
==================================================

Entries   111912
```

```
Mean      2.954
Std Dev   1.792

#--------------------------------

# Fit Gaussiano:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)

# Parametri del fit:
A        (2071.617 ± 30.764)
mu       (4.627 ± 0.010)
sigma    (-0.702 ± 0.009)

---------------------------------

# Chi square test:
Chi2      257.107
dof       38
Chi2/dof  6.766
pvalue    0.000

---------------------------------

FWHM      (1.653 ± 0.021)
R         (35.714 ± 0.455)%

=================================================
=================================================

Entries   111912
Mean      2.954
Std Dev   1.792

#--------------------------------

# Fit Gaussiano + Lineare:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b

# Parametri del fit:
A        (1720.438 ± 19.843)
mu       (4.749 ± 0.007)
sigma    (-0.594 ± 0.009)
a        (-206.392 ± 9.757)
b        (1341.280 ± 64.890)

---------------------------------

# Chi square test:
```

```
Chi2        31.112
dof         38
Chi2/dof    0.819
pvalue      0.778

---------------------------------

FWHM        (1.400 ± 0.020)
R           (29.477 ± 0.426)%

=================================================
```



1.3.4 LSO1 - ^{18}F

[267]:
```python
# LSO1 - F18

# Entries, Media, Standard deviation, Errore sulla media
entries    = len(LSO1F18)
mean, std  = np.mean(BGO3F18), np.std(LSO1F18)
mean_error = std/np.sqrt(len(LSO1F18))

# Creazione dell'istogramma (con barre di errore)
fig                          = plt.figure(1, figsize = (10,6))
```

```
bin_heights, bin_borders, _ = plt.
 ↪hist(LS01F18,bins=200,facecolor='g',ec='black',alpha=0.5,density=False) #␣
 ↪label='histogram data'
bin_centers                 = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights               = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black') # label='error'



# FIT GAUSSIANO
# Definisco gli estremi di intervallo su cui eseguire il fit gaussiano
a = 2.5
b = 3.1

# Creazione dei vettori per il fit sul fotopicco => elimino la parte che non␣
 ↪interessa
bin_centers_new = []
bin_heights_new = []

for i in range(len(bin_heights)):
    if bin_centers[i] < b and bin_centers[i] > a:
        bin_centers_new.append(bin_centers[i])
        bin_heights_new.append(bin_heights[i])
    else:
        pass

bin_centers_new   = np.array(bin_centers_new)
bin_heights_new   = np.array(bin_heights_new)
sigma_heights_new = np.sqrt(bin_heights_new)

#===============================================
# Funzione di fit
#===============================================
def fit_gauss(x, A, mu, sigma):
    return A*np.exp(-(x-mu)**2/(2*sigma**2))

# Parametri iniziali
param0     = [entries, mean, std]
# Best Parameters
popt, pcov = curve_fit(fit_gauss, bin_centers_new, bin_heights_new, param0,␣
 ↪sigma = sigma_heights_new)

# Best-Parameters
A          = popt[0]
sigma_A    = np.sqrt(pcov[0,0])
mu         = popt[1]
sigma_mu   = np.sqrt(pcov[1,1])
```

```
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
x = np.linspace(min(bin_centers),max(bin_centers),500)
#x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss(x, *popt), '-', color="blue", label='Gauss')


#=================================================
# CHI2 TEST
#=================================================
chi2    = sum(((bin_heights_new - fit_gauss(bin_centers_new, *popt)) /␣
↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2*np.sqrt(2*np.log(2))*abs(sigma)
sigma_FWHM = 2*np.sqrt(2*np.log(2))*sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error␣
↪propagation



# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('LSO1 - F18')
plt.xlim(-0.1, 4)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$    ␣
↪(%.2f $\pm$ %.2f)'          %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.
↪2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
↪label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
```

```python
plt.legend(frameon=False,fancybox=True, shadow=False, loc='best', prop={"size":
↪12}, numpoints = 1)


#=================================================
# STAMPA RISULTATI DEL FIT
#=================================================
print('================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#------------------------------\n')
print('# Fit Gaussiano:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('\n-------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n-------------------------------\n')
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n================================================')


#=================================================
# Funzione di fit
#=================================================
def fit_gauss_linear(x, A, mu, sigma, a, b):
    fotopicco = A*np.exp(-(x-mu)**2/(2*sigma**2))   # Modello gaussiano per il␣
↪fotopicco
    fondo     = a * x + b                           # Modello lineare per il␣
↪fondo
    return fotopicco + fondo

# Parametri iniziali
#param0     = [entries, (a+b)/2, std]
# Best Parameters
#popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new,␣
↪param0, sigma = sigma_heights_new)
popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new,␣
↪sigma = sigma_heights_new)
```

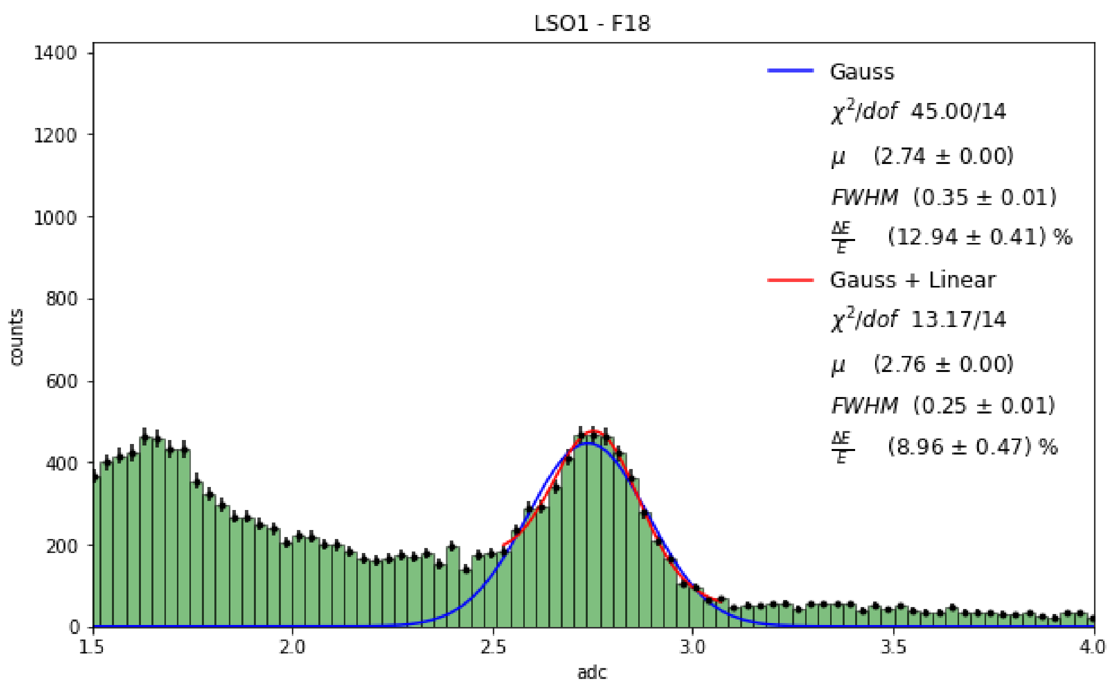
```python
# Best-Parameters
A           = popt[0]
sigma_A     = np.sqrt(pcov[0,0])
mu          = popt[1]
sigma_mu    = np.sqrt(pcov[1,1])
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])
a           = popt[3]
sigma_a     = np.sqrt(pcov[3,3])
b           = popt[4]
sigma_b     = np.sqrt(pcov[4,4])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
#x = np.linspace(min(bin_centers),max(bin_centers),500)
x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss_linear(x, *popt), '-', color="red", label='Gauss +␣
↪Linear')



#================================================
# CHI2 TEST
#================================================
chi2    = sum(((bin_heights_new - fit_gauss_linear(bin_centers_new, *popt)) /␣
↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2 * np.sqrt(2 * np.log(2)) * abs(sigma)
sigma_FWHM = 2 * np.sqrt(2 * np.log(2)) * sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error␣
↪propagation



# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('LSO1 - F18')
```

```python
plt.xlim(1.5, 4.0)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$    (%.2f $\pm$ %.2f)'           %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
plt.legend(frameon=False, fancybox=True, shadow=False, loc='best', prop={"size":12}, numpoints = 1)

#=================================================
# STAMPA RISULTATI DEL FIT
#=================================================
print('=================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#---------------------------------\n')
print('# Fit Gaussiano + Lineare:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('a        (%.3f ± %.3f)' %(popt[3],np.sqrt(pcov[3,3])))
print('b        (%.3f ± %.3f)' %(popt[4],np.sqrt(pcov[4,4])))
print('\n---------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n---------------------------------\n')
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n=================================================')


plt.savefig('FIGURE/LS01F18_gauss.pdf', format='pdf',bbox_inches="tight", dpi=100)
plt.show()
```

```
=================================================

Entries   52685
```

```
Mean      2.954
Std Dev   0.905

#--------------------------------

# Fit Gaussiano:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)

# Parametri del fit:
A        (446.756 ± 14.699)
mu       (2.739 ± 0.005)
sigma    (-0.151 ± 0.005)

---------------------------------

# Chi square test:
Chi2      45.002
dof       14
Chi2/dof  3.214
pvalue    0.000

---------------------------------

FWHM      (0.354 ± 0.011)
R         (12.941 ± 0.409)%

=================================================
=================================================

Entries   52685
Mean      2.954
Std Dev   0.905

#--------------------------------

# Fit Gaussiano + Lineare:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b

# Parametri del fit:
A        (355.701 ± 13.747)
mu       (2.759 ± 0.004)
sigma    (-0.105 ± 0.005)
a        (-207.004 ± 33.914)
b        (691.008 ± 104.196)

---------------------------------

# Chi square test:
```

```
Chi2       13.174
dof        14
Chi2/dof   0.941
pvalue     0.513

--------------------------------

FWHM       (0.247 ± 0.013)
R          (8.955 ± 0.469)%

================================================
```



1.3.5 LSO2 - ^{18}F

[268]:
```
# LSO2 - F18

# Entries, Media, Standard deviation, Errore sulla media
entries    = len(LSO2F18)
mean, std  = np.mean(LSO2F18), np.std(LSO2F18)
mean_error = std/np.sqrt(len(LSO2F18))

# Creazione dell'istogramma (con barre di errore)
fig                          = plt.figure(1, figsize = (10,6))
```

```
bin_heights, bin_borders, _ = plt.
 ↪hist(LS02F18,bins=200,facecolor='g',ec='black',alpha=0.5,density=False) #␣
 ↪label='histogram data'
bin_centers                  = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black') # label='error'


# FIT GAUSSIANO
# Definisco gli estremi di intervallo su cui eseguire il fit gaussiano
a = 3.5
b = 4.6

# Creazione dei vettori per il fit sul fotopicco => elimino la parte che non␣
 ↪interessa
bin_centers_new = []
bin_heights_new = []

for i in range(len(bin_heights)):
    if bin_centers[i] < b and bin_centers[i] > a:
        bin_centers_new.append(bin_centers[i])
        bin_heights_new.append(bin_heights[i])
    else:
        pass

bin_centers_new   = np.array(bin_centers_new)
bin_heights_new   = np.array(bin_heights_new)
sigma_heights_new = np.sqrt(bin_heights_new)

#==================================================
# Funzione di fit
#==================================================
def fit_gauss(x, A, mu, sigma):
    return A*np.exp(-(x-mu)**2/(2*sigma**2))

# Parametri iniziali
param0     = [entries, mean, std]
# Best Parameters
popt, pcov = curve_fit(fit_gauss, bin_centers_new, bin_heights_new, param0,␣
 ↪sigma = sigma_heights_new)

# Best-Parameters
A          = popt[0]
sigma_A    = np.sqrt(pcov[0,0])
mu         = popt[1]
sigma_mu   = np.sqrt(pcov[1,1])
```

```python
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
x = np.linspace(min(bin_centers),max(bin_centers),500)
#x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss(x, *popt), '-', color="blue", label='Gauss')


#==================================================
# CHI2 TEST
#==================================================
chi2    = sum(((bin_heights_new - fit_gauss(bin_centers_new, *popt)) /␣
 ↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2*np.sqrt(2*np.log(2))*abs(sigma)
sigma_FWHM = 2*np.sqrt(2*np.log(2))*sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error␣
 ↪propagation


# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('LSO2 - F18')
plt.xlim(-0.1, 6)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
 ↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$   ␣
 ↪(%.2f $\pm$ %.2f)'          %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.
 ↪2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
 ↪label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
```

```
plt.legend(frameon=False, fancybox=True, shadow=False, loc='best', prop={"size":
 ↪12}, numpoints = 1)

#================================================
# STAMPA RISULTATI DEL FIT
#================================================
print('=====================================================\n')
print('Entries    %.i'    %entries)
print('Mean       %.3f'   %mean)
print('Std Dev    %.3f'   %std)
print('\n#--------------------------------\n')
print('# Fit Gaussiano:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('\n--------------------------------\n')
print('# Chi square test:')
print('Chi2       %.3f' %chi2)
print('dof        %.i'  %dof)
print('Chi2/dof   %.3f' %chi2_rid)
print('pvalue     %.3f' %pvalue)
print('\n--------------------------------\n')
print('FWHM       (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R          (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n=================================================')


#================================================
# Funzione di fit
#================================================
def fit_gauss_linear(x, A, mu, sigma, a, b):
    fotopicco = A*np.exp(-(x-mu)**2/(2*sigma**2))   # Modello gaussiano per il␣
 ↪fotopicco
    fondo     = a * x + b                           # Modello lineare per il␣
 ↪fondo
    return fotopicco + fondo

