HPGe Detector Appendix

Alessandro Maraio & Samuel Morgan

Abstract

The objective of this project was to identify an unknown gamma ray source by using a high-purity germanium (HPGe) detector. We calibrated the detector by using three known sources, ^{22}Na, ^{60}Co & ^{133}Ba, and then measured the energy spectrum from the unknown source. We determined the unknown source to have contained a sample of ^{238}U at some point in its lifetime, due to the fact that we detected many emissions from elements in the ^{238}U decay chain, such as ^{214}Bi & ^{214}Pb. We were also tasked with identifying radioactive sources present in the environmental background and what decay chains they might belong to. By running the detector without a source present, we found that there were many peaks from ^{214}Pb & ^{214}Bi which would point to the ^{238}U decay chain, along with peaks from sources such as ^{40}K & ^{137}Cs. We believe that we carried out the experiment successfully, completing all objectives – and if we had more time on the experiment it would have been nice to investigate the detector properties in more detail and how this would affect the obtained spectrum, especially in the low-energy limit.

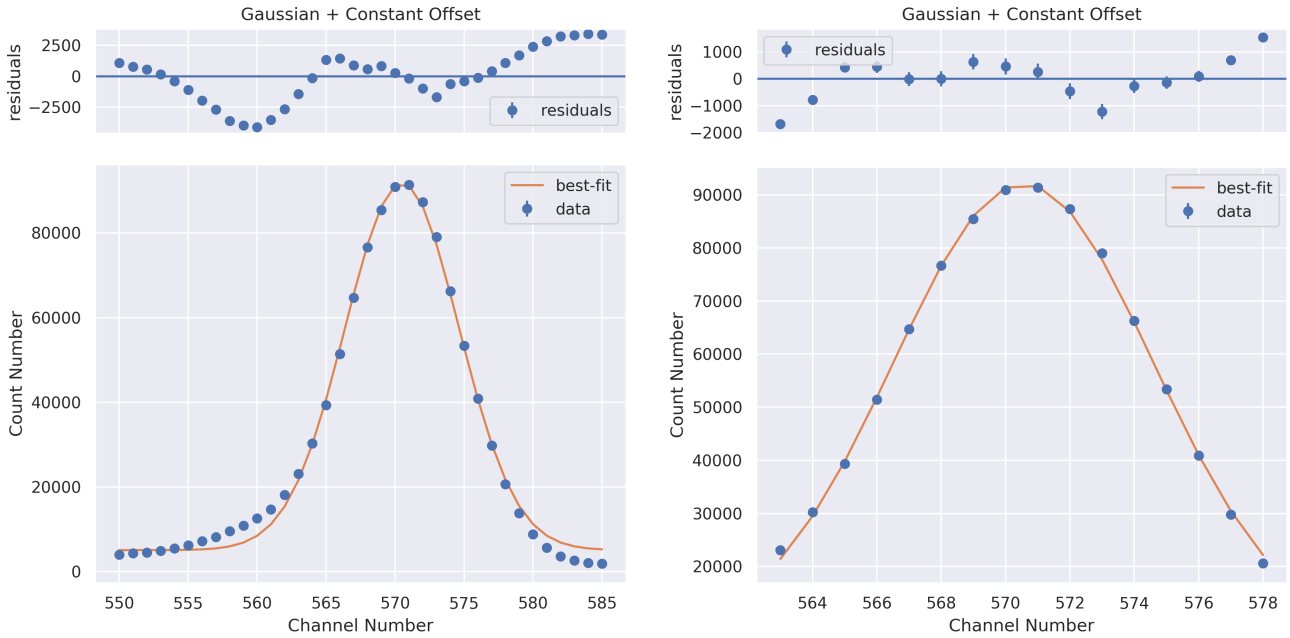
---

1 Fitted Peaks

All of these are for the 24 hour data runs, hence the high count numbers

1.1 Barium-133

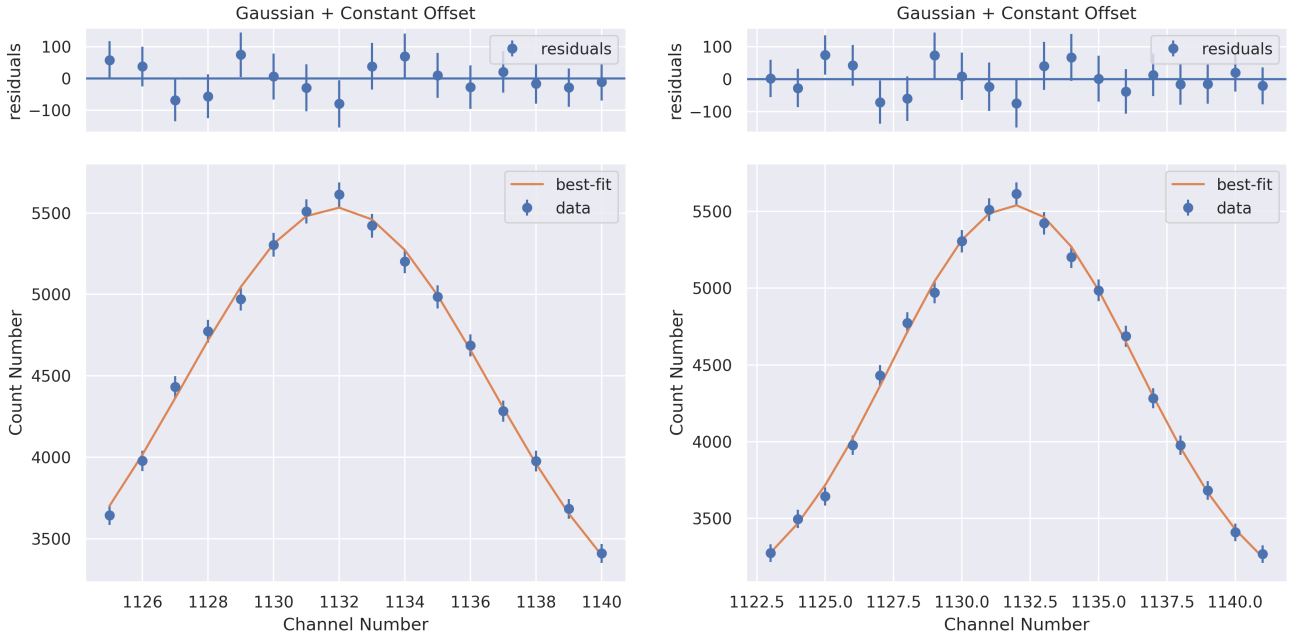
1.1.1 Gaussian Fit Only



(a) Full peak with fit

(b) Zoomed in peak with fit

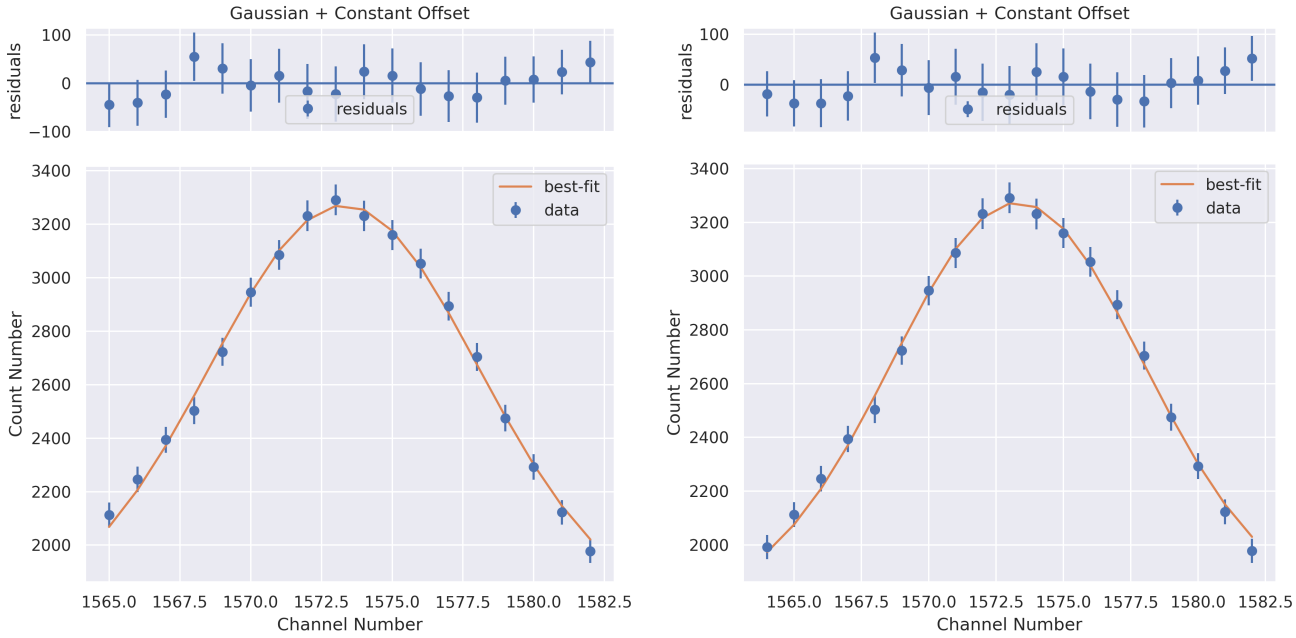
Figure 1: Fit of full & zoomed in peak of ^{133}Ba 81 keV peak



(a) Full peak with fit

(b) Zoomed in peak with fit

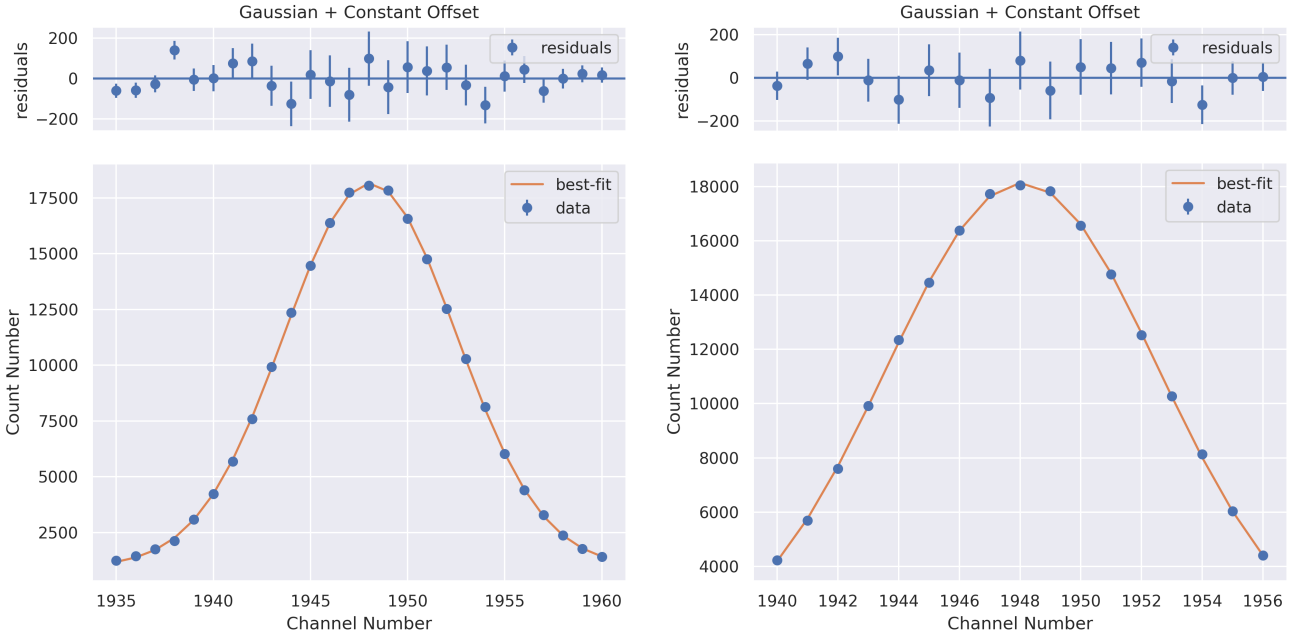
Figure 2: Fit of full & zoomed in peak of ^{133}Ba 161 keV peak