# Parametri iniziali
param0     = [entries, (a+b)/2, std, 1,1]
# Best Parameters
popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new,␣
 ↪param0, sigma = sigma_heights_new)
#popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new,␣
 ↪sigma = sigma_heights_new)
# Best-Parameters
```

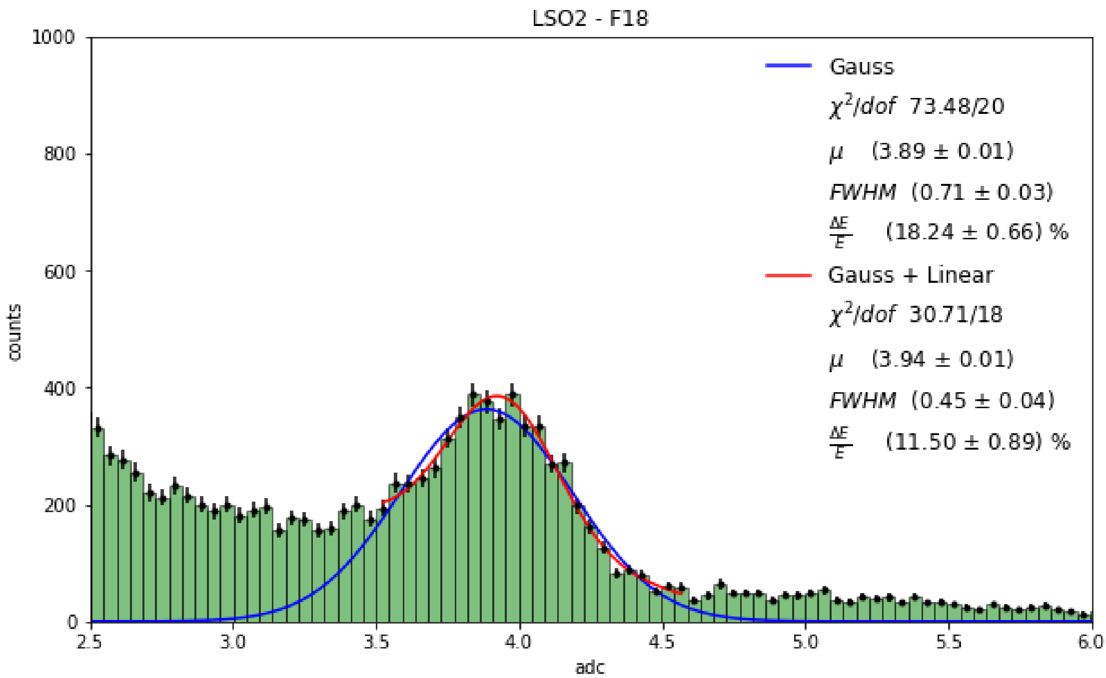
```
A           = popt[0]
sigma_A     = np.sqrt(pcov[0,0])
mu          = popt[1]
sigma_mu    = np.sqrt(pcov[1,1])
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])
a           = popt[3]
sigma_      = np.sqrt(pcov[3,3])
b           = popt[4]
sigma_b     = np.sqrt(pcov[4,4])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
#x = np.linspace(min(bin_centers),max(bin_centers),500)
x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss_linear(x, *popt), '-', color="red", label='Gauss +␣
 ↪Linear')



#================================================
# CHI2 TEST
#================================================
chi2    = sum(((bin_heights_new - fit_gauss_linear(bin_centers_new, *popt)) /␣
 ↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2 * np.sqrt(2 * np.log(2)) * abs(sigma)
sigma_FWHM = 2 * np.sqrt(2 * np.log(2)) * sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error␣
 ↪propagation



# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('LS02 - F18')
plt.xlim(2.5, 6)
```

```
plt.ylim(0,1000)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$   (%.2f $\pm$ %.2f)'           %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
plt.legend(frameon=False, fancybox=True, shadow=False, loc='best', prop={"size":12}, numpoints = 1)

#==================================================
# STAMPA RISULTATI DEL FIT
#==================================================
print('=================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#--------------------------------\n')
print('# Fit Gaussiano + Lineare:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('a        (%.3f ± %.3f)' %(popt[3],np.sqrt(pcov[3,3])))
print('b        (%.3f ± %.3f)' %(popt[4],np.sqrt(pcov[4,4])))
print('\n--------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n--------------------------------\n')
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n================================================')



plt.savefig('FIGURE/LSO2F18_gauss.pdf', format='pdf',bbox_inches="tight", dpi=100)
plt.show()
```

===

```
Entries   52969
Mean      1.676
Std Dev   1.294

#--------------------------------

# Fit Gaussiano:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)

# Parametri del fit:
A        (362.457 ± 12.132)
mu       (3.886 ± 0.011)
sigma    (0.301 ± 0.011)

---------------------------------

# Chi square test:
Chi2      73.477
dof       20
Chi2/dof  3.674
pvalue    0.000

---------------------------------

FWHM      (0.709 ± 0.026)
R         (18.243 ± 0.662)%

=================================================
=================================================

Entries   52969
Mean      1.676
Std Dev   1.294

#--------------------------------

# Fit Gaussiano + Lineare:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b

# Parametri del fit:
A        (258.654 ± 15.713)
mu       (3.940 ± 0.012)
sigma    (0.192 ± 0.015)
a        (-127.147 ± 22.448)
b        (626.469 ± 102.221)

---------------------------------
```

```
# Chi square test:
Chi2      30.712
dof       18
Chi2/dof  1.706
pvalue    0.031

---------------------------------

FWHM      (0.453 ± 0.035)
R         (11.495 ± 0.890)%

=================================================
```



1.3.6 LSO3 - ^{18}F

[269]:
```
# LSO3 - F18

# Entries, Media, Standard deviation, Errore sulla media
entries     = len(LSO3F18)
mean, std   = np.mean(LSO3F18), np.std(LSO3F18)
mean_error = std/np.sqrt(len(LSO3F18))

# Creazione dell'istogramma (con barre di errore)
fig                            = plt.figure(1, figsize = (10,6))
```

```
bin_heights, bin_borders, _ = plt.
 ↪hist(LS03F18,bins=200,facecolor='g',ec='black',alpha=0.5,density=False) #␣
 ↪label='histogram data'
bin_centers                  = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black') # label='error'


# FIT GAUSSIANO
# Definisco gli estremi di intervallo su cui eseguire il fit gaussiano
a = 2.6
b = 6.4

# Creazione dei vettori per il fit sul fotopicco => elimino la parte che non␣
 ↪interessa
bin_centers_new = []
bin_heights_new = []

for i in range(len(bin_heights)):
    if bin_centers[i] < b and bin_centers[i] > a:
        bin_centers_new.append(bin_centers[i])
        bin_heights_new.append(bin_heights[i])
    else:
        pass

bin_centers_new   = np.array(bin_centers_new)
bin_heights_new   = np.array(bin_heights_new)
sigma_heights_new = np.sqrt(bin_heights_new)

#================================================
# Funzione di fit
#================================================
def fit_gauss(x, A, mu, sigma):
    return A*np.exp(-(x-mu)**2/(2*sigma**2))

# Parametri iniziali
param0     = [entries, mean, std]
# Best Parameters
popt, pcov = curve_fit(fit_gauss, bin_centers_new, bin_heights_new, param0,␣
 ↪sigma = sigma_heights_new)

# Best-Parameters
A           = popt[0]
sigma_A     = np.sqrt(pcov[0,0])
mu          = popt[1]
sigma_mu    = np.sqrt(pcov[1,1])
```

```
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
x = np.linspace(min(bin_centers),max(bin_centers),500)
#x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss(x, *popt), '-', color="blue", label='Gauss')


#================================================
# CHI2 TEST
#================================================
chi2    = sum(((bin_heights_new - fit_gauss(bin_centers_new, *popt)) /
 ↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2*np.sqrt(2*np.log(2))*sigma
sigma_FWHM = 2*np.sqrt(2*np.log(2))*sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error
 ↪propagation


# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('LSO3 - F18')
plt.xlim(-0.1, 6)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
 ↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$
 ↪(%.2f $\pm$ %.2f)'          %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.
 ↪2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None',
 ↪label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
```

```
plt.legend(frameon=False,fancybox=True, shadow=False, loc='best', prop={"size":
↪12}, numpoints = 1)


#=================================================
# STAMPA RISULTATI DEL FIT
#=================================================
print('=================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#-------------------------------\n')
print('# Fit Gaussiano:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('\n-------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n-------------------------------\n')
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n=================================================')


#=================================================
# Funzione di fit
#=================================================
def fit_gauss_linear(x, A, mu, sigma, a, b):
    fotopicco = A*np.exp(-(x-mu)**2/(2*sigma**2))   # Modello gaussiano per il␣
↪fotopicco
    fondo     = a * x + b                           # Modello lineare per il␣
↪fondo
    return fotopicco + fondo

# Parametri iniziali
#param0     = [entries, (a+b)/2, std]
# Best Parameters
#popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new,␣
↪param0, sigma = sigma_heights_new)
popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new,␣
↪sigma = sigma_heights_new)
```

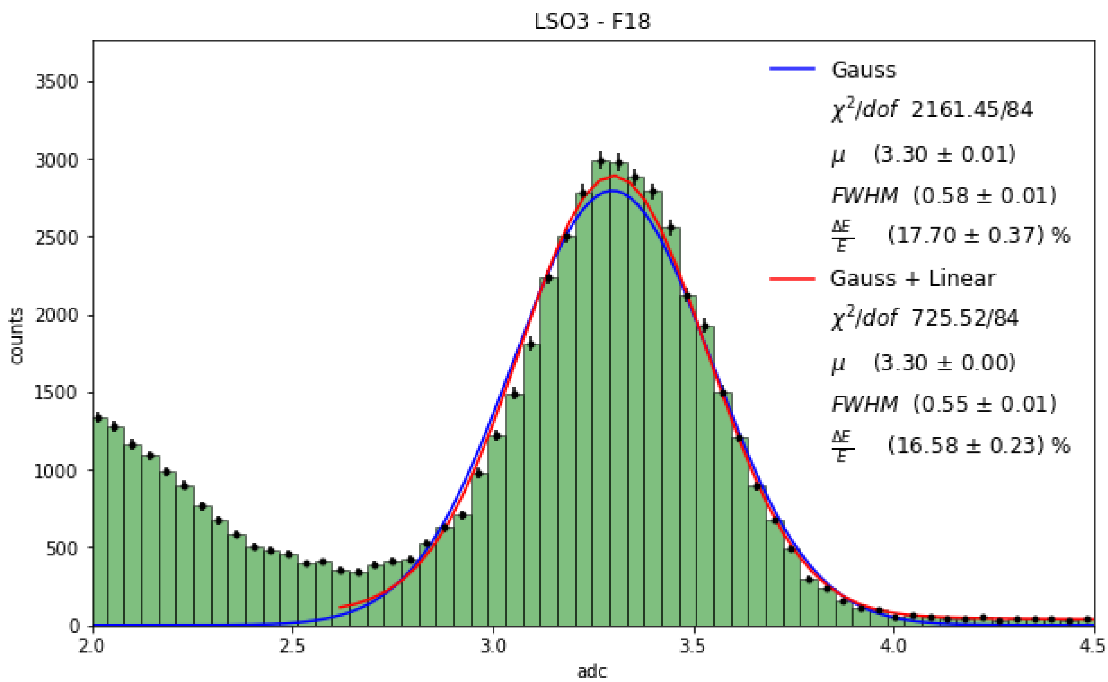
```
# Best-Parameters
A           = popt[0]
sigma_A     = np.sqrt(pcov[0,0])
mu          = popt[1]
sigma_mu    = np.sqrt(pcov[1,1])
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])
a           = popt[3]
sigma_a     = np.sqrt(pcov[3,3])
b           = popt[4]
sigma_b     = np.sqrt(pcov[4,4])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
#x = np.linspace(min(bin_centers),max(bin_centers),500)
x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss_linear(x, *popt), '-', color="red", label='Gauss + Linear')



#================================================
# CHI2 TEST
#================================================
chi2    = sum(((bin_heights_new - fit_gauss_linear(bin_centers_new, *popt)) / sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2 * np.sqrt(2 * np.log(2)) * abs(sigma)
sigma_FWHM = 2 * np.sqrt(2 * np.log(2)) * sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error propagation



# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('LSO3 - F18')
```

```python
plt.xlim(2, 4.5)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$   ␣
↪(%.2f $\pm$ %.2f)'           %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.
↪2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
↪label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
plt.legend(frameon=False, fancybox=True, shadow=False, loc='best', prop={"size":
↪12}, numpoints = 1)

#==================================================
# STAMPA RISULTATI DEL FIT
#==================================================
print('==================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#--------------------------------\n')
print('# Fit Gaussiano + Lineare:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('a        (%.3f ± %.3f)' %(popt[3],np.sqrt(pcov[3,3])))
print('b        (%.3f ± %.3f)' %(popt[4],np.sqrt(pcov[4,4])))
print('\n--------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n--------------------------------\n')
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n==================================================')



plt.savefig('FIGURE/LSO3F18_gauss.pdf', format='pdf',bbox_inches="tight",␣
↪dpi=100)
plt.show()
```

===

```
Entries   133630
Mean      1.775
Std Dev   1.240

#--------------------------------

# Fit Gaussiano:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)

# Parametri del fit:
A        (2791.783 ± 90.222)
mu       (3.299 ± 0.006)
sigma    (0.248 ± 0.005)

---------------------------------

# Chi square test:
Chi2      2161.447
dof       84
Chi2/dof  25.732
pvalue    0.000

---------------------------------

FWHM      (0.584 ± 0.012)
R         (17.698 ± 0.367)%

=================================================
=================================================

Entries   133630
Mean      1.775
Std Dev   1.240

#--------------------------------

# Fit Gaussiano + Lineare:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b

# Parametri del fit:
A        (2825.712 ± 55.544)
mu       (3.302 ± 0.004)
sigma    (0.232 ± 0.003)
a        (-20.710 ± 2.502)
b        (132.350 ± 14.199)

---------------------------------
```

```
# Chi square test:
Chi2      725.523
dof       84
Chi2/dof  8.637
pvalue    0.000

---------------------------------

FWHM      (0.547 ± 0.008)
R         (16.578 ± 0.234)%

=================================================
```



1.3.7 LSO1 - ^{133}Ba

[296]:
```
# LSO1 - Ba133

# Entries, Media, Standard deviation, Errore sulla media
entries    = len(LSO1Ba133)
mean, std  = np.mean(LSO1Ba133), np.std(LSO1Ba133)
mean_error = std/np.sqrt(len(LSO1Ba133))

# Creazione dell'istogramma (con barre di errore)
fig                             = plt.figure(1, figsize = (10,6))
```

```python
bin_heights, bin_borders, _ = plt.
 ↪hist(LS01Ba133,bins=200,facecolor='g',ec='black',alpha=0.5,density=False) #␣
 ↪label='histogram data'
bin_centers                 = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights               = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black') # label='error'


# FIT GAUSSIANO
# Definisco gli estremi di intervallo su cui eseguire il fit gaussiano
a = 1.75
b = 2.8



# Creazione dei vettori per il fit sul fotopicco => elimino la parte che non␣
 ↪interessa
bin_centers_new = []
bin_heights_new = []

for i in range(len(bin_heights)):
    if bin_centers[i] < b and bin_centers[i] > a:
        bin_centers_new.append(bin_centers[i])
        bin_heights_new.append(bin_heights[i])
    else:
        pass

bin_centers_new   = np.array(bin_centers_new)
bin_heights_new   = np.array(bin_heights_new)
sigma_heights_new = np.sqrt(bin_heights_new)

#================================================
# Funzione di fit
#================================================
def fit_gauss(x, A1, mu1, sigma1, A2, mu2, sigma2):
    fotopicco1 = A1*np.exp(-(x-mu1)**2/(2*sigma1**2))
    fotopicco2 = A2*np.exp(-(x-mu2)**2/(2*sigma2**2))
    return fotopicco1 + fotopicco2


# Parametri iniziali
#param0     = [entries, (a+b)/2, std, entries, (a+b)/2, std]
# Best Parameters
#popt, pcov = curve_fit(fit_gauss, bin_centers_new, bin_heights_new, param0,␣
 ↪sigma = sigma_heights_new)
```

```
popt, pcov = curve_fit(fit_gauss, bin_centers_new, bin_heights_new, sigma =␣
↪sigma_heights_new)
# Best-Parameters
A1            = popt[0]
sigma_A1      = np.sqrt(pcov[0,0])
mu1           = popt[1]
sigma_mu1     = np.sqrt(pcov[1,1])
sigma1        = popt[2]
sigma_sigma1 = np.sqrt(pcov[2,2])
A2            = popt[3]
sigma_A2      = np.sqrt(pcov[3,3])
mu2           = popt[4]
sigma_mu2     = np.sqrt(pcov[4,4])
sigma2        = popt[5]
sigma_sigma2 = np.sqrt(pcov[5,5])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
x = np.linspace(min(bin_centers),max(bin_centers),500)
#x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss(x, *popt), '-', color="blue", label='Double Gauss')


#================================================
# CHI2 TEST
#================================================
chi2    = sum(((bin_heights_new - fit_gauss(bin_centers_new, *popt)) /␣
↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM1       = 2*np.sqrt(2*np.log(2))*abs(sigma1)
sigma_FWHM1 = 2*np.sqrt(2*np.log(2))*sigma_sigma1
FWHM2       = 2*np.sqrt(2*np.log(2))*abs(sigma2)
sigma_FWHM2 = 2*np.sqrt(2*np.log(2))*sigma_sigma2

# Risuoluzione
R1       = (FWHM1/mu1) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R1 = R1 * (np.sqrt((sigma_FWHM1/FWHM1)**2 + (sigma_mu1/mu1)**2))  # error␣
↪propagation
R2       = (FWHM2/mu2) * 100      # %Energy resolution = FWHM x 100 /photo peak
```

```
sigma_R2 = R1 * (np.sqrt((sigma_FWHM2/FWHM2)**2 + (sigma_mu2/mu2)**2))  # error␣
↪propagation


# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('LSO1 - Ba133')
plt.xlim(1, 7)
plt.ylim(0,4000)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu_{1}$ ␣
↪  (%.2f $\pm$ %.2f)'           %(mu1, sigma_mu1))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM_{1}$␣
↪ (%.2f $\pm$ %.2f)'    %(FWHM1, sigma_FWHM1))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
↪label=r'$\frac{\Delta E_{1}}{E_{1}}$     (%.2f $\pm$ %.2f) %%'       %(R1,␣
↪sigma_R1))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu_{2}$ ␣
↪  (%.2f $\pm$ %.2f)'           %(mu2, sigma_mu2))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM_{2}$␣
↪ (%.2f $\pm$ %.2f)'    %(FWHM2, sigma_FWHM2))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
↪label=r'$\frac{\Delta E_{2}}{E_{2}}$     (%.2f $\pm$ %.2f) %%'       %(R2,␣
↪sigma_R2))
plt.legend(frameon=False,fancybox=True, shadow=False, loc='best', prop={"size":
↪12}, numpoints = 1)

#==================================================
# STAMPA RISULTATI DEL FIT
#==================================================
print('==================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n--------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n#-------------------------------\n')
print('# Fit Gaussiano:')
print('p(x) = A1 * exp(-(x-mu1)^2/(2 * sigma1^2) + A2 * exp(-(x-mu2)^2/(2 *␣
↪sigma2^2)\n')
```

```
print('# Parametri del fit:')
print('A1        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu1       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma1    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('A2        (%.3f ± %.3f)' %(popt[3],np.sqrt(pcov[3,3])))
print('mu2       (%.3f ± %.3f)' %(popt[4],np.sqrt(pcov[4,4])))
print('sigma2    (%.3f ± %.3f)' %(popt[5],np.sqrt(pcov[5,5])))
print('\n---------------------------------\n')
print('FWHM1      (%.3f ± %.3f)'   %(FWHM1,sigma_FWHM1))
print('R1         (%.3f ± %.3f)%%' %(R1,sigma_R1))
print('FWHM2      (%.3f ± %.3f)'   %(FWHM2,sigma_FWHM2))
print('R2         (%.3f ± %.3f)%%' %(R2,sigma_R2))
print('\n=================================================')



#=================================================
# Funzione di fit
#=================================================
def fit_gauss_linear(x, A1, mu1, sigma1, A2, mu2, sigma2, a, b):
    fotopicco1 = A1*np.exp(-(x-mu1)**2/(2*sigma1**2))  # Modello gaussiano per␣
 ↪il fotopicco
    fotopicco2 = A2*np.exp(-(x-mu2)**2/(2*sigma2**2))  # Modello gaussiano per␣
 ↪il fotopicco
    fondo      = a * x + b                             # Modello lineare per il␣
 ↪fondo
    return fotopicco1 + fotopicco2 + fondo


# Parametri iniziali
#param0     = [entries, (a+b)/2, std, entries, (a+b)/2, std]
# Best Parameters
#popt, pcov = curve_fit(fit_gauss, bin_centers_new, bin_heights_new, param0,␣
 ↪sigma = sigma_heights_new)
popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new,␣
 ↪sigma = sigma_heights_new)
# Best-Parameters
A1           = popt[0]
sigma_A1     = np.sqrt(pcov[0,0])
mu1          = popt[1]
sigma_mu1    = np.sqrt(pcov[1,1])
sigma1       = popt[2]
sigma_sigma1 = np.sqrt(pcov[2,2])
A2           = popt[3]
sigma_A2     = np.sqrt(pcov[3,3])
mu2          = popt[4]
sigma_mu2    = np.sqrt(pcov[4,4])
```

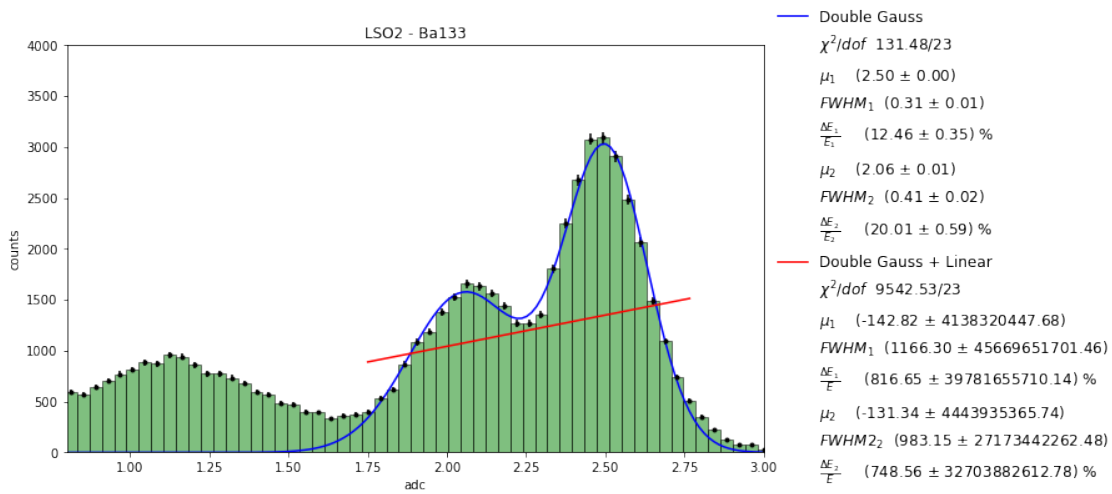
```
sigma2       = popt[5]
sigma_sigma2 = np.sqrt(pcov[5,5])
a            = popt[6]
sigma_a      = np.sqrt(pcov[6,6])
b            = popt[7]
sigma_b     = np.sqrt(pcov[7,7])


# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
#x = np.linspace(min(bin_centers),max(bin_centers),500)
x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss_linear(x, *popt), '-', color="red", label='Double Gauss +␣
↪Linear')



#==================================================
# CHI2 TEST
#==================================================
chi2    = sum(((bin_heights_new - fit_gauss_linear(bin_centers_new, *popt)) /␣
↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM1       = 2 * np.sqrt(2 * np.log(2)) * abs(sigma1)
sigma_FWHM1 = 2 * np.sqrt(2 * np.log(2)) * sigma_sigma1
FWHM2       = 2 * np.sqrt(2 * np.log(2)) * abs(sigma2)
sigma_FWHM2 = 2 * np.sqrt(2 * np.log(2)) * sigma_sigma2

# Risuoluzione
R1       = abs((FWHM1/mu1)) * 100      # %Energy resolution = FWHM x 100 /photo␣
↪peak
sigma_R1 = R1 * (np.sqrt((sigma_FWHM1/FWHM1)**2 + (sigma_mu1/mu1)**2))  # error␣
↪propagation
R2       = abs((FWHM2/mu2)) * 100      # %Energy resolution = FWHM x 100 /photo␣
↪peak
sigma_R2 = R2 * (np.sqrt((sigma_FWHM2/FWHM2)**2 + (sigma_mu2/mu2)**2))  # error␣
↪propagation

# Bellurie
```

```
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('LSO2 - Ba133')
plt.xlim(0.8, 3)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu_{1}$ ␣
↪  (%.2f $\pm$ %.2f)'           %(mu1, sigma_mu1))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM_{1}$␣
↪ (%.2f $\pm$ %.2f)'    %(FWHM1, sigma_FWHM1))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
↪label=r'$\frac{\Delta E_{1}}{E}$     (%.2f $\pm$ %.2f) %%'       %(R1,␣
↪sigma_R1))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu_{2}$ ␣
↪  (%.2f $\pm$ %.2f)'           %(mu2, sigma_mu2))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
↪label=r'$FWHM2_{2}$  (%.2f $\pm$ %.2f)'    %(FWHM2, sigma_FWHM2))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
↪label=r'$\frac{\Delta E_{2}}{E}$     (%.2f $\pm$ %.2f) %%'       %(R2,␣
↪sigma_R2))
plt.legend(frameon=False, fancybox=True, shadow=False, loc='center left',␣
↪prop={"size":12}, numpoints = 1, bbox_to_anchor=(1, 0.5))

#==================================================
# STAMPA RISULTATI DEL FIT
#==================================================
print('====================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#--------------------------------\n')
print('# Fit Gaussiano + Lineare:')
print('p(x) = A1 * exp(-(x-mu1)^2/(2 * sigma1^2) + A2 * exp(-(x-mu2)^2/(2 *␣
↪sigma2^2) + a * x + b\n')
print('# Parametri del fit:')
print('A1        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu1       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma1    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('A2        (%.3f ± %.3f)' %(popt[3],np.sqrt(pcov[3,3])))
print('mu2       (%.3f ± %.3f)' %(popt[4],np.sqrt(pcov[4,4])))
print('sigma2    (%.3f ± %.3f)' %(popt[5],np.sqrt(pcov[5,5])))
print('a        (%.3f ± %.3f)' %(popt[6],np.sqrt(pcov[6,6])))
print('b        (%.3f ± %.3f)' %(popt[7],np.sqrt(pcov[7,7])))
print('\n--------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
```

```python
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n--------------------------------\n')
print('FWHM1       (%.3f ± %.3f)'   %(FWHM1,sigma_FWHM1))
print('R1          (%.3f ± %.3f)%%' %(R1,sigma_R1))
print('FWHM2       (%.3f ± %.3f)'   %(FWHM2,sigma_FWHM2))
print('R2          (%.3f ± %.3f)%%' %(R2,sigma_R2))
print('\n=================================================')




plt.savefig('FIGURE/LS01Ba133_gauss.pdf', format='pdf',bbox_inches="tight",␣
 ↪dpi=100)
plt.show()
```

```
=================================================

Entries   275363
Mean      0.651
Std Dev   0.831

--------------------------------

# Chi square test:
Chi2      131.480
dof       23
Chi2/dof  5.717
pvalue    0.000

#-------------------------------

# Fit Gaussiano:
p(x) = A1 * exp(-(x-mu1)^2/(2 * sigma1^2) + A2 * exp(-(x-mu2)^2/(2 * sigma2^2)

# Parametri del fit:
A1        (2968.527 ± 66.333)
mu1       (2.501 ± 0.005)
sigma1    (0.132 ± 0.004)
A2        (1566.276 ± 42.765)
mu2       (2.056 ± 0.009)
sigma2    (0.175 ± 0.008)

--------------------------------

FWHM1       (0.312 ± 0.009)
```

```
R1         (12.465 ± 0.346)%
FWHM2      (0.411 ± 0.020)
R2         (20.010 ± 0.595)%

=================================================
=================================================