(a) Full peak with fit

(b) Zoomed in peak with fit

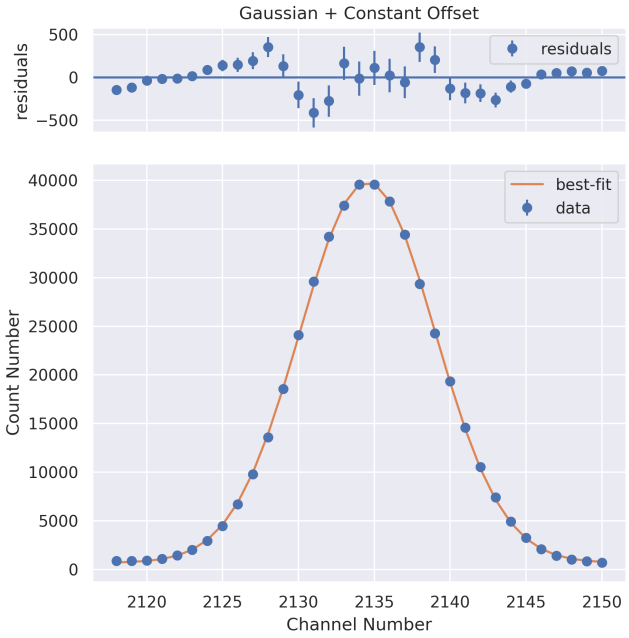
Figure 3: Fit of full & zoomed in peak of ^{133}Ba 223 keV peak