Entries   275363
Mean      0.651
Std Dev   0.831

#--------------------------------

# Fit Gaussiano + Lineare:
p(x) = A1 * exp(-(x-mu1)^2/(2 * sigma1^2) + A2 * exp(-(x-mu2)^2/(2 * sigma2^2) +
a * x + b

# Parametri del fit:
A1        (167.530 ± 5758126917.099)
mu1       (-142.815 ± 4138320447.682)
sigma1    (-495.282 ± 19394115400.805)
A2        (177.161 ± 5369000683.931)
mu2       (-131.340 ± 4443935365.739)
sigma2    (-417.507 ± 11539498451.198)
a        (613.729 ± 6399551.597)
b        (-514.333 ± 8480841107.666)

---------------------------------

# Chi square test:
Chi2      9542.529
dof       23
Chi2/dof  414.893
pvalue    0.000

---------------------------------

FWHM1      (1166.300 ± 45669651701.458)
R1         (816.650 ± 39781655710.137)%
FWHM2      (983.154 ± 27173442262.484)
R2         (748.557 ± 32703882612.782)%

=================================================
```



1.3.8 LSO2 - ^{133}Ba

[309]:
```
# LSO1 - Ba133

# Entries, Media, Standard deviation, Errore sulla media
entries    = len(LSO2Ba133)
mean, std  = np.mean(LSO2Ba133), np.std(LSO2Ba133)
mean_error = std/np.sqrt(len(LSO2Ba133))

# Creazione dell'istogramma (con barre di errore)
fig                            = plt.figure(1, figsize = (10,6))
bin_heights, bin_borders, _ = plt.hist(LSO2Ba133,bins=200,facecolor='g',ec='black',alpha=0.5,density=False) # label='histogram data'
bin_centers                    = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                  = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black', ecolor='black') # label='error'


# FIT GAUSSIANO
# Definisco gli estremi di intervallo su cui eseguire il fit gaussiano
a = 1.75
b = 2.8



# Creazione dei vettori per il fit sul fotopicco => elimino la parte che non interessa
bin_centers_new = []
```

```python
bin_heights_new = []

for i in range(len(bin_heights)):
    if bin_centers[i] < b and bin_centers[i] > a:
        bin_centers_new.append(bin_centers[i])
        bin_heights_new.append(bin_heights[i])
    else:
        pass

bin_centers_new   = np.array(bin_centers_new)
bin_heights_new   = np.array(bin_heights_new)
sigma_heights_new = np.sqrt(bin_heights_new)

#================================================
# Funzione di fit
#================================================
def fit_gauss(x, A1, mu1, sigma1, A2, mu2, sigma2):
    return A1*np.exp(-(x-mu1)**2/(2*sigma1**2)) \
           + A2*np.exp(-(x-mu2)**2/(2*sigma2**2))


# Parametri iniziali
#param0     = [entries, (a+b)/2, std, entries, (a+b)/2, std]
# Best Parameters
#popt, pcov = curve_fit(fit_gauss, bin_centers_new, bin_heights_new, param0,␣
 ↪sigma = sigma_heights_new)
popt, pcov = curve_fit(fit_gauss, bin_centers_new, bin_heights_new, sigma =␣
 ↪sigma_heights_new)
# Best-Parameters
A1           = popt[0]
sigma_A1     = np.sqrt(pcov[0,0])
mu1          = popt[1]
sigma_mu1    = np.sqrt(pcov[1,1])
sigma1       = popt[2]
sigma_sigma1 = np.sqrt(pcov[2,2])
A2           = popt[3]
sigma_A2     = np.sqrt(pcov[3,3])
mu2          = popt[4]
sigma_mu2    = np.sqrt(pcov[4,4])
sigma2       = popt[5]
sigma_sigma2 = np.sqrt(pcov[5,5])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
x = np.linspace(min(bin_centers),max(bin_centers),500)
#x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
```

```
plt.plot(x, fit_gauss(x, *popt), '--', color="blue", label='Double Gauss')


#=================================================
# CHI2 TEST
#=================================================
chi2    = sum(((bin_heights_new - fit_gauss(bin_centers_new, *popt)) /␣
 ↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM1       = 2*np.sqrt(2*np.log(2))*sigma1
sigma_FWHM1 = 2*np.sqrt(2*np.log(2))*sigma_sigma1
FWHM2       = 2*np.sqrt(2*np.log(2))*sigma2
sigma_FWHM2 = 2*np.sqrt(2*np.log(2))*sigma_sigma2

# Risuoluzione
R1       = (FWHM1/mu1) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R1 = R1 * (np.sqrt((sigma_FWHM1/FWHM1)**2 + (sigma_mu1/mu1)**2))  # error␣
 ↪propagation
R2       = (FWHM2/mu2) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R2 = R1 * (np.sqrt((sigma_FWHM2/FWHM2)**2 + (sigma_mu2/mu2)**2))  # error␣
 ↪propagation



# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('LSO2 - Ba133')
plt.xlim(1, 3)
plt.ylim(0,4000)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
 ↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu_{1}$ ␣
 ↪  (%.2f $\pm$ %.2f)'          %(mu1, sigma_mu1))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM_{1}$␣
 ↪ (%.2f $\pm$ %.2f)'    %(FWHM1, sigma_FWHM1))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
 ↪label=r'$\frac{\Delta E_{1}}{E_{1}}$     (%.2f $\pm$ %.2f) %%'       %(R1,␣
 ↪sigma_R1))
```

```
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu_{2}$ ␣
↪  (%.2f $\pm$ %.2f)'          %(mu2, sigma_mu2))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM_{2}$␣
↪ (%.2f $\pm$ %.2f)'    %(FWHM2, sigma_FWHM2))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
↪label=r'$\frac{\Delta E_{2}}{E_{2}}$     (%.2f $\pm$ %.2f) %%'       %(R2,␣
↪sigma_R2))
plt.legend(frameon=False,fancybox=True, shadow=False, loc='best', prop={"size":
↪12}, numpoints = 1)


#=================================================
# STAMPA RISULTATI DEL FIT
#=================================================
print('=====================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n----------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n#---------------------------------\n')
print('# Fit Gaussiano:')
print('p(x) = A1 * exp(-(x-mu1)^2/(2 * sigma1^2) + A2 * exp(-(x-mu2)^2/(2 *␣
↪sigma2^2)\n')
print('# Parametri del fit:')
print('A1        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu1       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma1    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('A2        (%.3f ± %.3f)' %(popt[3],np.sqrt(pcov[3,3])))
print('mu2       (%.3f ± %.3f)' %(popt[4],np.sqrt(pcov[4,4])))
print('sigma2    (%.3f ± %.3f)' %(popt[5],np.sqrt(pcov[5,5])))
print('\n----------------------------------\n')
print('FWHM1      (%.3f ± %.3f)'   %(FWHM1,sigma_FWHM1))
print('R1         (%.3f ± %.3f)%%' %(R1,sigma_R1))
print('FWHM2      (%.3f ± %.3f)'   %(FWHM2,sigma_FWHM2))
print('R2         (%.3f ± %.3f)%%' %(R2,sigma_R2))
print('\n===================================================')



#=================================================
# Funzione di fit
#=================================================
```

```
def fit_gauss_linear(x, A1, mu1, sigma1, A2, mu2, sigma2, a, b):
    fotopicco1 = A1*np.exp(-(x-mu1)**2/(2*sigma1**2))  # Modello gaussiano per␣
 ↪il fotopicco
    fotopicco2 = A2*np.exp(-(x-mu2)**2/(2*sigma2**2))  # Modello gaussiano per␣
 ↪il fotopicco
    fondo      = a * x + b                             # Modello lineare per il␣
 ↪fondo
    return fotopicco1 + fotopicco2 + fondo


# Parametri iniziali
#param0     = [entries, (a+b)/2, std, entries, (a+b)/2, std]
# Best Parameters
#popt, pcov = curve_fit(fit_gauss, bin_centers_new, bin_heights_new, param0,␣
 ↪sigma = sigma_heights_new)
popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new,␣
 ↪sigma = sigma_heights_new)
# Best-Parameters
A1           = popt[0]
sigma_A1     = np.sqrt(pcov[0,0])
mu1          = popt[1]
sigma_mu1    = np.sqrt(pcov[1,1])
sigma1       = popt[2]
sigma_sigma1 = np.sqrt(pcov[2,2])
A2           = popt[3]
sigma_A2     = np.sqrt(pcov[3,3])
mu2          = popt[4]
sigma_mu2    = np.sqrt(pcov[4,4])
sigma2       = popt[5]
sigma_sigma2 = np.sqrt(pcov[5,5])
a            = popt[6]
sigma_a      = np.sqrt(pcov[6,6])
b            = popt[7]
sigma_b      = np.sqrt(pcov[7,7])


# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
#x = np.linspace(min(bin_centers),max(bin_centers),500)
x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss_linear(x, *popt), '-', color="red", label='Double Gauss +␣
 ↪Linear Model')


#================================================
```

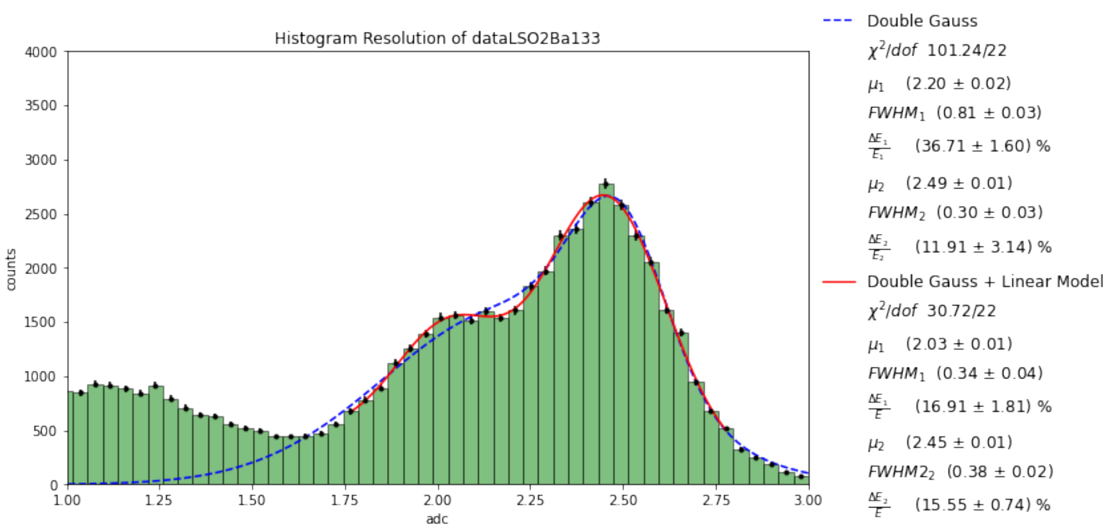
```
# CHI2 TEST
#==================================================
chi2    = sum(((bin_heights_new - fit_gauss_linear(bin_centers_new, *popt)) /␣
 ↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM1       = 2 * np.sqrt(2 * np.log(2)) * abs(sigma1)
sigma_FWHM1 = 2 * np.sqrt(2 * np.log(2)) * sigma_sigma1
FWHM2       = 2 * np.sqrt(2 * np.log(2)) * abs(sigma2)
sigma_FWHM2 = 2 * np.sqrt(2 * np.log(2)) * sigma_sigma2

# Risuoluzione
R1       = (FWHM1/mu1) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R1 = R1 * (np.sqrt((sigma_FWHM1/FWHM1)**2 + (sigma_mu1/mu1)**2))  # error␣
 ↪propagation
R2       = (FWHM2/mu2) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R2 = R2 * (np.sqrt((sigma_FWHM2/FWHM2)**2 + (sigma_mu2/mu2)**2))  # error␣
 ↪propagation

# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('Histogram Resolution of dataLSO2Ba133')
#plt.xlim(0.8, 1.5)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
 ↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu_{1}$ ␣
 ↪  (%.2f $\pm$ %.2f)'          %(mu1, sigma_mu1))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM_{1}$␣
 ↪ (%.2f $\pm$ %.2f)'    %(FWHM1, sigma_FWHM1))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
 ↪label=r'$\frac{\Delta E_{1}}{E}$     (%.2f $\pm$ %.2f) %%'       %(R1,␣
 ↪sigma_R1))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu_{2}$ ␣
 ↪  (%.2f $\pm$ %.2f)'          %(mu2, sigma_mu2))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
 ↪label=r'$FWHM2_{2}$  (%.2f $\pm$ %.2f)'    %(FWHM2, sigma_FWHM2))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
 ↪label=r'$\frac{\Delta E_{2}}{E}$     (%.2f $\pm$ %.2f) %%'       %(R2,␣
 ↪sigma_R2))
```

```
plt.legend(frameon=False, fancybox=True, shadow=False, loc='center left',␣
 ↪prop={"size":12}, numpoints = 1, bbox_to_anchor=(1, 0.5))

#================================================
# STAMPA RISULTATI DEL FIT
#================================================
print('================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#-------------------------------\n')
print('# Fit Gaussiano + Lineare:')
print('p(x) = A1 * exp(-(x-mu1)^2/(2 * sigma1^2) + A2 * exp(-(x-mu2)^2/(2 *␣
 ↪sigma2^2) + a * x + b\n')
print('# Parametri del fit:')
print('A1        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu1       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma1    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('A2        (%.3f ± %.3f)' %(popt[3],np.sqrt(pcov[3,3])))
print('mu2       (%.3f ± %.3f)' %(popt[4],np.sqrt(pcov[4,4])))
print('sigma2    (%.3f ± %.3f)' %(popt[5],np.sqrt(pcov[5,5])))
print('a        (%.3f ± %.3f)' %(popt[6],np.sqrt(pcov[6,6])))
print('b        (%.3f ± %.3f)' %(popt[7],np.sqrt(pcov[7,7])))
print('\n-------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n-------------------------------\n')
print('FWHM1      (%.3f ± %.3f)'   %(FWHM1,sigma_FWHM1))
print('R1         (%.3f ± %.3f)%%' %(R1,sigma_R1))
print('FWHM2      (%.3f ± %.3f)'   %(FWHM2,sigma_FWHM2))
print('R2         (%.3f ± %.3f)%%' %(R2,sigma_R2))
print('\n================================================')


plt.savefig('FIGURE/LSO2Ba133_gauss.pdf', format='pdf',bbox_inches="tight",␣
 ↪dpi=100)
plt.show()
```