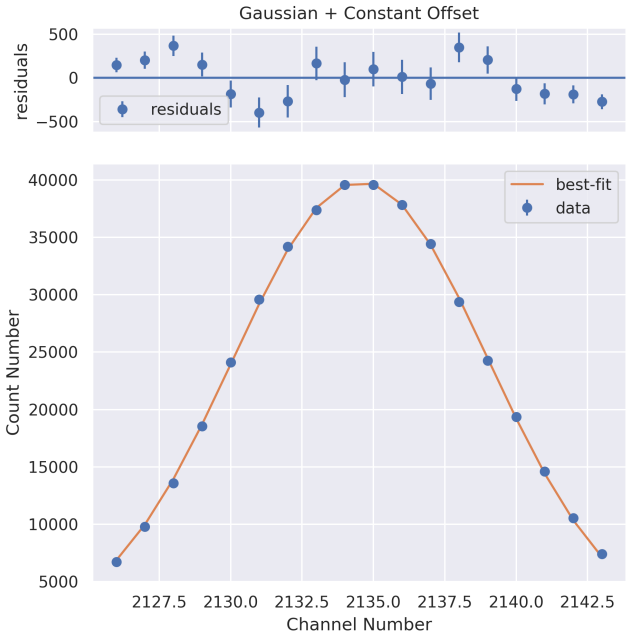
(a) Full peak with fit

(b) Zoomed in peak with fit

Figure 4: Fit of full & zoomed in peak of ^{133}Ba 276 keV peak

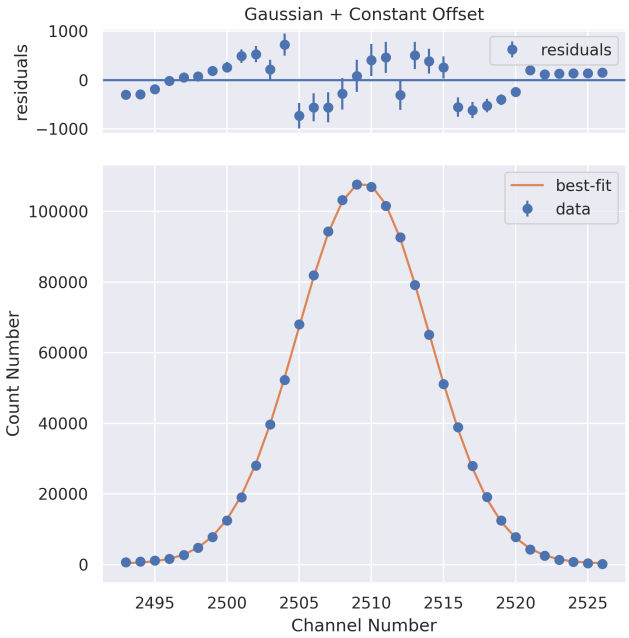


(a) Full peak with fit

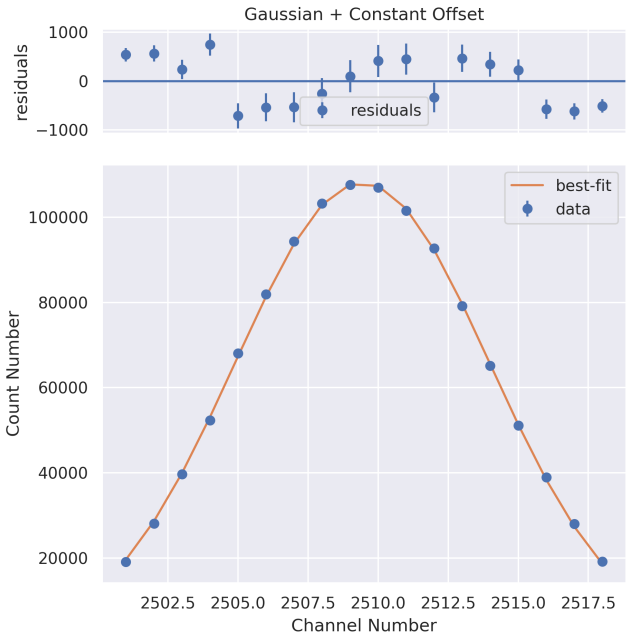


(b) Zoomed in peak with fit

Figure 5: Fit of full & zoomed in peak of ^{133}Ba 303 keV peak

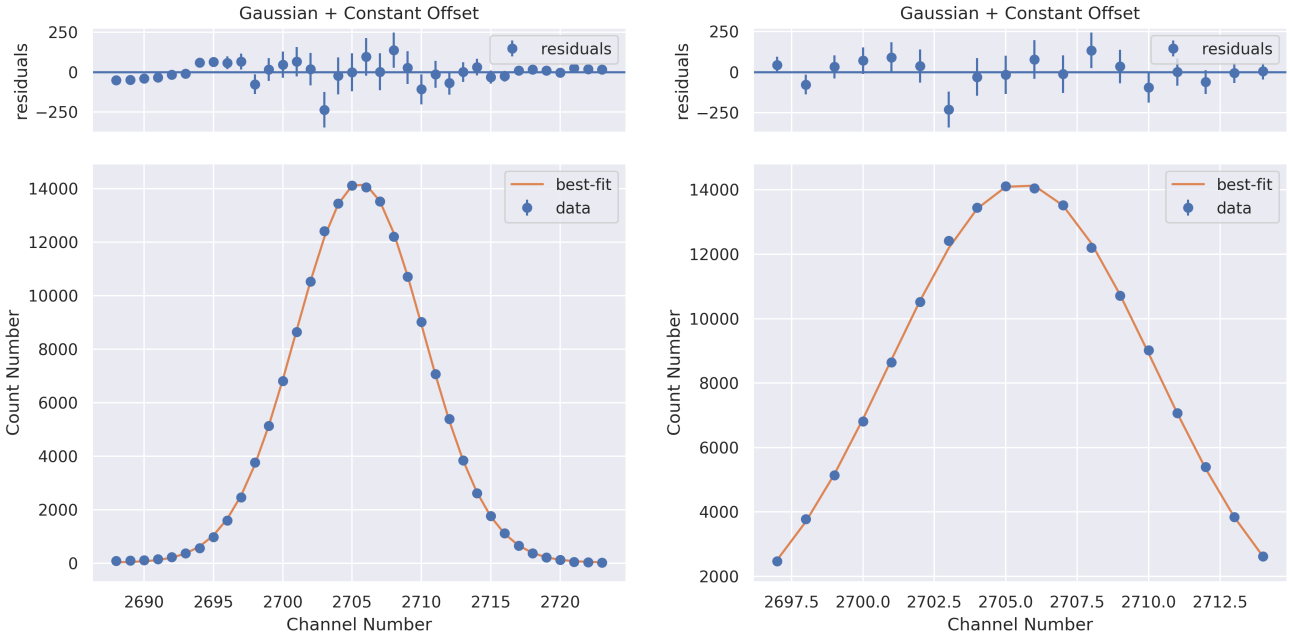


(a) Full peak with fit



(b) Zoomed in peak with fit

Figure 6: Fit of full & zoomed in peak of ^{133}Ba 356 keV peak

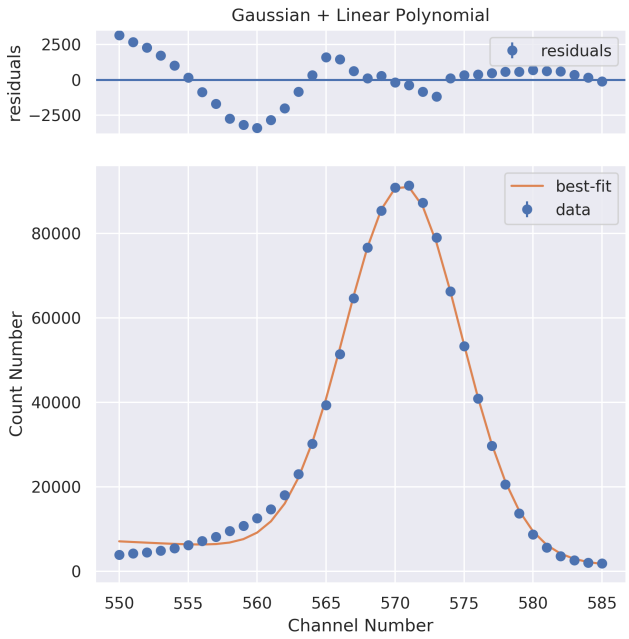


(a) Full peak with fit

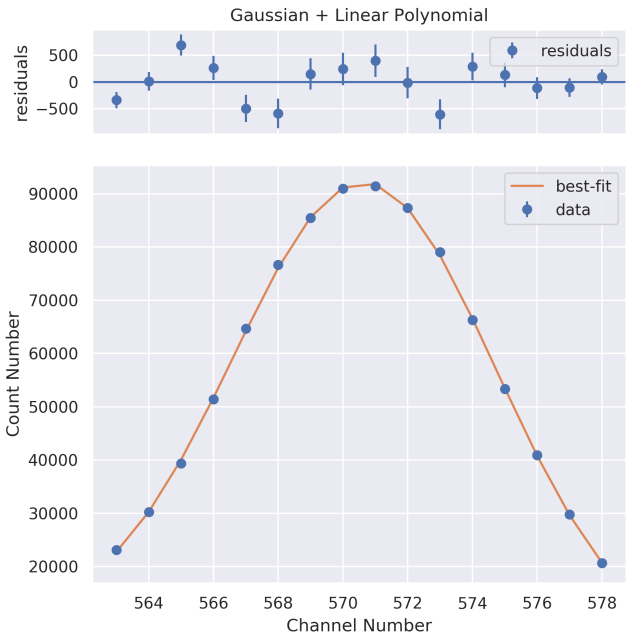
(b) Zoomed in peak with fit

Figure 7: Fit of full & zoomed in peak of ^{133}Ba 384 keV peak

1.1.2 Linear + Gaussian Fit

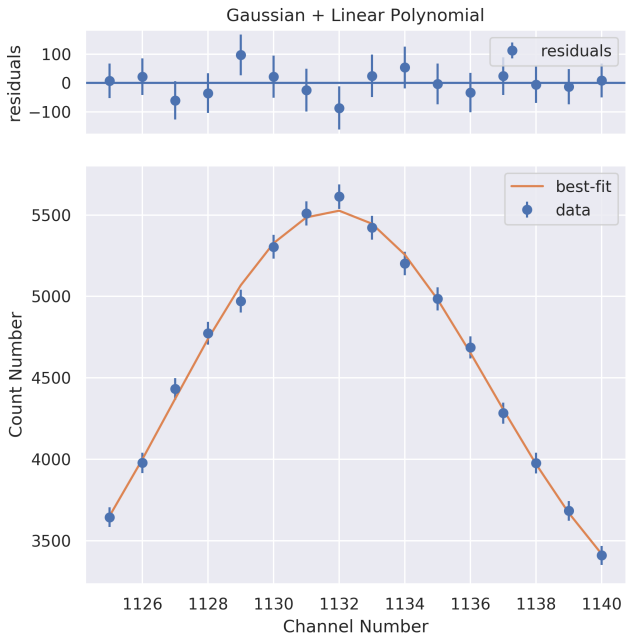


(a) Full peak with fit

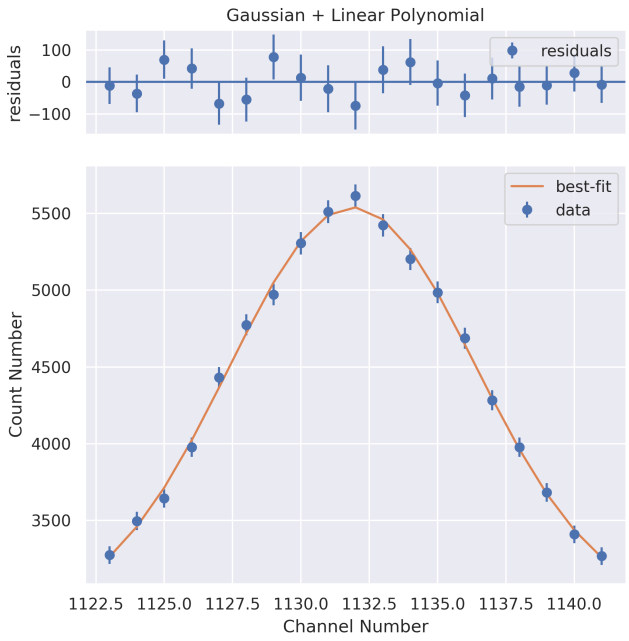


(b) Zoomed in peak with fit

Figure 8: Fit of full & zoomed in peak of ^{133}Ba 81 keV peak

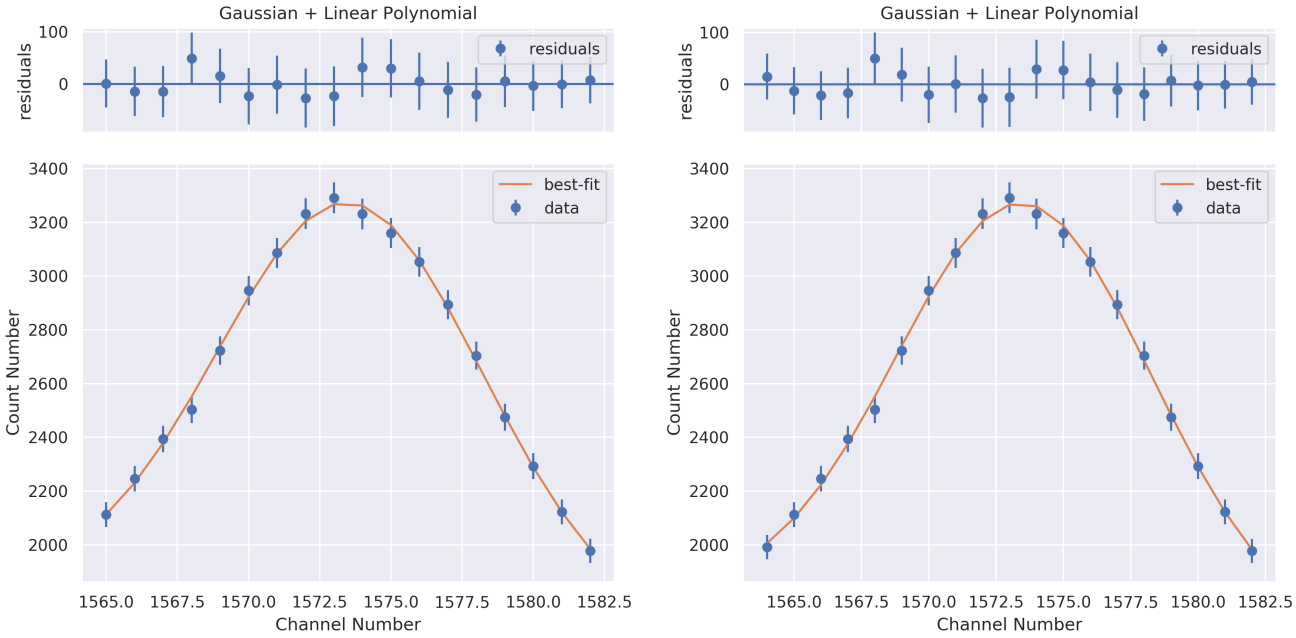


(a) Full peak with fit



(b) Zoomed in peak with fit

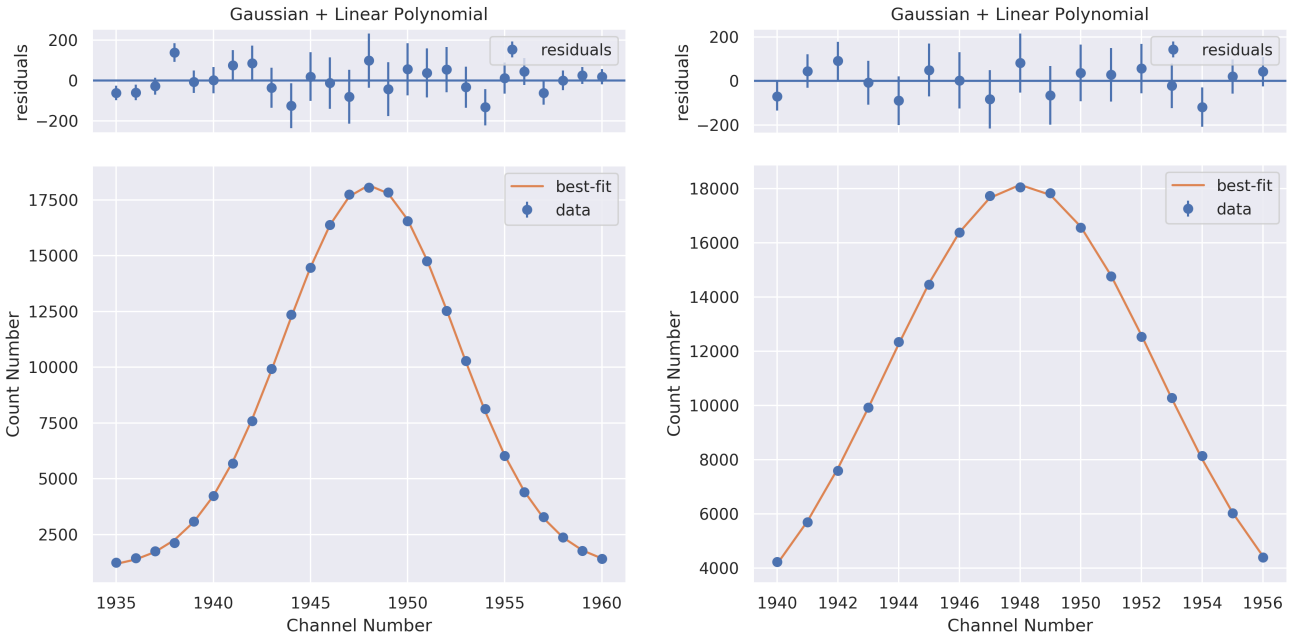
Figure 9: Fit of full & zoomed in peak of ^{133}Ba 161 keV peak



(a) Full peak with fit

(b) Zoomed in peak with fit

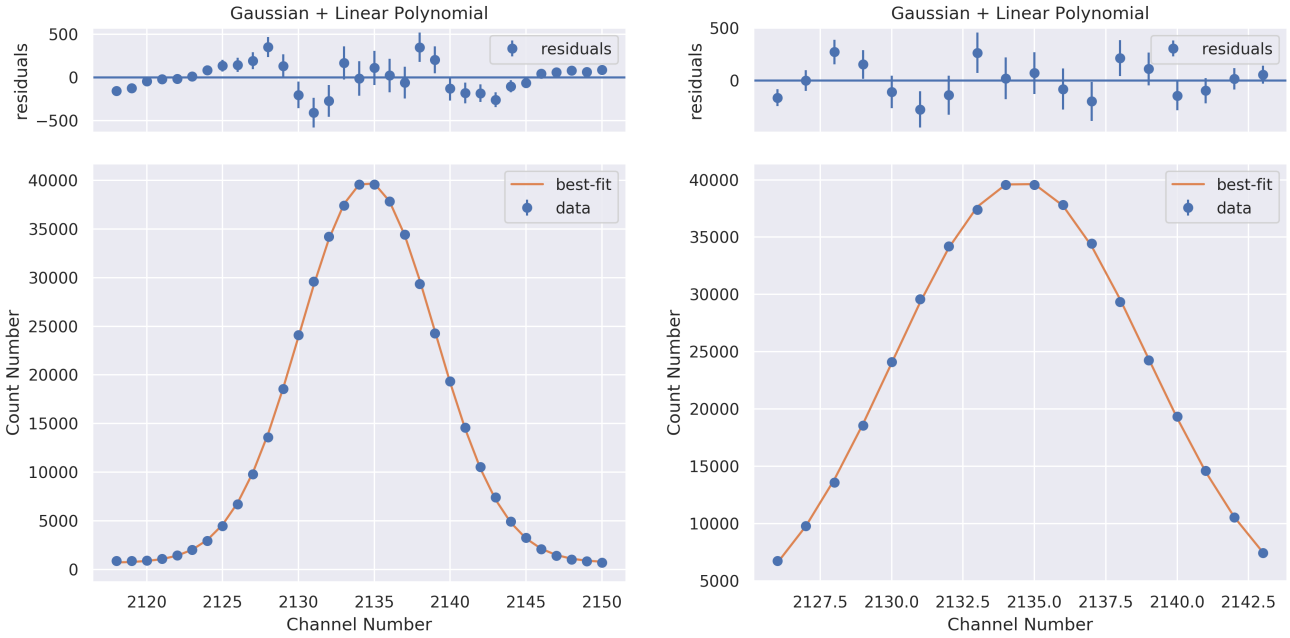
Figure 10: Fit of full & zoomed in peak of ^{133}Ba 223 keV peak



(a) Full peak with fit

(b) Zoomed in peak with fit

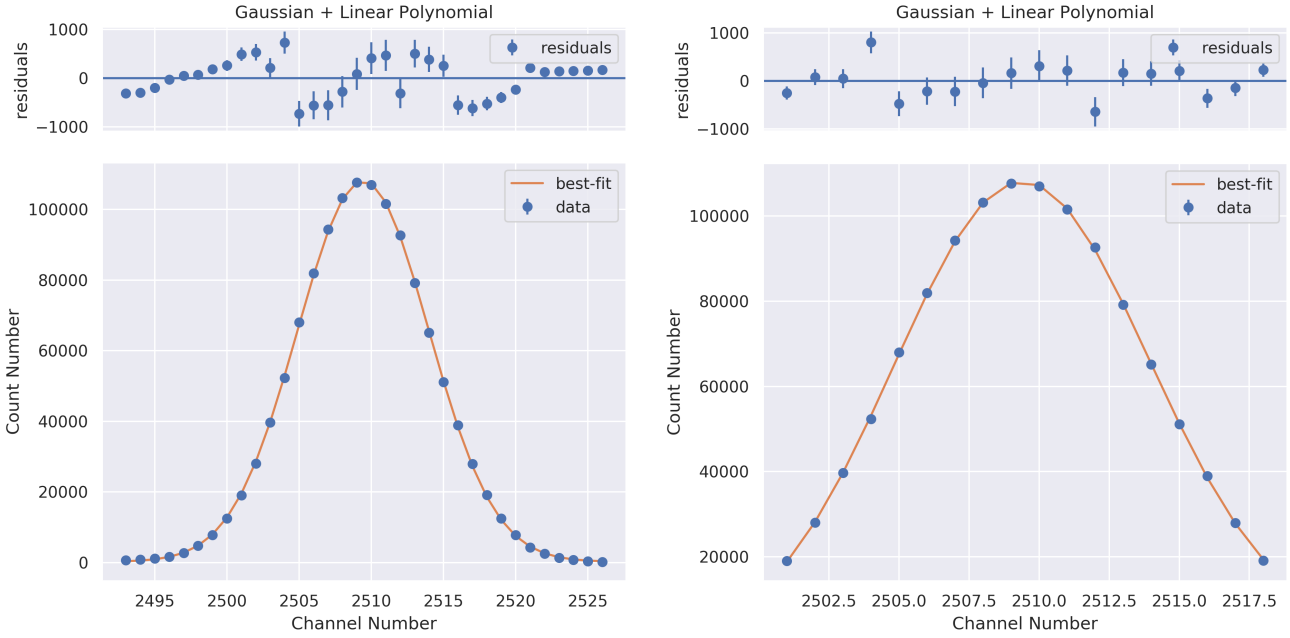
Figure 11: Fit of full & zoomed in peak of ^{133}Ba 276 keV peak



(a) Full peak with fit

(b) Zoomed in peak with fit

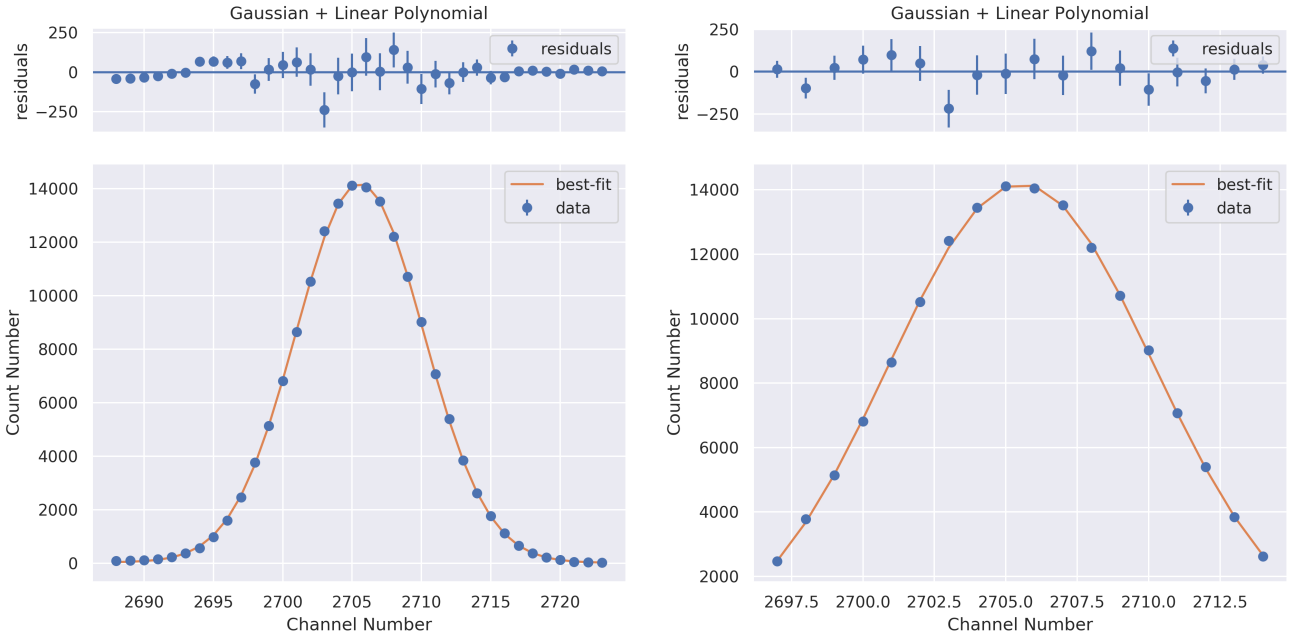
Figure 12: Fit of full & zoomed in peak of ^{133}Ba 303 keV peak



(a) Full peak with fit

(b) Zoomed in peak with fit

Figure 13: Fit of full & zoomed in peak of ^{133}Ba 356 keV peak

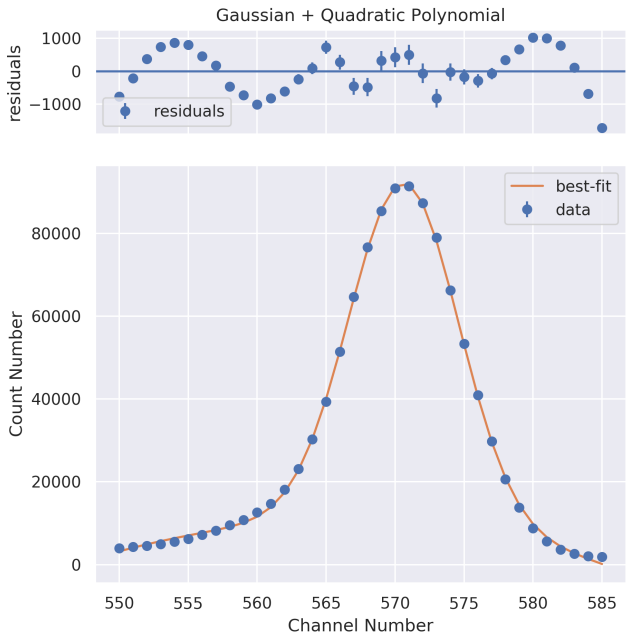


(a) Full peak with fit

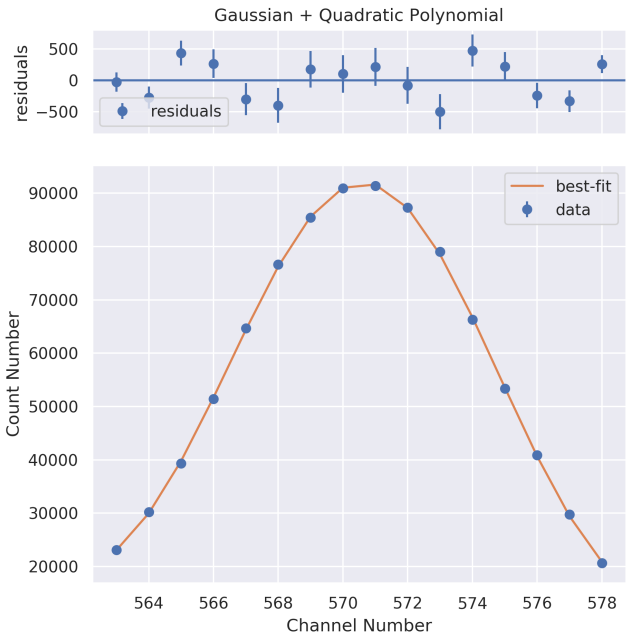
(b) Zoomed in peak with fit

Figure 14: Fit of full & zoomed in peak of ^{133}Ba 384 keV peak

1.1.3 Quadratic + Gaussian Fit

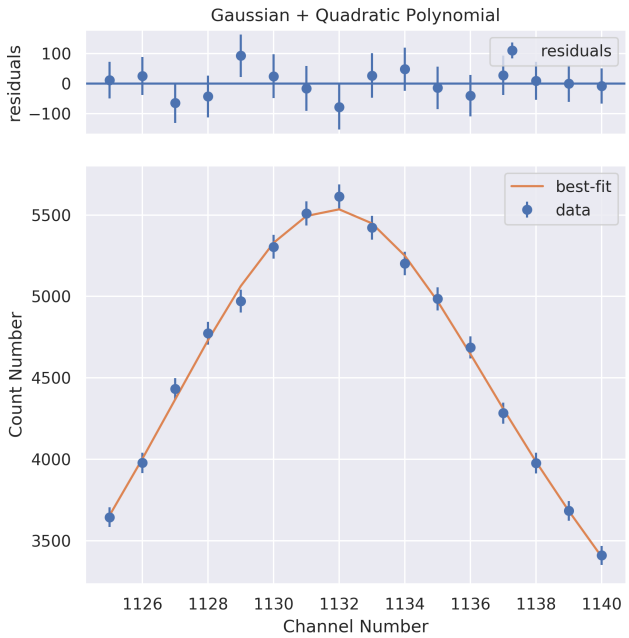


(a) Full peak with fit

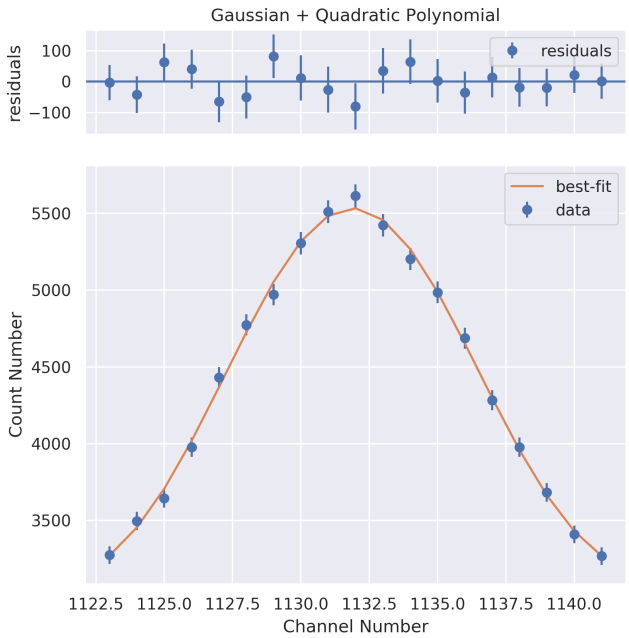


(b) Zoomed in peak with fit

Figure 15: Fit of full & zoomed in peak of ^{133}Ba 81 keV peak

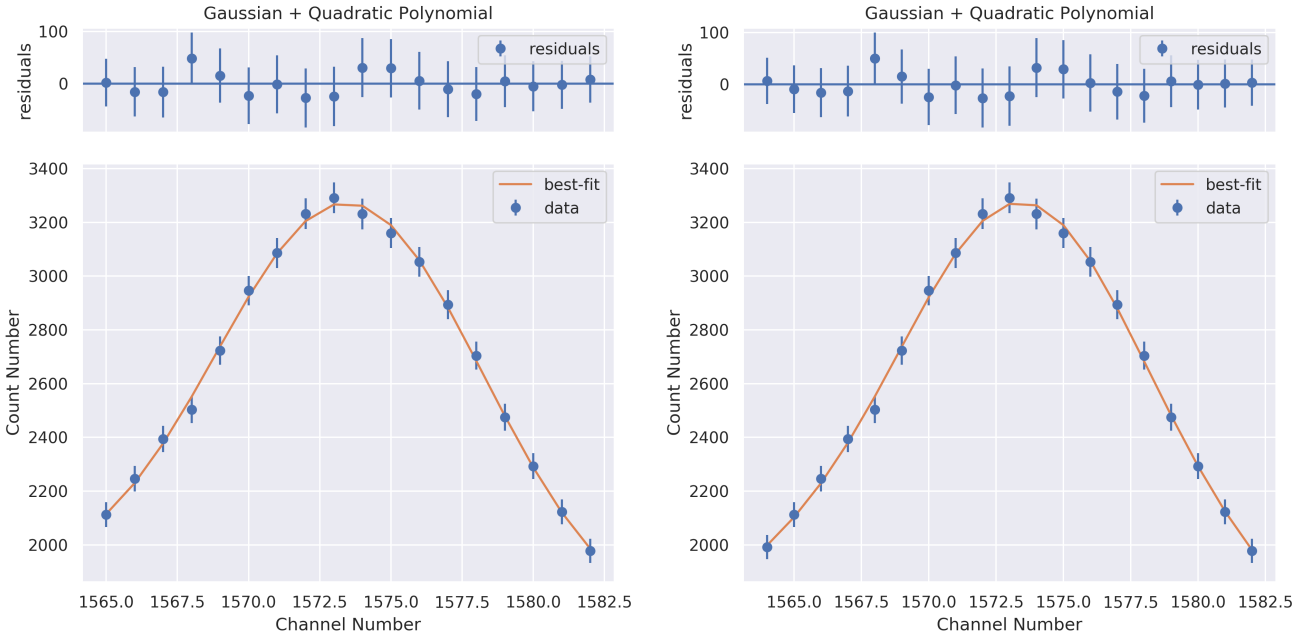


(a) Full peak with fit



(b) Zoomed in peak with fit

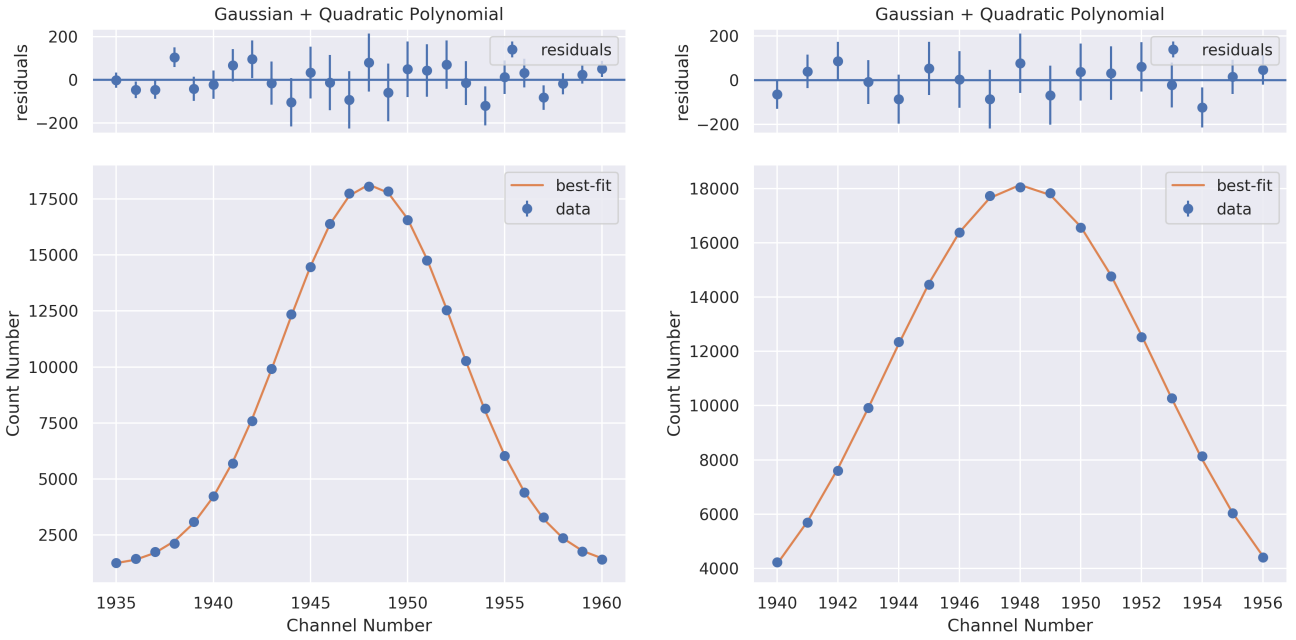
Figure 16: Fit of full & zoomed in peak of ^{133}Ba 161 keV peak