```
================================================

Entries   274755
Mean      0.645
Std Dev   0.825
```

```
---------------------------------

# Chi square test:
Chi2      101.244
dof       22
Chi2/dof  4.602
pvalue    0.000

#--------------------------------

# Fit Gaussiano:
p(x) = A1 * exp(-(x-mu1)^2/(2 * sigma1^2) + A2 * exp(-(x-mu2)^2/(2 * sigma2^2)

# Parametri del fit:
A1        (1615.362 ± 68.043)
mu1       (2.198 ± 0.024)
sigma1    (0.343 ± 0.014)
A2        (1497.968 ± 119.366)
mu2       (2.490 ± 0.007)
sigma2    (0.126 ± 0.011)

---------------------------------

FWHM1      (0.807 ± 0.034)
R1         (36.711 ± 1.599)%
FWHM2      (0.296 ± 0.025)
R2         (11.907 ± 3.143)%

=================================================
=================================================

Entries   274755
Mean      0.645
Std Dev   0.825

#--------------------------------

# Fit Gaussiano + Lineare:
p(x) = A1 * exp(-(x-mu1)^2/(2 * sigma1^2) + A2 * exp(-(x-mu2)^2/(2 * sigma2^2) +
a * x + b

# Parametri del fit:
A1        (1073.329 ± 75.271)
mu1       (2.033 ± 0.011)
sigma1    (-0.146 ± 0.016)
A2        (2378.457 ± 52.779)
mu2       (2.454 ± 0.005)
sigma2    (0.162 ± 0.008)
```

```
a         (-286.273 ± 130.977)
b         (975.244 ± 316.698)

---------------------------------

# Chi square test:
Chi2       30.719
dof        22
Chi2/dof   1.396
pvalue     0.102

---------------------------------

FWHM1       (0.344 ± 0.037)
R1          (16.908 ± 1.809)%
FWHM2       (0.382 ± 0.018)
R2          (15.550 ± 0.739)%

=================================================
```



1.3.9 NaI1 - ^{99m}Tc

[300]:
```
# NaI1 - Tc99m

# Entries, Media, Standard deviation, Errore sulla media
entries    = len(NaI1Tc99m)
mean, std  = np.mean(NaI1Tc99m), np.std(NaI1Tc99m)
mean_error = std/np.sqrt(len(NaI1Tc99m))
```

```python
# Creazione dell'istogramma (con barre di errore)
fig                            = plt.figure(1, figsize = (10,6))
bin_heights, bin_borders, _ = plt.
 ↪hist(NaI1Tc99m,bins=200,facecolor='g',ec='black',alpha=0.5, density=False) #␣
 ↪label='histogram data'
bin_centers                    = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                  = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black') # label='error'


# FIT GAUSSIANO
# Definisco gli estremi di intervallo su cui eseguire il fit gaussiano
a = 1.26
b = 1.4

# Creazione dei vettori per il fit sul fotopicco => elimino la parte che non␣
 ↪interessa
bin_centers_new = []
bin_heights_new = []

for i in range(len(bin_heights)):
    if bin_centers[i] < b and bin_centers[i] > a:
        bin_centers_new.append(bin_centers[i])
        bin_heights_new.append(bin_heights[i])
    else:
        pass

bin_centers_new   = np.array(bin_centers_new)
bin_heights_new   = np.array(bin_heights_new)
sigma_heights_new = np.sqrt(bin_heights_new)

#================================================
# Funzione di fit
#================================================
def fit_gauss(x, A, mu, sigma):
    return A*np.exp(-(x-mu)**2/(2*sigma**2))

# Parametri iniziali
param0     = [entries, mean, std]
# Best Parameters
popt, pcov = curve_fit(fit_gauss, bin_centers_new, bin_heights_new, param0,␣
 ↪sigma = sigma_heights_new)

# Best-Parameters
A            = popt[0]
sigma_A      = np.sqrt(pcov[0,0])
```

```
mu          = popt[1]
sigma_mu    = np.sqrt(pcov[1,1])
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
x = np.linspace(min(bin_centers),max(bin_centers),500)
#x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss(x, *popt), '-', color="blue", label='Gauss')


#================================================
# CHI2 TEST
#================================================
chi2    = sum(((bin_heights_new - fit_gauss(bin_centers_new, *popt)) /␣
 ↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2*np.sqrt(2*np.log(2))*sigma
sigma_FWHM = 2*np.sqrt(2*np.log(2))*sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error␣
 ↪propagation



# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('NaI1 - Tc99m')
plt.xlim(0.8, 1.5)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
 ↪$dof$  %.2f/%.i'     %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$   ␣
 ↪(%.2f $\pm$ %.2f)'          %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.
 ↪2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
```

```
plt.plot([], [], color='white', marker='.',linestyle='None',␣
↪label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
plt.legend(frameon=False, fancybox=True, shadow=False, loc='best', prop={"size":
↪12}, numpoints = 1)


#=================================================
# STAMPA RISULTATI DEL FIT
#=================================================
print('==================================================\n')
print('Entries    %.i'    %entries)
print('Mean       %.3f'   %mean)
print('Std Dev    %.3f'   %std)
print('\n#--------------------------------\n')
print('# Fit Gaussiano:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)\n')
print('# Parametri del fit:')
print('A         (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu        (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma     (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('\n--------------------------------\n')
print('# Chi square test:')
print('Chi2       %.3f' %chi2)
print('dof        %.i'  %dof)
print('Chi2/dof   %.3f' %chi2_rid)
print('pvalue     %.3f' %pvalue)
print('\n--------------------------------\n')
print('FWHM       (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R          (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n================================================')


#=================================================
# Funzione di fit
#=================================================
def fit_gauss_linear(x, A, mu, sigma, a, b):
    fotopicco = A*np.exp(-(x-mu)**2/(2*sigma**2))   # Modello gaussiano per il␣
↪fotopicco
    fondo     = a * x + b                           # Modello lineare per il␣
↪fondo
    return fotopicco + fondo

# Parametri iniziali
#param0     = [entries, (a+b)/2, std,0,0]
# Best Parameters
#popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new,␣
↪param0, sigma = sigma_heights_new)
```

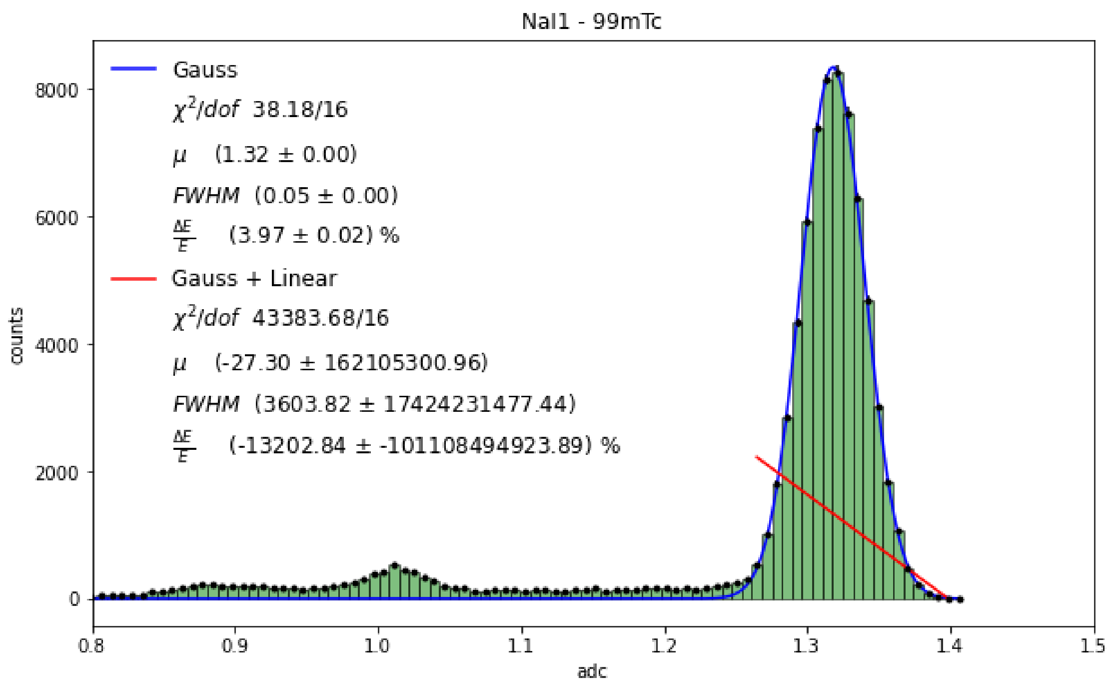
```python
popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new,␣
 ↪sigma = sigma_heights_new)
# Best-Parameters
A           = popt[0]
sigma_A     = np.sqrt(pcov[0,0])
mu          = popt[1]
sigma_mu    = np.sqrt(pcov[1,1])
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])
a           = popt[3]
sigma_a     = np.sqrt(pcov[3,3])
b           = popt[4]
sigma_b     = np.sqrt(pcov[4,4])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
#x = np.linspace(min(bin_centers),max(bin_centers),500)
x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss_linear(x, *popt), '-', color="red", label='Gauss +␣
 ↪Linear')


#=================================================
# CHI2 TEST
#=================================================
chi2    = sum(((bin_heights_new - fit_gauss_linear(bin_centers_new, *popt)) /␣
 ↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2 * np.sqrt(2 * np.log(2)) * abs(sigma)
sigma_FWHM = 2 * np.sqrt(2 * np.log(2)) * sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100       # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error␣
 ↪propagation


# Bellurie
plt.xlabel('adc')
```

```python
plt.ylabel('counts')
plt.title('NaI1 - 99mTc')
plt.xlim(0.8, 1.5)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$   ␣
↪(%.2f $\pm$ %.2f)'          %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.
↪2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
↪label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
plt.legend(frameon=False, fancybox=True, shadow=False, loc='best', prop={"size":
↪12}, numpoints = 1)

#=================================================
# STAMPA RISULTATI DEL FIT
#=================================================
print('================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#--------------------------------\n')
print('# Fit Gaussiano + Lineare:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('a        (%.3f ± %.3f)' %(popt[3],np.sqrt(pcov[3,3])))
print('b        (%.3f ± %.3f)' %(popt[4],np.sqrt(pcov[4,4])))
print('\n--------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n--------------------------------\n')
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n================================================')

plt.savefig('FIGURE/NaI1Tc99m_gauss.pdf', format='pdf',bbox_inches="tight",␣
↪dpi=100)
plt.show()
```

```
================================================
```

```
Entries   87875
Mean      1.157
Std Dev   0.346

#--------------------------------

# Fit Gaussiano:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)

# Parametri del fit:
A        (8338.007 ± 60.783)
mu       (1.318 ± 0.000)
sigma    (0.022 ± 0.000)

---------------------------------

# Chi square test:
Chi2      38.182
dof       16
Chi2/dof  2.386
pvalue    0.001

---------------------------------

FWHM      (0.052 ± 0.000)
R         (3.967 ± 0.018)%

=================================================
=================================================

Entries   87875
Mean      1.157
Std Dev   0.346

#--------------------------------

# Fit Gaussiano + Lineare:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b

# Parametri del fit:
A        (11595.646 ± 10584751118.699)
mu       (-27.296 ± 162105300.960)
sigma    (-1530.403 ± 7399389823.527)
a        (-16574.419 ± 2086659.718)
b        (11586.815 ± 10582047096.652)

---------------------------------
```

```
# Chi square test:
Chi2       43383.684
dof        16
Chi2/dof   2711.480
pvalue     0.000

---------------------------------

FWHM       (3603.825 ± 17424231477.439)
R          (-13202.844 ± -101108494923.887)%

=================================================
```



1.3.10 NaI2 - ^{99m}Tc

[301]:
```
# NaI2 - Tc99m

# Entries, Media, Standard deviation, Errore sulla media
entries     = len(NaI2Tc99m)
mean, std   = np.mean(NaI2Tc99m), np.std(NaI2Tc99m)
mean_error = std/np.sqrt(len(NaI2Tc99m))

# Creazione dell'istogramma (con barre di errore)
fig                             = plt.figure(1, figsize = (10,6))
```

```python
bin_heights, bin_borders, _ = plt.hist(NaI2Tc99m,bins=200,facecolor='g',ec='black',alpha=0.5 ,density=False) # label='histogram data'
bin_centers                  = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black', ecolor='black') # label='error'


# FIT GAUSSIANO
# Definisco gli estremi di intervallo su cui eseguire il fit gaussiano
a = 0.9
b = 1.2
# Creazione dei vettori per il fit sul fotopicco => elimino la parte che non interessa
bin_centers_new = []
bin_heights_new = []

for i in range(len(bin_heights)):
    if bin_centers[i] < b and bin_centers[i] > a:
        bin_centers_new.append(bin_centers[i])
        bin_heights_new.append(bin_heights[i])
    else:
        pass

bin_centers_new   = np.array(bin_centers_new)
bin_heights_new   = np.array(bin_heights_new)
sigma_heights_new = np.sqrt(bin_heights_new)

#================================================
# Funzione di fit
#================================================
def fit_gauss(x, A, mu, sigma):
    return A*np.exp(-(x-mu)**2/(2*sigma**2))

# Parametri iniziali
param0     = [entries, mean, std]
# Best Parameters
popt, pcov = curve_fit(fit_gauss, bin_centers_new, bin_heights_new, param0, sigma = sigma_heights_new)

# Best-Parameters
A           = popt[0]
sigma_A     = np.sqrt(pcov[0,0])
mu          = popt[1]
sigma_mu    = np.sqrt(pcov[1,1])
sigma       = popt[2]
```

```
sigma_sigma = np.sqrt(pcov[2,2])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
x = np.linspace(min(bin_centers),max(bin_centers),500)
#x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss(x, *popt), '-', color="blue", label='Gauss')


#==================================================
# CHI2 TEST
#==================================================
chi2    = sum(((bin_heights_new - fit_gauss(bin_centers_new, *popt)) /␣
 ↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2*np.sqrt(2*np.log(2))*sigma
sigma_FWHM = 2*np.sqrt(2*np.log(2))*sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error␣
 ↪propagation


# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('NaI2 - Tc99m')
plt.xlim(-0.05, 1.4)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
 ↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$   ␣
 ↪(%.2f $\pm$ %.2f)'          %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.
 ↪2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
 ↪label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
```

```
plt.legend(frameon=False, fancybox=True, shadow=False, loc='best', prop={"size":
↪12}, numpoints = 1)


#==================================================
# STAMPA RISULTATI DEL FIT
#==================================================
print('==================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#--------------------------------\n')
print('# Fit Gaussiano:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('\n--------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n--------------------------------\n')
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n==================================================')



#==================================================
# Funzione di fit
#==================================================
def fit_gauss_linear(x, A, mu, sigma, a, b):
    fotopicco = A*np.exp(-(x-mu)**2/(2*sigma**2))   # Modello gaussiano per il␣
↪fotopicco
    fondo     = a * x + b                           # Modello lineare per il␣
↪fondo
    return fotopicco + fondo

# Parametri iniziali
#param0     = [entries, (a+b)/2, std,0,0]
# Best Parameters
#popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new,␣
↪param0, sigma = sigma_heights_new)
```

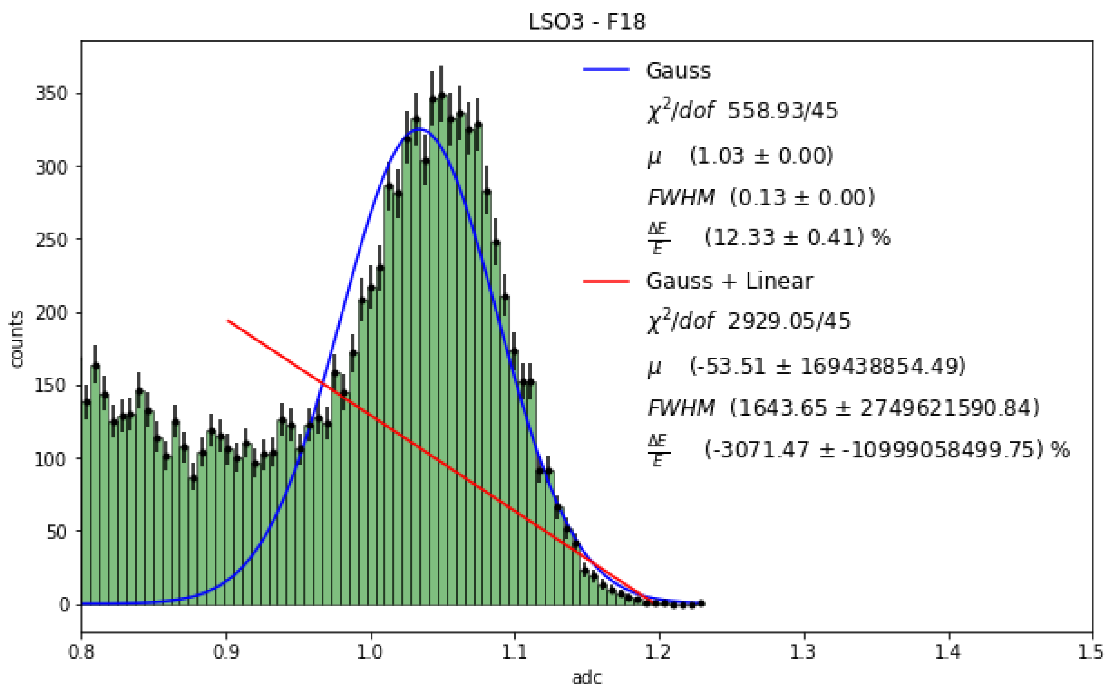
```python
popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new,␣
 ↪sigma = sigma_heights_new)
# Best-Parameters
A           = popt[0]
sigma_A     = np.sqrt(pcov[0,0])
mu          = popt[1]
sigma_mu    = np.sqrt(pcov[1,1])
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])
a           = popt[3]
sigma_a     = np.sqrt(pcov[3,3])
b           = popt[4]
sigma_b     = np.sqrt(pcov[4,4])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
#x = np.linspace(min(bin_centers),max(bin_centers),500)
x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss_linear(x, *popt), '-', color="red", label='Gauss +␣
 ↪Linear')


#==================================================
# CHI2 TEST
#==================================================
chi2    = sum(((bin_heights_new - fit_gauss_linear(bin_centers_new, *popt)) /␣
 ↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2 * np.sqrt(2 * np.log(2)) * abs(sigma)
sigma_FWHM = 2 * np.sqrt(2 * np.log(2)) * sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error␣
 ↪propagation


# Bellurie
plt.xlabel('adc')
```

```
plt.ylabel('counts')
plt.title('LSO3 - F18')
plt.xlim(0.8, 1.5)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$   ␣
↪(%.2f $\pm$ %.2f)'          %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.
↪2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
↪label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
plt.legend(frameon=False, fancybox=True, shadow=False, loc='best', prop={"size":
↪12}, numpoints = 1)

#================================================
# STAMPA RISULTATI DEL FIT
#================================================
print('================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#--------------------------------\n')
print('# Fit Gaussiano + Lineare:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('a        (%.3f ± %.3f)' %(popt[3],np.sqrt(pcov[3,3])))
print('b        (%.3f ± %.3f)' %(popt[4],np.sqrt(pcov[4,4])))
print('\n--------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n--------------------------------\n')
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n================================================')

plt.savefig('FIGURE/NaI2Tc99m_gauss.pdf', format='pdf',bbox_inches="tight",␣
↪dpi=100)
plt.show()
```

```
================================================
```

```
Entries   26181
Mean      0.687
Std Dev   0.282

#---------------------------------

# Fit Gaussiano:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)

# Parametri del fit:
A        (324.746 ± 16.849)
mu       (1.034 ± 0.002)
sigma    (0.054 ± 0.002)

----------------------------------

# Chi square test:
Chi2      558.934
dof       45
Chi2/dof  12.421
pvalue    0.000

----------------------------------

FWHM      (0.127 ± 0.004)
R         (12.327 ± 0.413)%

==================================================
==================================================

Entries   26181
Mean      0.687
Std Dev   0.282

#---------------------------------

# Fit Gaussiano + Lineare:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b

# Parametri del fit:
A        (393.907 ± nan)
mu       (-53.513 ± 169438854.492)
sigma    (697.993 ± 1167656779.820)
a        (-655.900 ± 179266.741)
b        (392.330 ± nan)

----------------------------------
```

```
# Chi square test:
Chi2      2929.053
dof       45
Chi2/dof  65.090
pvalue    0.000

---------------------------------

FWHM      (1643.647 ± 2749621590.837)
R         (-3071.466 ± -10999058499.752)%

=================================================

/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:136:
RuntimeWarning: invalid value encountered in sqrt
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:144:
RuntimeWarning: invalid value encountered in sqrt
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:196:
RuntimeWarning: invalid value encountered in sqrt
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:200:
RuntimeWarning: invalid value encountered in sqrt
```



1.3.11 NaI3 - ^{99m}Tc

[304]:
```
# NaI3 - Tc99m

# Entries, Media, Standard deviation, Errore sulla media
entries     = len(NaI3Tc99m)
mean, std   = np.mean(NaI3Tc99m), np.std(NaI3Tc99m)
mean_error = std/np.sqrt(len(NaI3Tc99m))

# Creazione dell'istogramma (con barre di errore)
fig                            = plt.figure(1, figsize = (10,6))
bin_heights, bin_borders, _ = plt.
 ↪hist(NaI3Tc99m,bins=200,facecolor='g',ec='black',alpha=0.5, density=False) #␣
 ↪label='histogram data'
bin_centers                    = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                  = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black') # label='error'


# FIT GAUSSIANO
# Definisco gli estremi di intervallo su cui eseguire il fit gaussiano
a = 0.481
b = 0.66

# Creazione dei vettori per il fit sul fotopicco => elimino la parte che non␣
 ↪interessa
bin_centers_new = []
bin_heights_new = []

for i in range(len(bin_heights)):
    if bin_centers[i] < b and bin_centers[i] > a:
        bin_centers_new.append(bin_centers[i])
        bin_heights_new.append(bin_heights[i])
    else:
        pass

bin_centers_new   = np.array(bin_centers_new)
bin_heights_new   = np.array(bin_heights_new)
sigma_heights_new = np.sqrt(bin_heights_new)

#=================================================
# Funzione di fit
#=================================================
def fit_gauss(x, A, mu, sigma):
    return A*np.exp(-(x-mu)**2/(2*sigma**2))
```

```python
# Parametri iniziali
param0     = [entries, (a+b)/2, std]
# Best Parameters
popt, pcov = curve_fit(fit_gauss, bin_centers_new, bin_heights_new, param0, sigma = sigma_heights_new)

# Best-Parameters
A           = popt[0]
sigma_A     = np.sqrt(pcov[0,0])
mu          = popt[1]
sigma_mu    = np.sqrt(pcov[1,1])
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
x = np.linspace(min(bin_centers),max(bin_centers),500)
#x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss(x, *popt), '-', color="blue", label='Gauss')


#=================================================
# CHI2 TEST
#=================================================
chi2    = sum(((bin_heights_new - fit_gauss(bin_centers_new, *popt)) / sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2*np.sqrt(2*np.log(2))*sigma
sigma_FWHM = 2*np.sqrt(2*np.log(2))*sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error propagation


# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
```

```
plt.title('NaI3 - Tc99m')
#plt.xlim(-0.05,0.8)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$   ␣
↪(%.2f $\pm$ %.2f)'          %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.
↪2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
↪label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
plt.legend(frameon=False, fancybox=True, shadow=False, loc='best', prop={"size":
↪12}, numpoints = 1)


#================================================
# STAMPA RISULTATI DEL FIT
#================================================
print('================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#--------------------------------\n')
print('# Fit Gaussiano:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('\n--------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n--------------------------------\n')
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n================================================')


#================================================
# Funzione di fit
#================================================
def fit_gauss_linear(x, A, mu, sigma, a, b):
    fotopicco = A*np.exp(-(x-mu)**2/(2*sigma**2))   # Modello gaussiano per il␣
↪fotopicco
```

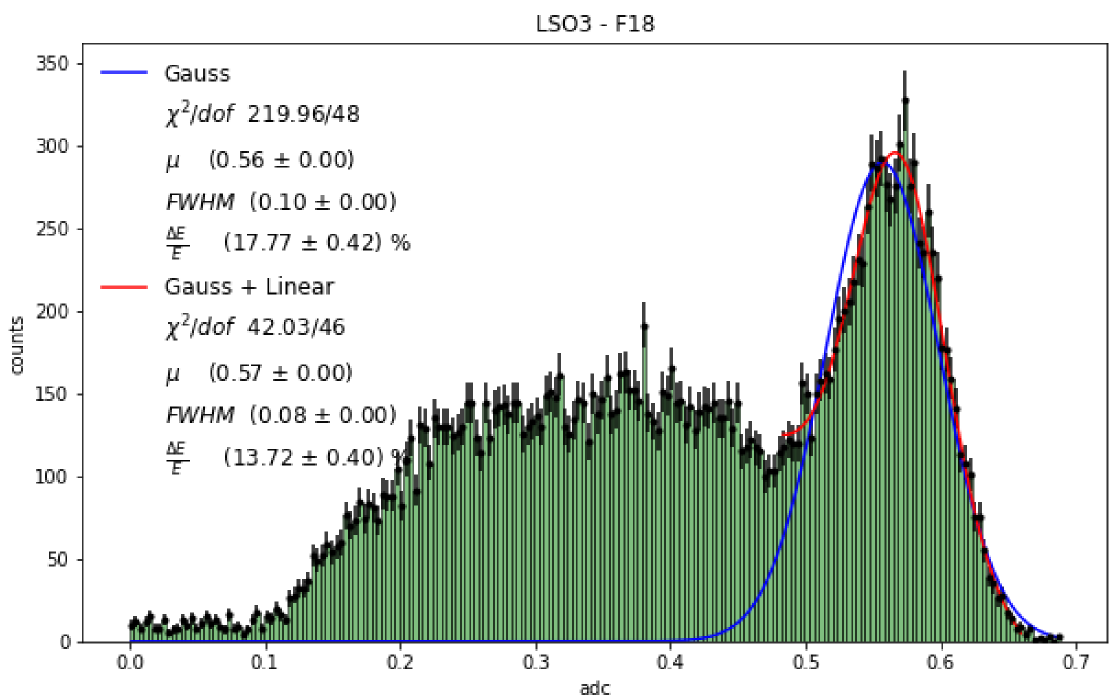
```
    fondo      = a * x + b                           # Modello lineare per il␣
↪fondo
    return fotopicco + fondo

# Parametri iniziali
param0     = [entries, (a+b)/2, std,1,1]
# Best Parameters
popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new,␣
↪param0, sigma = sigma_heights_new)
#popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new,␣
↪sigma = sigma_heights_new)
# Best-Parameters
A           = popt[0]
sigma_A     = np.sqrt(pcov[0,0])
mu          = popt[1]
sigma_mu    = np.sqrt(pcov[1,1])
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])
a           = popt[3]
sigma_a     = np.sqrt(pcov[3,3])
b           = popt[4]
sigma_b     = np.sqrt(pcov[4,4])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
#x = np.linspace(min(bin_centers),max(bin_centers),500)
x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss_linear(x, *popt), '-', color="red", label='Gauss +␣
↪Linear')


#=================================================
# CHI2 TEST
#=================================================
chi2    = sum(((bin_heights_new - fit_gauss_linear(bin_centers_new, *popt)) /␣
↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2 * np.sqrt(2 * np.log(2)) * abs(sigma)
sigma_FWHM = 2 * np.sqrt(2 * np.log(2)) * sigma_sigma
```

```
# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error␣
↪propagation


# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('LSO3 - F18')
#plt.xlim(0.8, 1.5)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$   ␣
↪(%.2f $\pm$ %.2f)'          %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.
↪2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
↪label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
plt.legend(frameon=False, fancybox=True, shadow=False, loc='best', prop={"size":
↪12}, numpoints = 1)