(a) Full peak with fit

(b) Zoomed in peak with fit

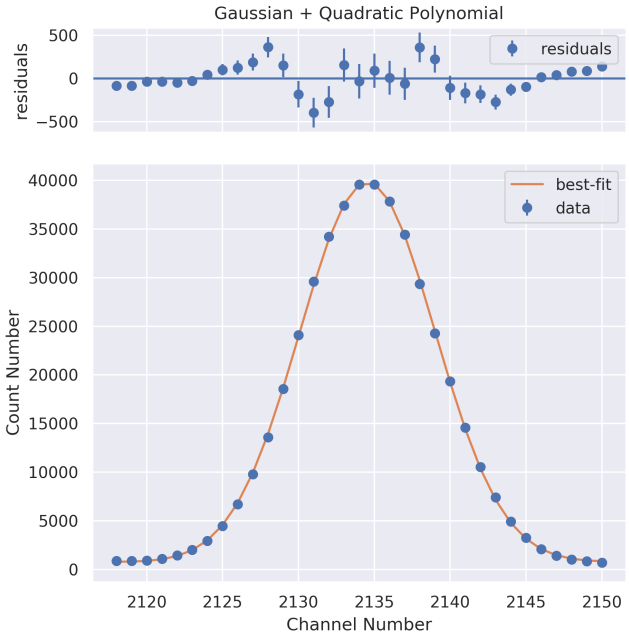
Figure 17: Fit of full & zoomed in peak of ^{133}Ba 223 keV peak



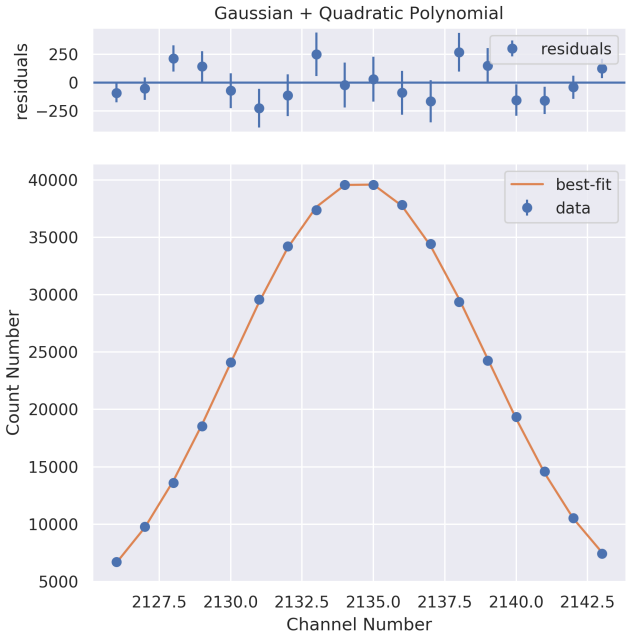
(a) Full peak with fit

(b) Zoomed in peak with fit

Figure 18: Fit of full & zoomed in peak of ^{133}Ba 276 keV peak

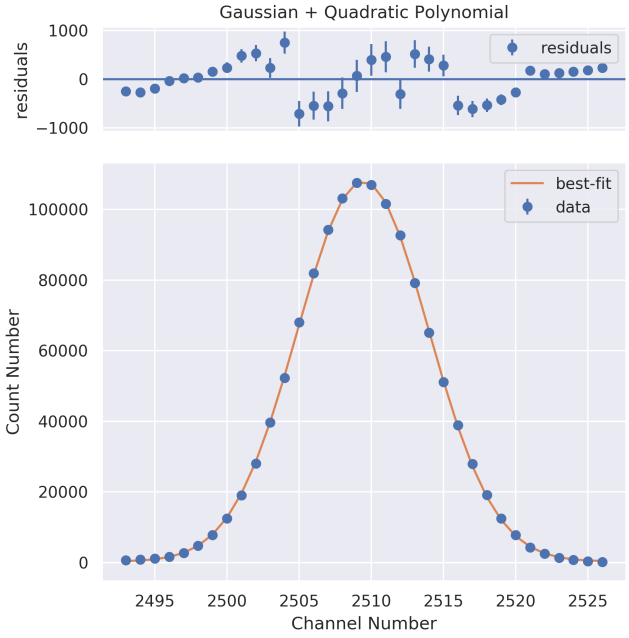


(a) Full peak with fit

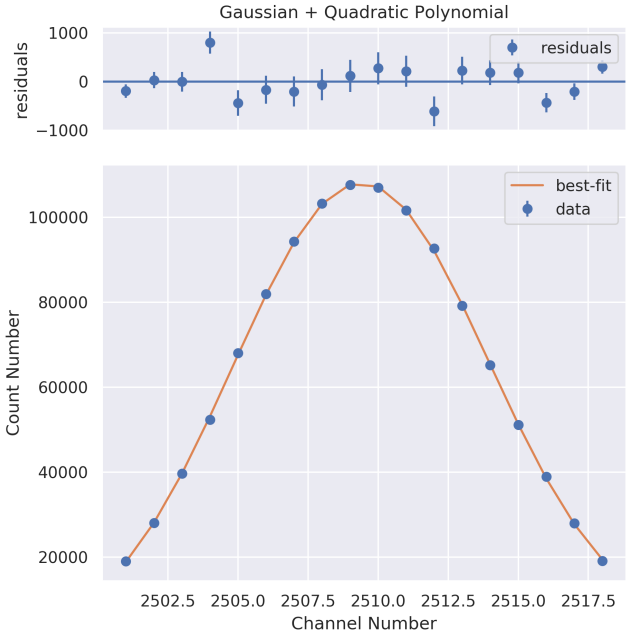


(b) Zoomed in peak with fit

Figure 19: Fit of full & zoomed in peak of ^{133}Ba 303 keV peak

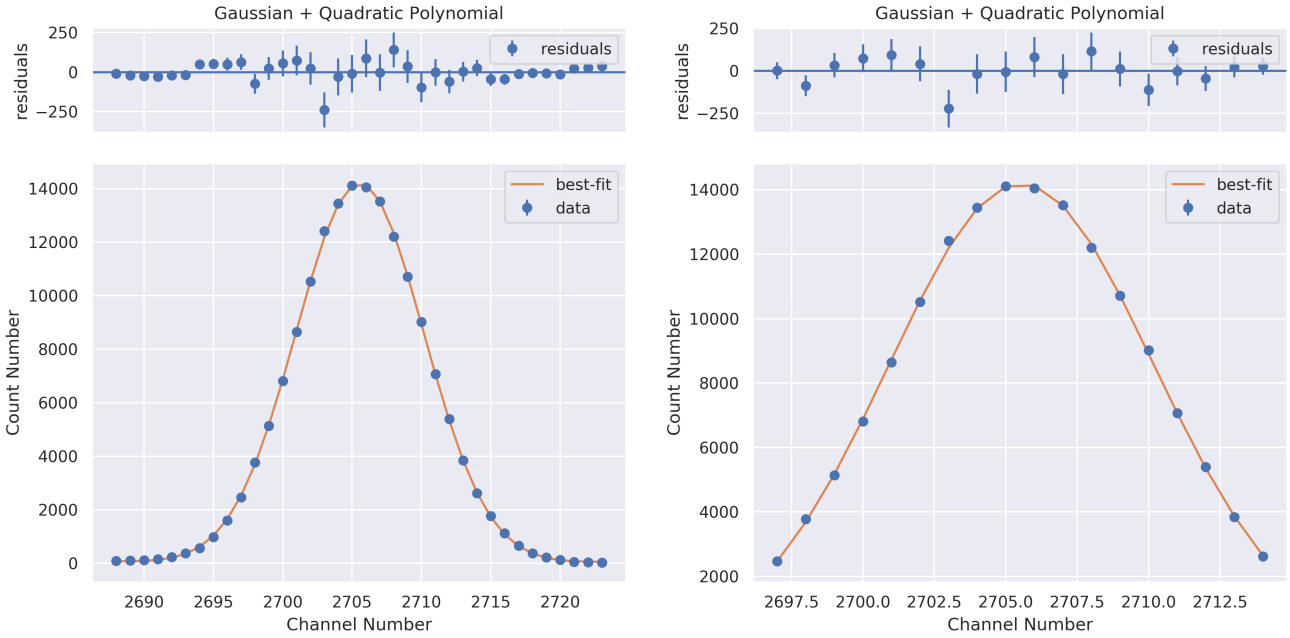


(a) Full peak with fit



(b) Zoomed in peak with fit

Figure 20: Fit of full & zoomed in peak of ^{133}Ba 356 keV peak



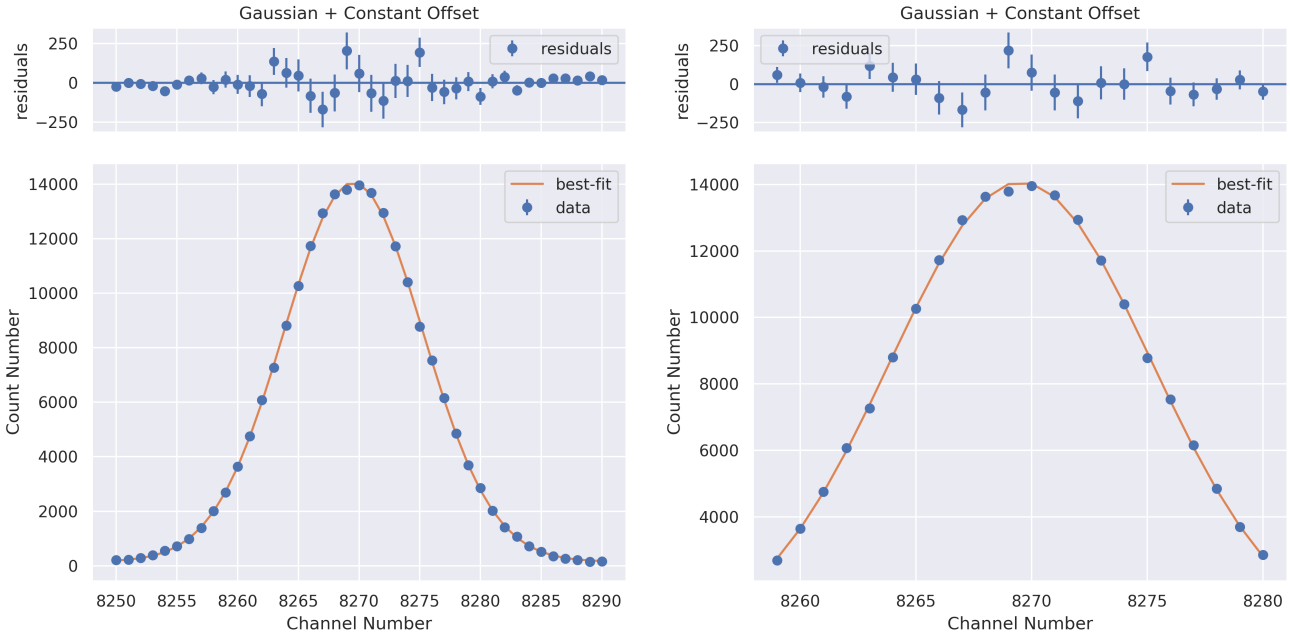
(a) Full peak with fit

(b) Zoomed in peak with fit

Figure 21: Fit of full & zoomed in peak of ^{133}Ba 384 keV peak

1.2 Cobalt-60

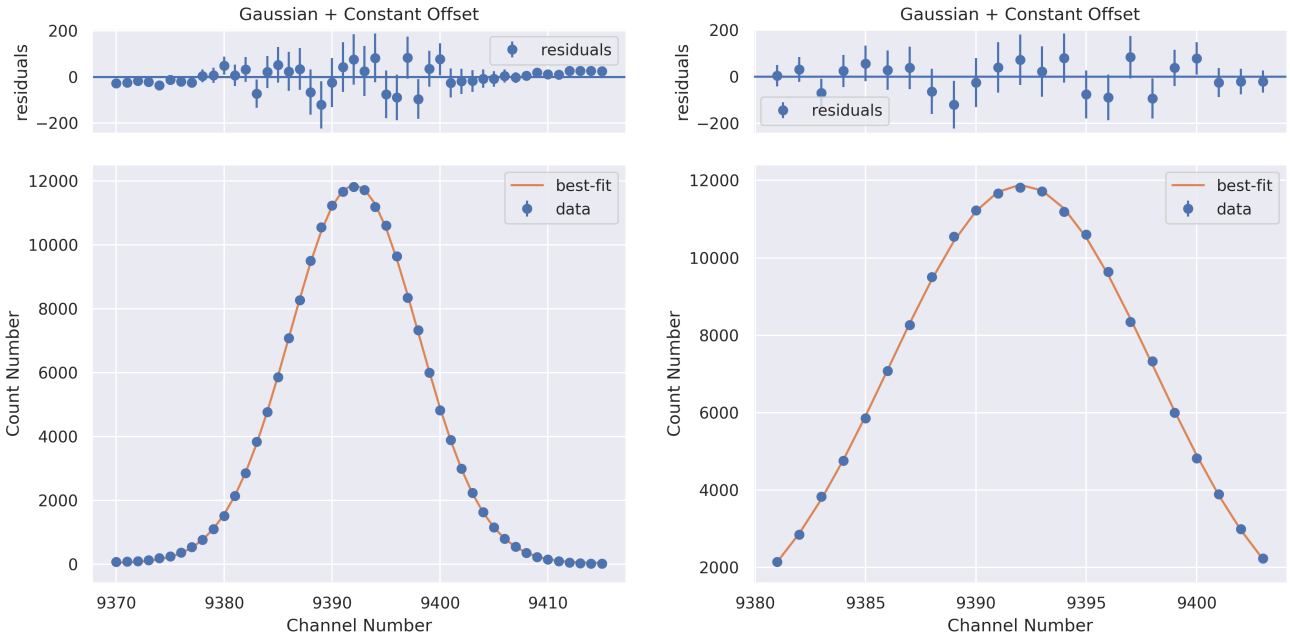
1.2.1 Gaussian Fit Only



(a) Full peak with fit

(b) Zoomed in peak with fit

Figure 22: Fit of full & zoomed in peak of ^{60}Co 1173 keV peak

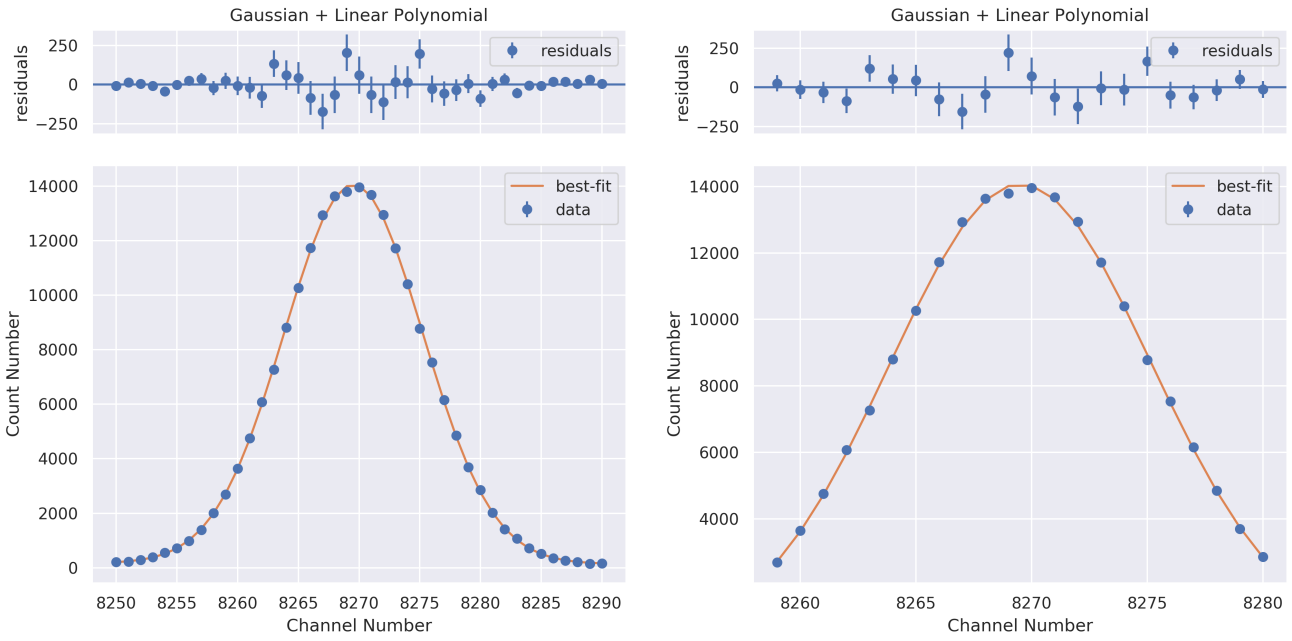


(a) Full peak with fit

(b) Zoomed in peak with fit

Figure 23: Fit of full & zoomed in peak of ^{60}Co 1332 keV peak

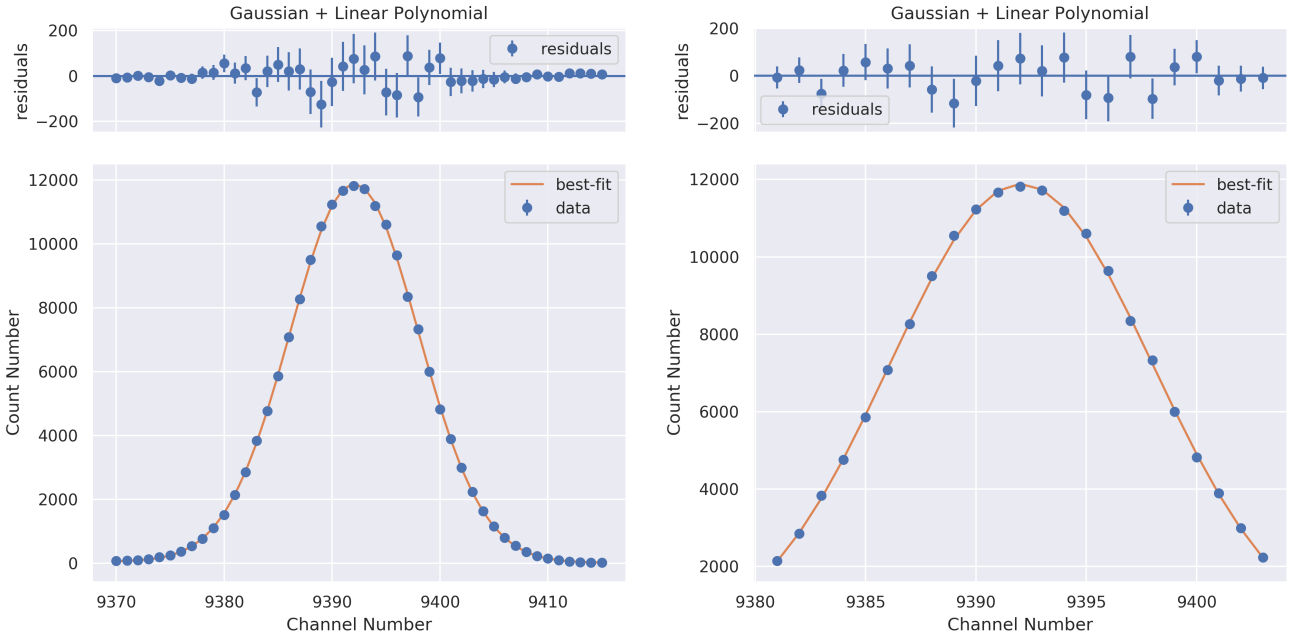