#================================================
# STAMPA RISULTATI DEL FIT
#================================================
print('================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#--------------------------------\n')
print('# Fit Gaussiano + Lineare:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('a        (%.3f ± %.3f)' %(popt[3],np.sqrt(pcov[3,3])))
print('b        (%.3f ± %.3f)' %(popt[4],np.sqrt(pcov[4,4])))
print('\n--------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n--------------------------------\n')
```

```python
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n=================================================')


plt.savefig('FIGURE/NaI3Tc99m_gauss.pdf', format='pdf',bbox_inches="tight",␣
 ↪dpi=100)
plt.show()
```

```
=================================================

Entries   21632
Mean      0.412
Std Dev   0.147

#--------------------------------

# Fit Gaussiano:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)

# Parametri del fit:
A        (289.534 ± 8.529)
mu       (0.557 ± 0.001)
sigma    (0.042 ± 0.001)

---------------------------------

# Chi square test:
Chi2      219.955
dof       48
Chi2/dof  4.582
pvalue    0.000

---------------------------------

FWHM      (0.099 ± 0.002)
R         (17.766 ± 0.424)%

=================================================
=================================================

Entries   21632
Mean      0.412
Std Dev   0.147

#--------------------------------
```

```
# Fit Gaussiano + Lineare:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b

# Parametri del fit:
A         (235.137 ± 5.090)
mu        (0.569 ± 0.001)
sigma     (0.033 ± 0.001)
a         (-667.081 ± 33.642)
b         (439.032 ± 22.375)

---------------------------------

# Chi square test:
Chi2      42.031
dof       46
Chi2/dof  0.914
pvalue    0.639

---------------------------------

FWHM      (0.078 ± 0.002)
R         (13.724 ± 0.397)%

=================================================
```

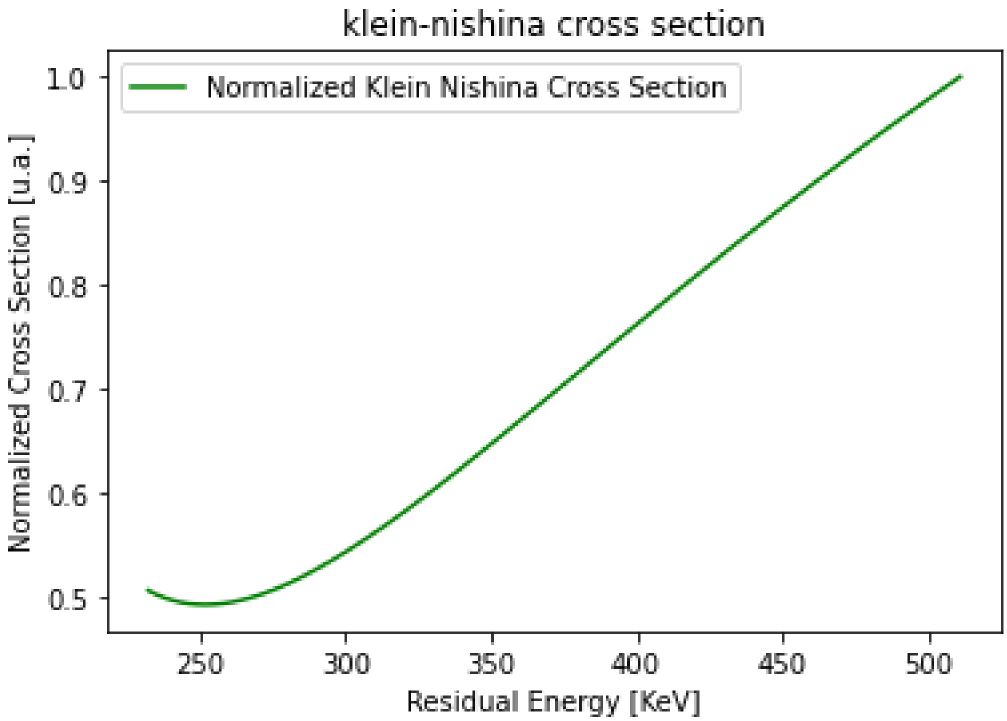


1.4 MODELLO: GAUSS + KLEIN-NISHINA

1.4.1 Formula di Klein - Nishina

[307]:
```
# Costanti fisiche
m_electron_c2 = 500 # [Kev]
energy        = 511 # [Kev]
r_e           = 8e-30
alpha         = 1/137

def ferg(x,A,B):
    return m_electron_c2/energy+1-m_electron_c2/(x*A+B)

def Utility_1(x,A,B):
    return (1+ferg(x,A,B)**2)/2

def Utility_2(x,A,B):
    return (1/(1+alpha*(1-ferg(x,A,B))))**2

def Utility_3(x,A,B):
    return (1+((alpha**2)*((1-ferg(x,A,B))**2))/
 ↪((1+ferg(x,A,B)**2)*(1+alpha*(1-ferg(x,A,B)))))

def kleinnishina(x,A,B,Z):
    return Z*r_e*Utility_1(x,A,B)*Utility_2(x,A,B)*Utility_3(x,A,B)


plt.title('klein-nishina cross section')
xplot = np.linspace((5/11)*energy,energy,10000)
plt.plot(xplot,kleinnishina(xplot,1,0,1)/np.
 ↪amax(kleinnishina(xplot,1,0,1)),'g-',label='Normalized Klein Nishina Cross␣
 ↪Section')
plt.legend()
plt.xlabel('Residual Energy [KeV]')
plt.ylabel('Normalized Cross Section [u.a.]')
```

[307]:
```
Text(0, 0.5, 'Normalized Cross Section [u.a.]')
```



1.4.2 BGO1 - ^{18}F

[308]:
```
# BGO1 - F18

# Entries, Media, Standard deviation, Errore sulla media
entries    = len(BGO1F18)
mean, std  = np.mean(BGO1F18), np.std(BGO1F18)
mean_error = std/np.sqrt(len(BGO1F18))

# Creazione dell'istogramma (con barre di errore)
fig                            = plt.figure(1, figsize = (10,6))
bin_heights, bin_borders, _ = plt.
↪hist(BGO1F18,bins=200,facecolor='g',ec='black',alpha=0.5 ,density=False) #␣
↪label='histogram data'
bin_centers                    = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                  = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
↪ecolor='black') # label='error'


# FIT GAUSSIANO
# Definisco gli estremi di intervallo su cui eseguire il fit gaussiano
a = 2.9
```

```
b = 4.7
# Creazione dei vettori per il fit sul fotopicco => elimino la parte che non␣
↪interessa
bin_centers_new = []
bin_heights_new = []

for i in range(len(bin_heights)):
    if bin_centers[i] < b and bin_centers[i] > a:
        bin_centers_new.append(bin_centers[i])
        bin_heights_new.append(bin_heights[i])
    else:
        pass

bin_centers_new   = np.array(bin_centers_new)
bin_heights_new   = np.array(bin_heights_new)
sigma_heights_new = np.sqrt(bin_heights_new)

#=================================================
# Funzione di fit
#=================================================
def fit_gauss(x, A, mu, sigma):
    return A*np.exp(-(x-mu)**2/(2*sigma**2))

# Parametri iniziali
param0     = [entries, mean, std]
# Best Parameters
popt, pcov = curve_fit(fit_gauss, bin_centers_new, bin_heights_new, param0,␣
↪sigma = sigma_heights_new)

# Best-Parameters
A           = popt[0]
sigma_A     = np.sqrt(pcov[0,0])
mu          = popt[1]
sigma_mu    = np.sqrt(pcov[1,1])
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
x = np.linspace(min(bin_centers),max(bin_centers),500)
#x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss(x, *popt), '-', color="blue", label='Gauss')


#=================================================
# CHI2 TEST
```

```
#=================================================
chi2    = sum(((bin_heights_new - fit_gauss(bin_centers_new, *popt)) /␣
 ↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2*np.sqrt(2*np.log(2))*sigma
sigma_FWHM = 2*np.sqrt(2*np.log(2))*sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error␣
 ↪propagation



# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('BGO1 - F18')
plt.xlim(-0.1, 6.5)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
 ↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$   ␣
 ↪(%.2f $\pm$ %.2f)'          %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.
 ↪2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
 ↪label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
plt.legend(frameon=False ,fancybox=True, shadow=False, loc='best', prop={"size":
 ↪12}, numpoints = 1)

#plt.savefig('FIGURE/BGO1F18_gauss.pdf', format='pdf',bbox_inches="tight",␣
 ↪dpi=100)
#plt.show()



#=================================================
# STAMPA RISULTATI DEL FIT
#=================================================
print('=================================================\n')
print('Entries   %.i'   %entries)
```

```python
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#---------------------------------\n')
print('# Fit Gaussiano:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('\n---------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n---------------------------------\n')
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n=================================================')

#=================================================
# Funzione di fit
#=================================================
def fit_gauss_linear(x, A, mu, sigma, a, b):
    fotopicco = A*np.exp(-(x-mu)**2/(2*sigma**2))   # Modello gaussiano per il fotopicco
    fondo     = a * x + b                           # Modello lineare per il fondo
    return fotopicco + fondo

# Parametri iniziali
#param0     = [entries, (a+b)/2, std]
# Best Parameters
#popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new, param0, sigma = sigma_heights_new)
popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new, sigma = sigma_heights_new)
# Best-Parameters
A           = popt[0]
sigma_A     = np.sqrt(pcov[0,0])
mu          = popt[1]
sigma_mu    = np.sqrt(pcov[1,1])
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])
a           = popt[3]
sigma_a     = np.sqrt(pcov[3,3])
b           = popt[4]
```

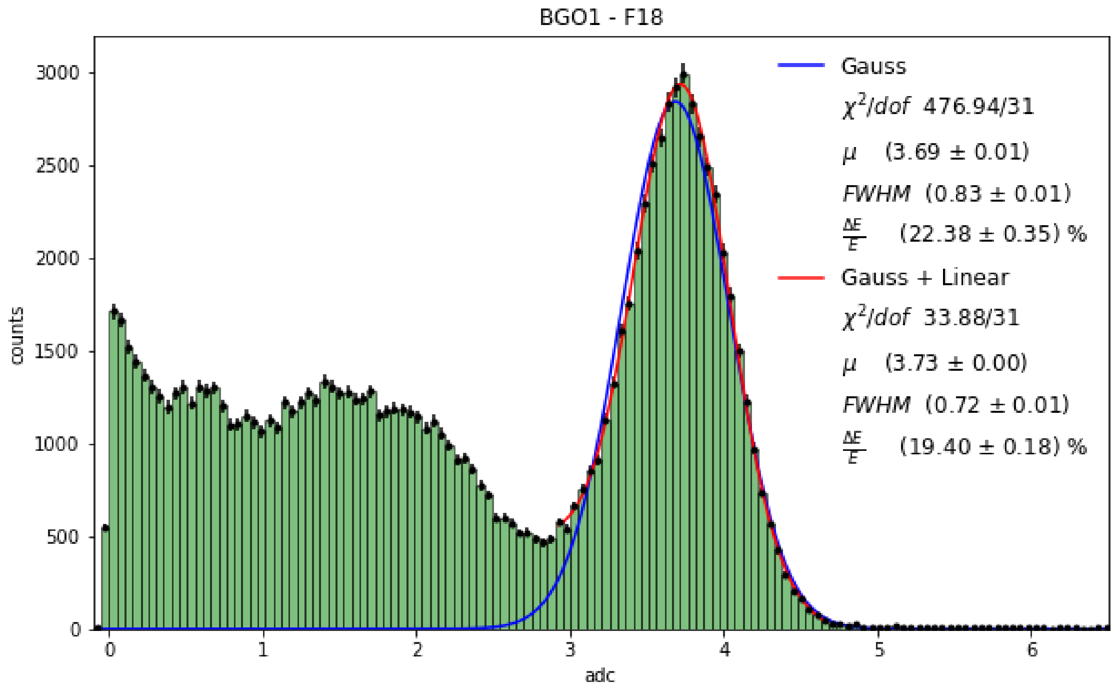
```
sigma_b     = np.sqrt(pcov[4,4])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
#x = np.linspace(min(bin_centers),max(bin_centers),500)
x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss_linear(x, *popt), '-', color="red", label='Gauss +␣
↪Linear')



#==================================================
# CHI2 TEST
#==================================================
chi2    = sum(((bin_heights_new - fit_gauss_linear(bin_centers_new, *popt)) /␣
↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2*np.sqrt(2*np.log(2))*sigma
sigma_FWHM = 2*np.sqrt(2*np.log(2))*sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error␣
↪propagation



# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('BGO1 - F18')
plt.xlim(-0.1, 6.5)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$   ␣
↪(%.2f $\pm$ %.2f)'          %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.
↪2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None',␣
↪label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
```

```
plt.legend(frameon=False, fancybox=True, shadow=False, loc='best', prop={"size":
 ↪12}, numpoints = 1)

#===============================================
# STAMPA RISULTATI DEL FIT
#===============================================
print('================================================\n')
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#--------------------------------\n')
print('# Fit Gaussiano + Lineare:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('a        (%.3f ± %.3f)' %(popt[3],np.sqrt(pcov[3,3])))
print('b        (%.3f ± %.3f)' %(popt[4],np.sqrt(pcov[4,4])))
print('\n--------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n--------------------------------\n')
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n================================================')

plt.savefig('FIGURE/BGO1F18_gauss_gauss_linear.pdf',␣
 ↪format='pdf',bbox_inches="tight", dpi=100)
plt.show()
```

```
================================================

Entries   112450
Mean      2.328
Std Dev   1.374

#--------------------------------

# Fit Gaussiano:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)

# Parametri del fit:
A        (2840.809 ± 63.610)
```

```
mu       (3.686 ± 0.007)
sigma    (0.350 ± 0.005)

---------------------------------

# Chi square test:
Chi2      476.940
dof       31
Chi2/dof  15.385
pvalue    0.000

---------------------------------

FWHM      (0.825 ± 0.013)
R         (22.383 ± 0.348)%

=================================================
=================================================

Entries   112450
Mean      2.328
Std Dev   1.374

#--------------------------------

# Fit Gaussiano + Lineare:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b

# Parametri del fit:
A        (2672.169 ± 20.278)
mu       (3.728 ± 0.003)
sigma    (0.307 ± 0.003)
a        (-263.079 ± 11.188)
b        (1240.796 ± 52.326)