1.2.2 Linear + Gaussian Fit



(a) Full peak with fit

(b) Zoomed in peak with fit

Figure 24: Fit of full & zoomed in peak of ^{60}Co 1173 keV peak

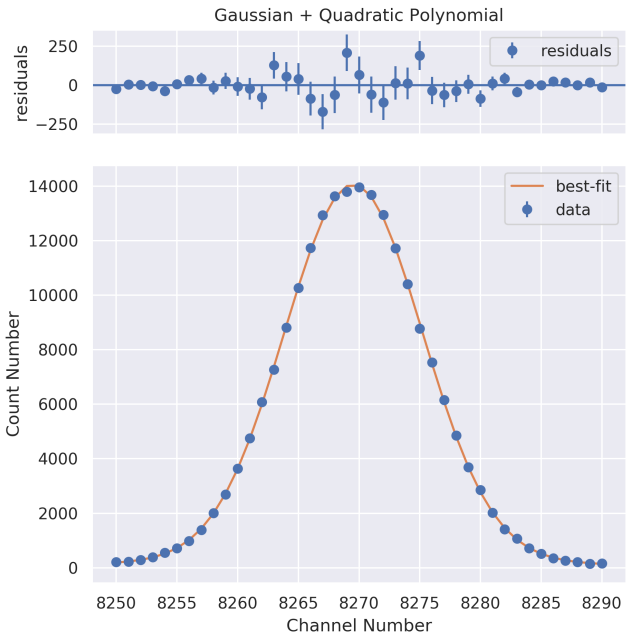


(a) Full peak with fit

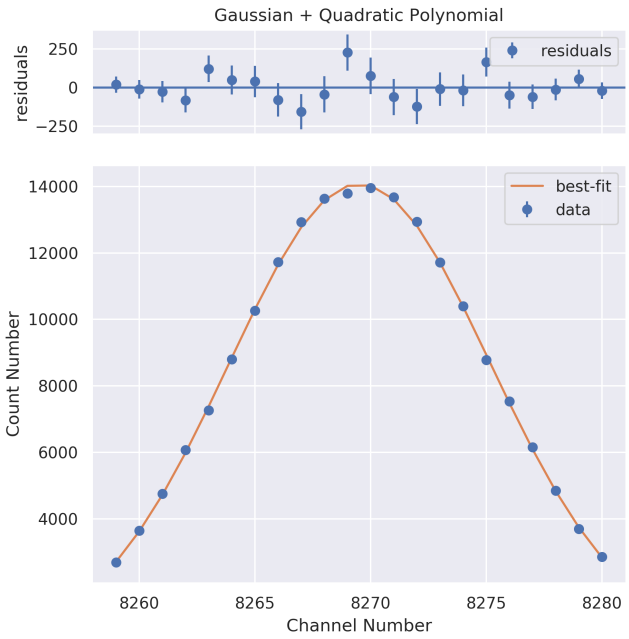
(b) Zoomed in peak with fit

Figure 25: Fit of full & zoomed in peak of ^{60}Co 1332 keV peak

1.2.3 Quadratic + Gaussian Fit

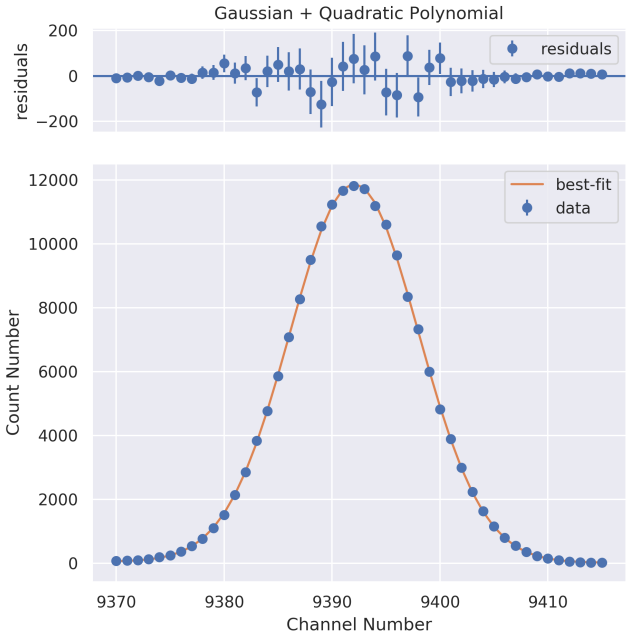


(a) Full peak with fit

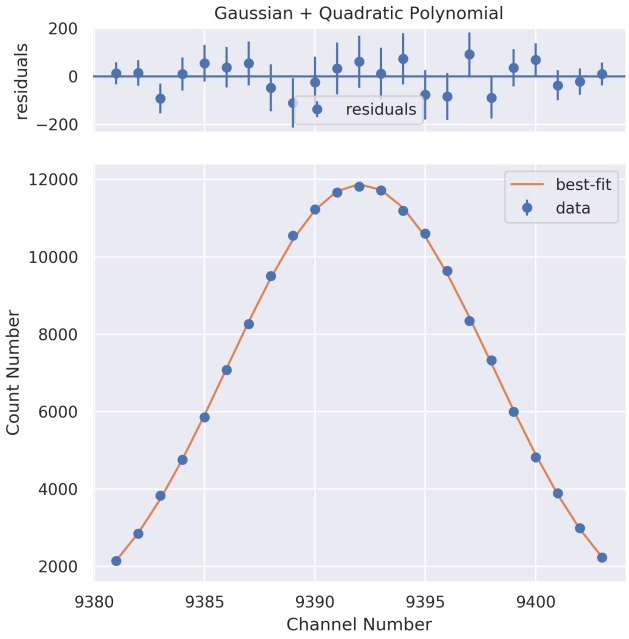


(b) Zoomed in peak with fit

Figure 26: Fit of full & zoomed in peak of ^{60}Co 1173 keV peak



(a) Full peak with fit

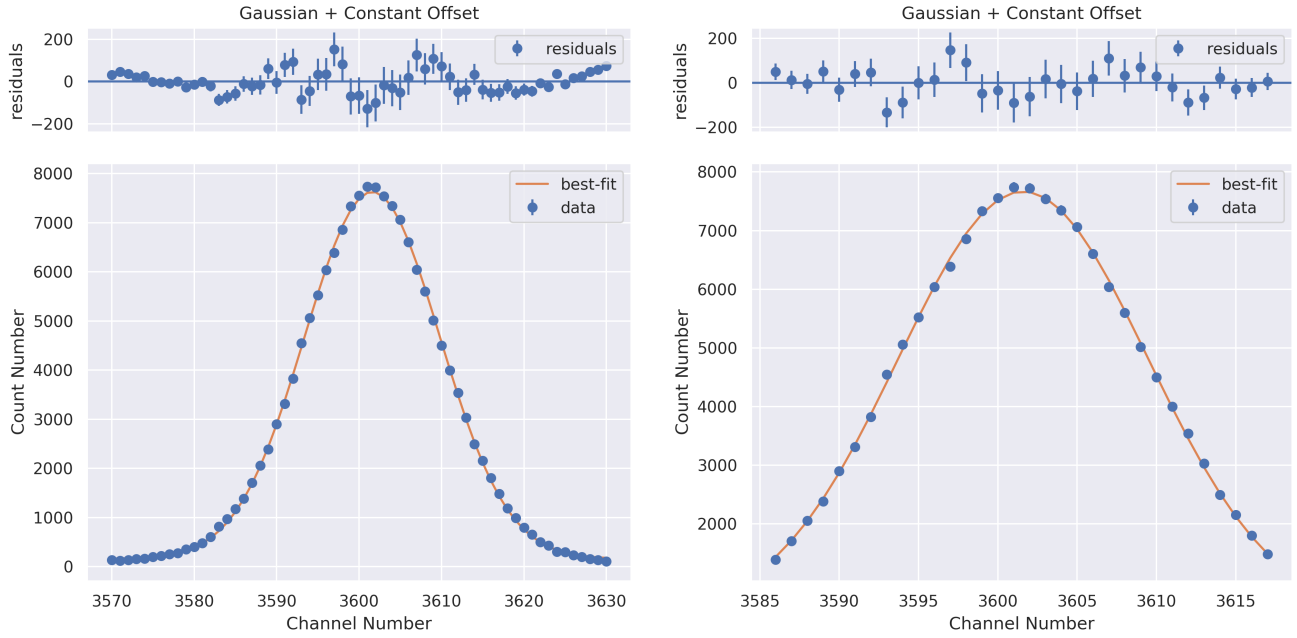


(b) Zoomed in peak with fit

Figure 27: Fit of full & zoomed in peak of ^{60}Co 1332 keV peak

1.3 Sodium-22

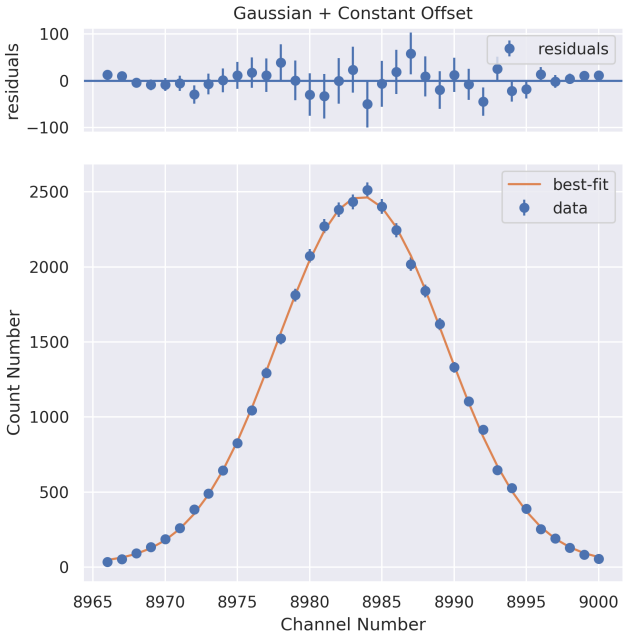
1.3.1 Gaussian Fit Only



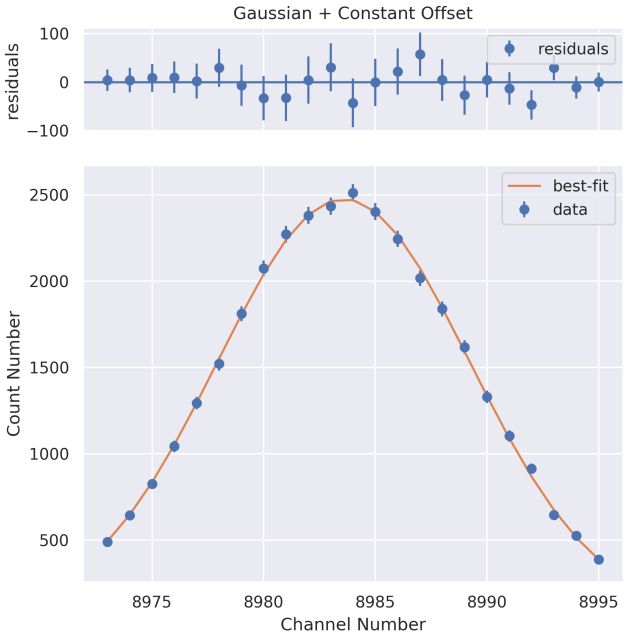
(a) Full peak with fit

(b) Zoomed in peak with fit

Figure 28: Fit of full & zoomed in peak of ^{22}Na 511 keV peak



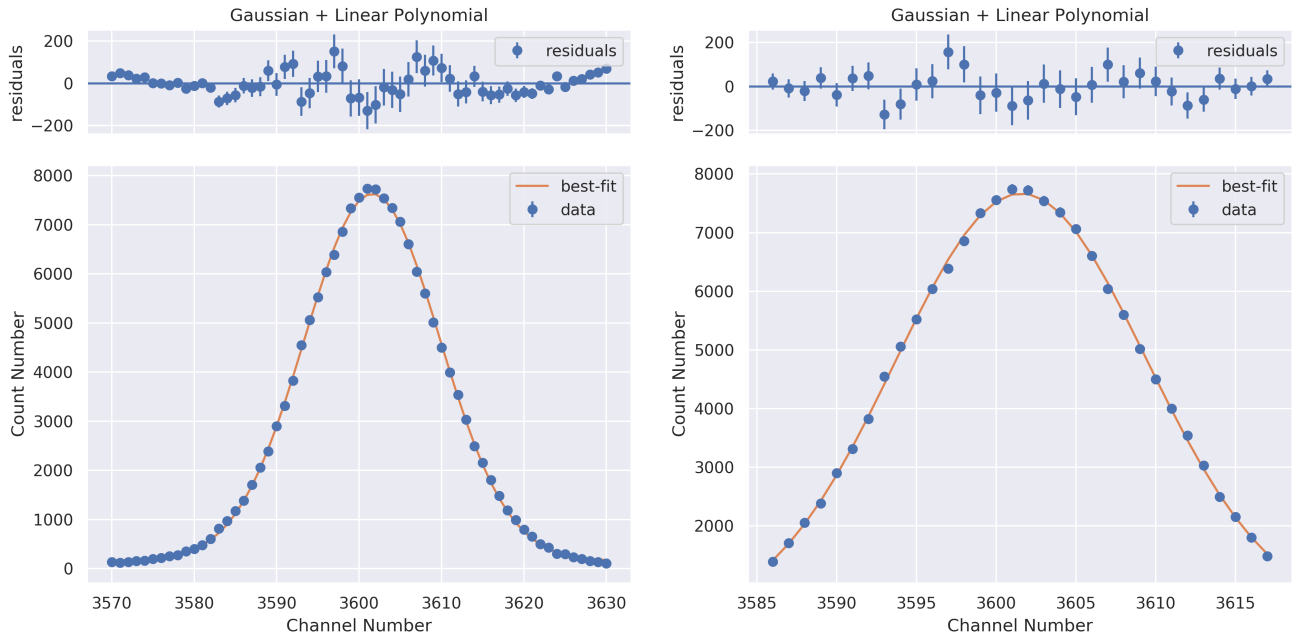
(a) Full peak with fit



(b) Zoomed in peak with fit

Figure 29: Fit of full & zoomed in peak of ^{22}Na 1274 keV peak

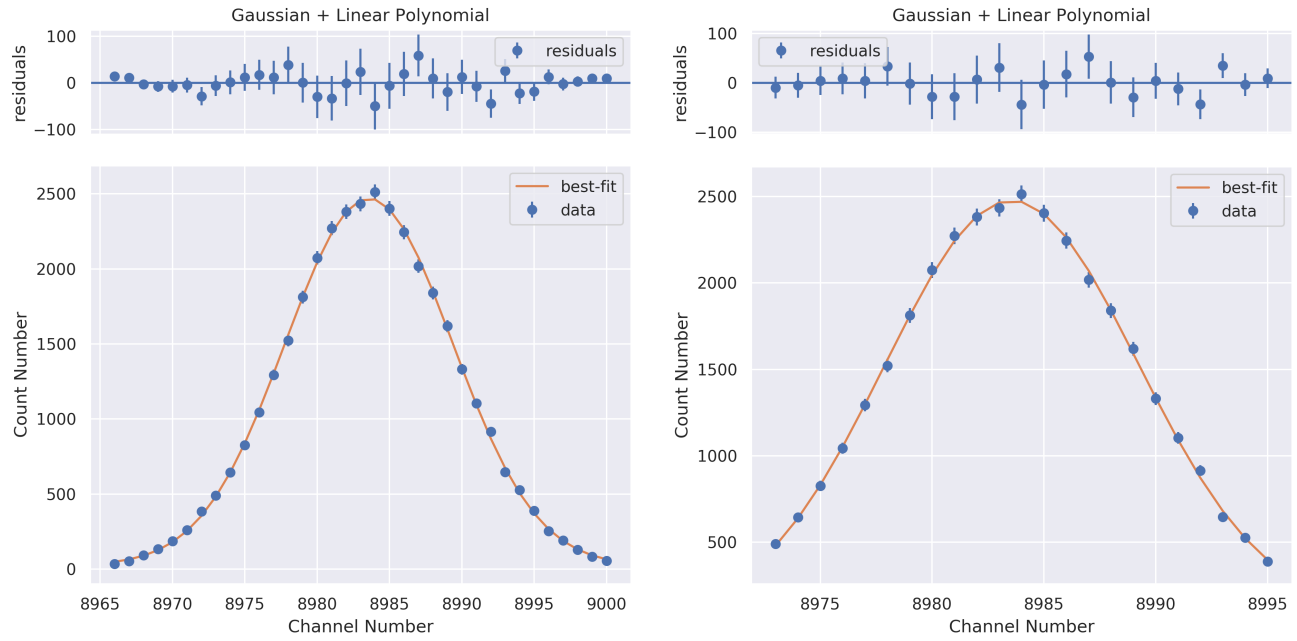
1.3.2 Linear + Gaussian Fit



(a) Full peak with fit

(b) Zoomed in peak with fit

Figure 30: Fit of full & zoomed in peak of ^{60}Na 511 keV peak

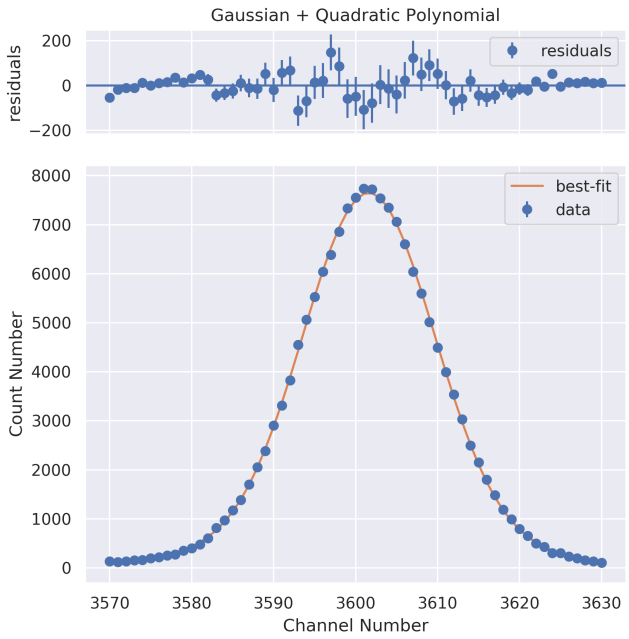


(a) Full peak with fit

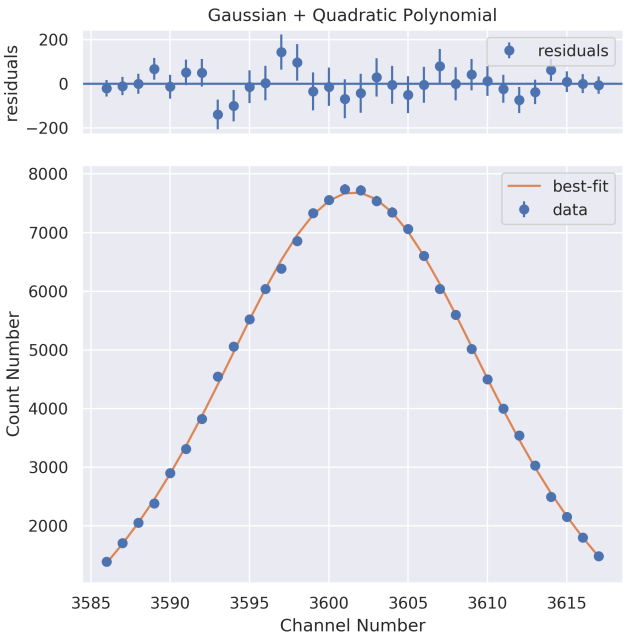
(b) Zoomed in peak with fit

Figure 31: Fit of full & zoomed in peak of ^{22}Na 1274 keV peak

1.3.3 Quadratic + Gaussian Fit

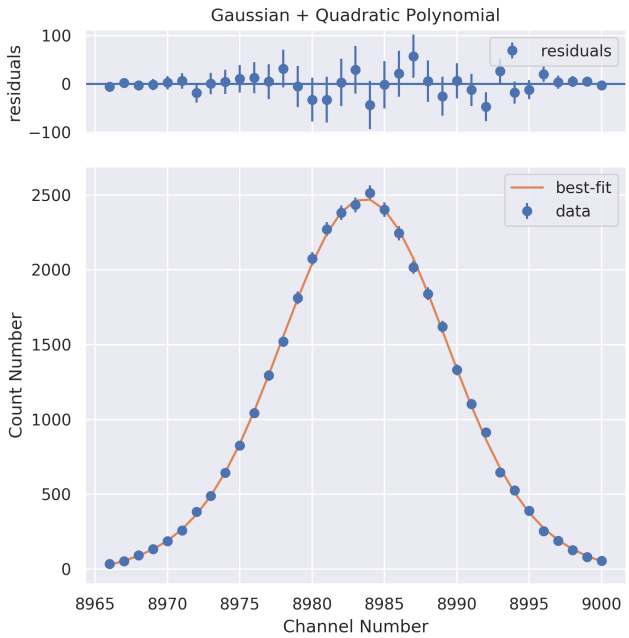


(a) Full peak with fit

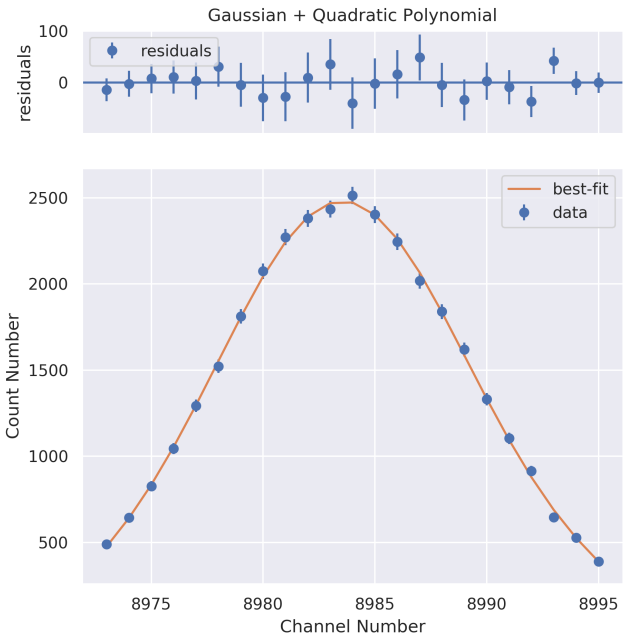


(b) Zoomed in peak with fit

Figure 32: Fit of full & zoomed in peak of ^{60}Na 511 keV peak



(a) Full peak with fit



(b) Zoomed in peak with fit

Figure 33: Fit of full & zoomed in peak of ^{22}Na 1274 keV peak

2 Fit Comparisons

2.1 Barium-133

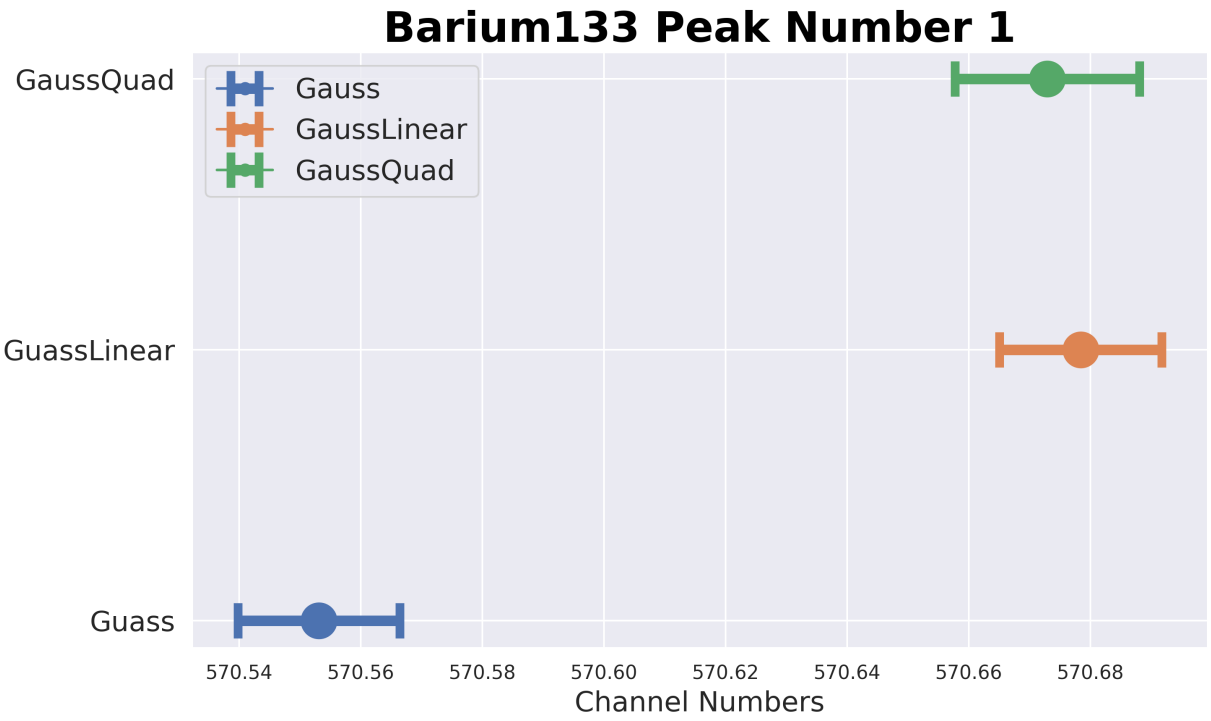


Figure 34: Fit comparisons for ^{133}Ba 81 keV peak

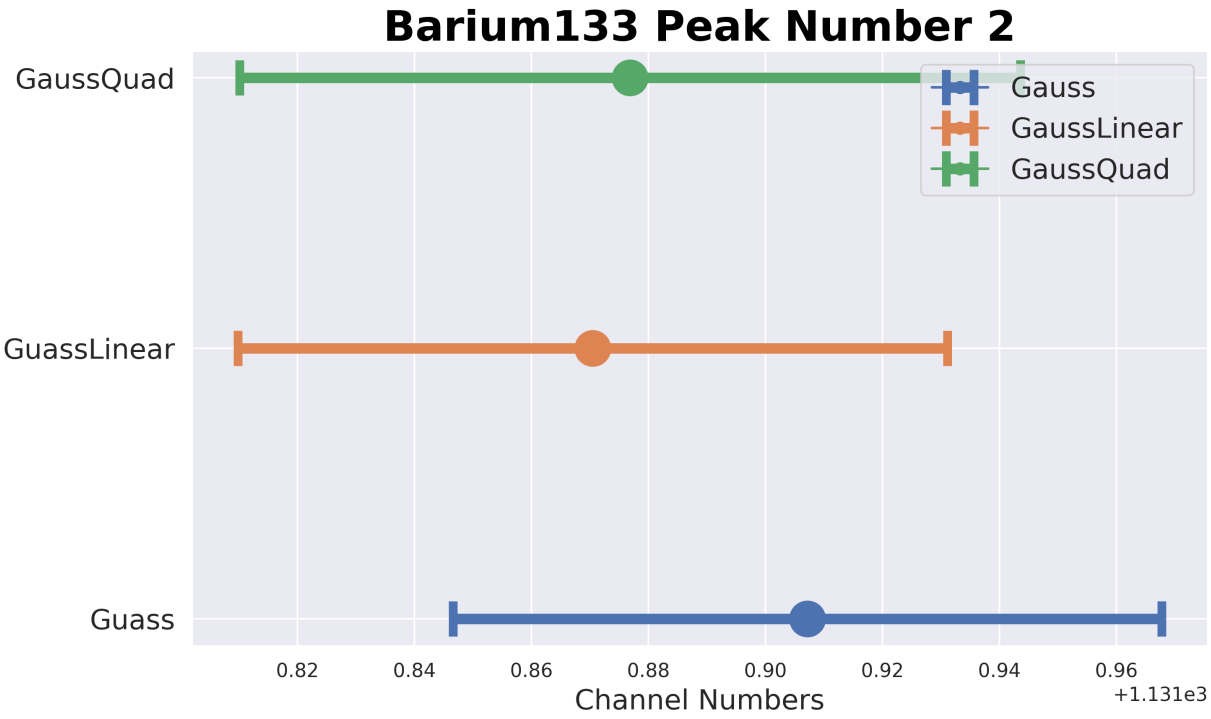


Figure 35: Fit comparisons for ^{133}Ba 161 keV peak

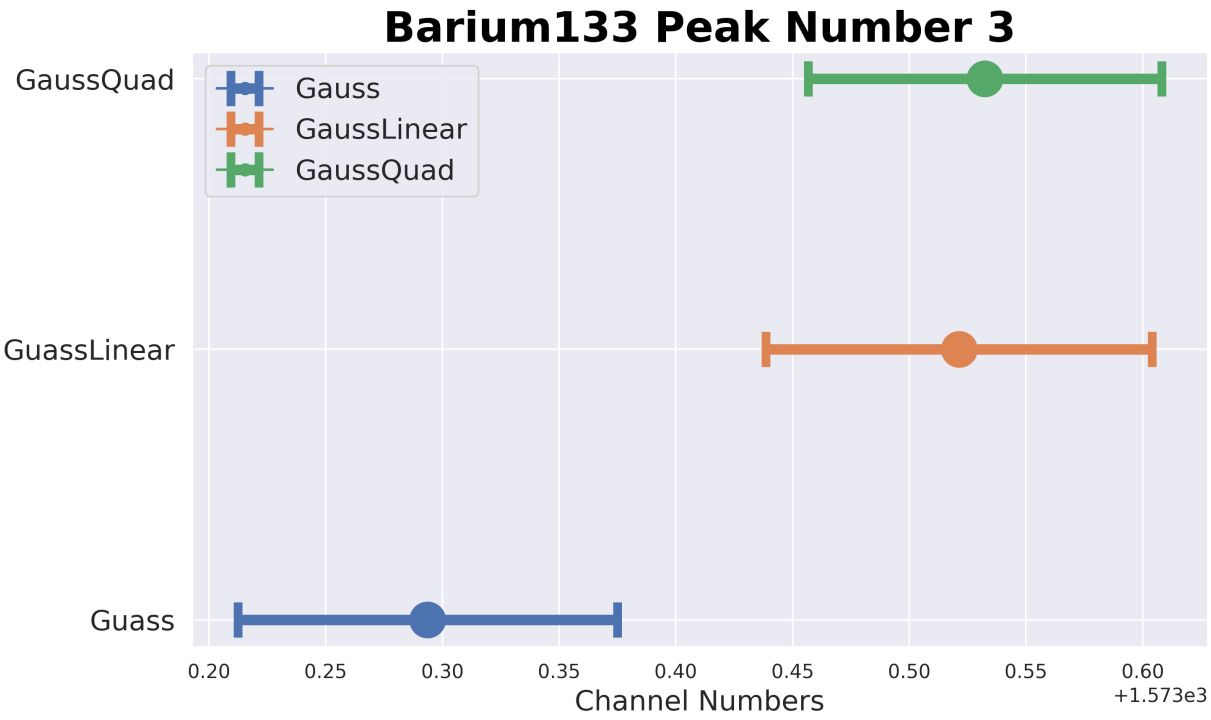


Figure 36: Fit comparisons for ^{133}Ba 223 keV peak