---------------------------------

# Chi square test:
Chi2      33.885
dof       31
Chi2/dof  1.093
pvalue    0.330

---------------------------------

FWHM      (0.723 ± 0.007)
R         (19.397 ± 0.179)%
```

==



1.5 MODELLO: GAUSS + CODE EXPO

[325]:
```
# BGO1 - F18

# Entries, Media, Standard deviation, Errore sulla media
entries    = len(BGO1F18)
mean, std  = np.mean(BGO1F18), np.std(BGO1F18)
mean_error = std/np.sqrt(len(BGO1F18))

# Creazione dell'istogramma (con barre di errore)
fig                             = plt.figure(1, figsize = (10,6))
bin_heights, bin_borders, _ = plt.
 ↪hist(BGO1F18,bins=200,facecolor='g',ec='black',alpha=0.5 ,density=False) #␣
 ↪label='histogram data'
bin_centers                     = 1/2 * (bin_borders[1:] + bin_borders[:-1])
sigma_heights                   = np.sqrt(bin_heights) # Errore Poissoniano
plt.errorbar(bin_centers, bin_heights, sigma_heights, fmt='.', color='black',␣
 ↪ecolor='black') # label='error'


# FIT GAUSSIANO
# Definisco gli estremi di intervallo su cui eseguire il fit gaussiano
```

```
a = 2.9
b = 4.7
# Creazione dei vettori per il fit sul fotopicco => elimino la parte che non␣
↪interessa
bin_centers_new = []
bin_heights_new = []

for i in range(len(bin_heights)):
    if bin_centers[i] < b and bin_centers[i] > a:
        bin_centers_new.append(bin_centers[i])
        bin_heights_new.append(bin_heights[i])
    else:
        pass

bin_centers_new   = np.array(bin_centers_new)
bin_heights_new   = np.array(bin_heights_new)
sigma_heights_new = np.sqrt(bin_heights_new)

#==================================================
# Funzione di fit
#==================================================
def fit_gauss(x, A, mu, sigma):
    return A*np.exp(-(x-mu)**2/(2*sigma**2))

# Parametri iniziali
param0     = [entries, mean, std]
# Best Parameters
popt, pcov = curve_fit(fit_gauss, bin_centers_new, bin_heights_new, param0,␣
↪sigma = sigma_heights_new)

# Best-Parameters
A           = popt[0]
sigma_A     = np.sqrt(pcov[0,0])
mu          = popt[1]
sigma_mu    = np.sqrt(pcov[1,1])
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])

# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
x = np.linspace(min(bin_centers),max(bin_centers),500)
#x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss(x, *popt), '-', color="blue", label='Gauss')


#==================================================
```

```
# CHI2 TEST
#================================================
chi2    = sum(((bin_heights_new - fit_gauss(bin_centers_new, *popt)) /
 ↪sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2*np.sqrt(2*np.log(2))*sigma
sigma_FWHM = 2*np.sqrt(2*np.log(2))*sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error
 ↪propagation


# Bellurie
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('BGO1 - F18')
plt.xlim(-0.1, 6.5)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/
 ↪$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$
 ↪(%.2f $\pm$ %.2f)'          %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.
 ↪2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None',
 ↪label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
plt.legend(frameon=False ,fancybox=True, shadow=False, loc='best', prop={"size":
 ↪12}, numpoints = 1)

#plt.savefig('FIGURE/BGO1F18_gauss.pdf', format='pdf',bbox_inches="tight",
 ↪dpi=100)
#plt.show()


#================================================
# STAMPA RISULTATI DEL FIT
#================================================
print('================================================\n')
```

```
print('Entries   %.i'   %entries)
print('Mean      %.3f'  %mean)
print('Std Dev   %.3f'  %std)
print('\n#--------------------------------\n')
print('# Fit Gaussiano:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('\n--------------------------------\n')
print('# Chi square test:')
print('Chi2      %.3f' %chi2)
print('dof       %.i'  %dof)
print('Chi2/dof  %.3f' %chi2_rid)
print('pvalue    %.3f' %pvalue)
print('\n--------------------------------\n')
print('FWHM      (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R         (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n================================================')

#================================================
# Funzione di fit
#================================================
def fit_gauss_linear_expo(x, A, mu, sigma, a, b, c, d, e,f,g,h):
    fotopicco = A*np.exp(-(x-mu)**2/(2*sigma**2))   # Modello gaussiano per il fotopicco
    fondo     = a * x + b                           # Modello lineare per il fondo
    coda1     = c * np.exp(d*x) + e                # Modello esponenziale per le code del fotopicco
    coda2     = f * np.exp(g*x) + h                # Modello esponenziale per le code del fotopicco
    return fotopicco + fondo + coda1 + coda2

# Parametri iniziali
param0     = [entries, (a+b)/2, std,0,1,1,1,1]
# Best Parameters
#popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new, bin_heights_new, param0, sigma = sigma_heights_new)
popt, pcov = curve_fit(fit_gauss_linear_expo, bin_centers_new, bin_heights_new, sigma = sigma_heights_new)
# Best-Parameters
A           = popt[0]
sigma_A     = np.sqrt(pcov[0,0])
mu          = popt[1]
```

```
sigma_mu    = np.sqrt(pcov[1,1])
sigma       = popt[2]
sigma_sigma = np.sqrt(pcov[2,2])
a           = popt[3]
sigma_a     = np.sqrt(pcov[3,3])
b           = popt[4]
sigma_b     = np.sqrt(pcov[4,4])
c           = popt[5]
sigma_c     = np.sqrt(pcov[5,5])
d           = popt[6]
sigma_d     = np.sqrt(pcov[6,6])
e           = popt[7]
sigma_e     = np.sqrt(pcov[7,7])


# Definizione vettore delle x (asse x di estremi minimo e massimo di x_data)
#x = np.linspace(min(bin_centers),max(bin_centers),500)
x = np.linspace(min(bin_centers_new),max(bin_centers_new),100)

# Plot fit
plt.plot(x, fit_gauss_linear_expo(x, *popt), '-', color="red", label='Gauss +␣
 ↪Linear + Expo')


#=================================================
# CHI2 TEST
#=================================================
chi2    = sum(((bin_heights_new - fit_gauss_linear_expo(bin_centers_new,␣
 ↪*popt)) / sigma_heights_new)**2)
# Numero di gradi di libertà
dof     = len(bin_centers_new) - len(param0) - 1 # Sottraggo 1 perché N costante
# Calcolo dei chi2 ridotto
chi2_rid = chi2/dof
# Calcolo del p-value
pvalue = 1 - stats.chi2.cdf(chi2, dof) # pvalue deve essere maggiore di 0.05

# Full Width at Half Maximum
FWHM       = 2*np.sqrt(2*np.log(2))*sigma
sigma_FWHM = 2*np.sqrt(2*np.log(2))*sigma_sigma

# Risuoluzione
R       = (FWHM/mu) * 100      # %Energy resolution = FWHM x 100 /photo peak
sigma_R = R * (np.sqrt((sigma_FWHM/FWHM)**2 + (sigma_mu/mu)**2))  # error␣
 ↪propagation


# Bellurie
```

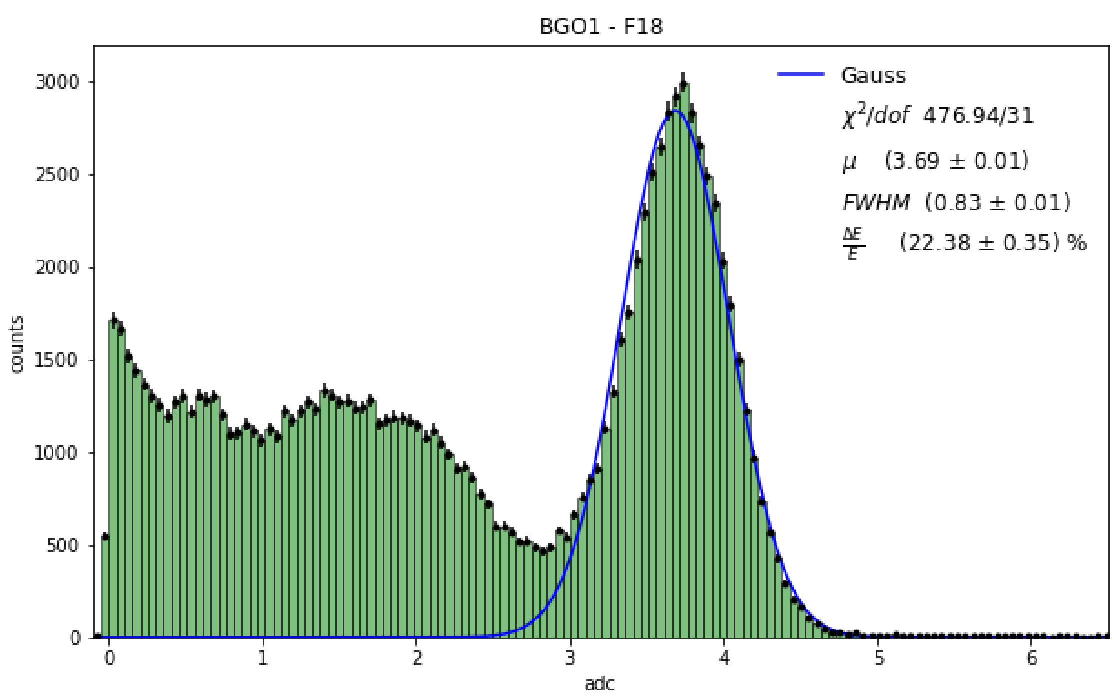
```python
plt.xlabel('adc')
plt.ylabel('counts')
plt.title('BGO1 - F18')
plt.xlim(-0.1, 6.5)
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\chi^2$/$dof$  %.2f/%.i'      %(chi2,dof))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\mu$    (%.2f $\pm$ %.2f)'           %(mu, sigma_mu))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$FWHM$  (%.2f $\pm$ %.2f)'    %(FWHM, sigma_FWHM))
plt.plot([], [], color='white', marker='.',linestyle='None', label=r'$\frac{\Delta E}{E}$     (%.2f $\pm$ %.2f) %%'       %(R, sigma_R))
plt.legend(frameon=False, fancybox=True, shadow=False, loc='best', prop={"size":12}, numpoints = 1)

#=================================================
# STAMPA RISULTATI DEL FIT
#=================================================
print('=====================================================\n')
print('Entries    %.i'    %entries)
print('Mean       %.3f'   %mean)
print('Std Dev    %.3f'   %std)
print('\n#-------------------------------\n')
print('# Fit Gaussiano + Lineare:')
print('p(x) = A * exp(-(x-mu)^2/(2 * sigma^2) + a * x + b + c * exp(d*x) + e\n')
print('# Parametri del fit:')
print('A        (%.3f ± %.3f)' %(popt[0],np.sqrt(pcov[0,0])))
print('mu       (%.3f ± %.3f)' %(popt[1],np.sqrt(pcov[1,1])))
print('sigma    (%.3f ± %.3f)' %(popt[2],np.sqrt(pcov[2,2])))
print('a        (%.3f ± %.3f)' %(popt[3],np.sqrt(pcov[3,3])))
print('b        (%.3f ± %.3f)' %(popt[4],np.sqrt(pcov[4,4])))
print('c        (%.3f ± %.3f)' %(popt[5],np.sqrt(pcov[5,5])))
print('d        (%.3f ± %.3f)' %(popt[6],np.sqrt(pcov[6,6])))
print('e        (%.3f ± %.3f)' %(popt[7],np.sqrt(pcov[7,7])))
print('\n-------------------------------\n')
print('# Chi square test:')
print('Chi2       %.3f' %chi2)
print('dof        %.i'  %dof)
print('Chi2/dof   %.3f' %chi2_rid)
print('pvalue     %.3f' %pvalue)
print('\n-------------------------------\n')
print('FWHM       (%.3f ± %.3f)'   %(FWHM,sigma_FWHM))
print('R          (%.3f ± %.3f)%%' %(R,sigma_R))
print('\n=================================================')

plt.savefig('FIGURE/BGO1F18_expo.pdf', format='pdf',bbox_inches="tight", dpi=100)
```

```
plt.show()
```

```
==================================================

Entries   112450
Mean      2.328
Std Dev   1.374

#--------------------------------

# Fit Gaussiano:
p(x) = A * exp(-(x-mu)^2/(2 * sigma^2)

# Parametri del fit:
A        (2840.809 ± 63.610)
mu       (3.686 ± 0.007)
sigma    (0.350 ± 0.005)

---------------------------------

# Chi square test:
Chi2      476.940
dof       31
Chi2/dof  15.385
pvalue    0.000

---------------------------------

FWHM      (0.825 ± 0.013)
R         (22.383 ± 0.348)%

==================================================
```

```
       ␣
 ↪----------------------------------------------------------------------------

        RuntimeError                              Traceback (most recent call␣
 ↪last)

        <ipython-input-325-b4720b539a53> in <module>
        134 # Best Parameters
        135 #popt, pcov = curve_fit(fit_gauss_linear, bin_centers_new,␣
 ↪bin_heights_new, param0, sigma = sigma_heights_new)
    --> 136 popt, pcov = curve_fit(fit_gauss_linear_expo, bin_centers_new,␣
 ↪bin_heights_new, sigma = sigma_heights_new)
        137 # Best-Parameters
        138 A           = popt[0]
```

```
/opt/anaconda3/lib/python3.7/site-packages/scipy/optimize/minpack.py in␣
↪curve_fit(f, xdata, ydata, p0, sigma, absolute_sigma, check_finite, bounds,␣
↪method, jac, **kwargs)
    787         cost = np.sum(infodict['fvec'] ** 2)
    788         if ier not in [1, 2, 3, 4]:
--> 789             raise RuntimeError("Optimal parameters not found: " +␣
↪errmsg)
    790     else:
    791         # Rename maxfev (leastsq) to max_nfev (least_squares), if␣
↪specified.


RuntimeError: Optimal parameters not found: Number of calls to function␣
↪has reached maxfev = 2400.
```



[ ]: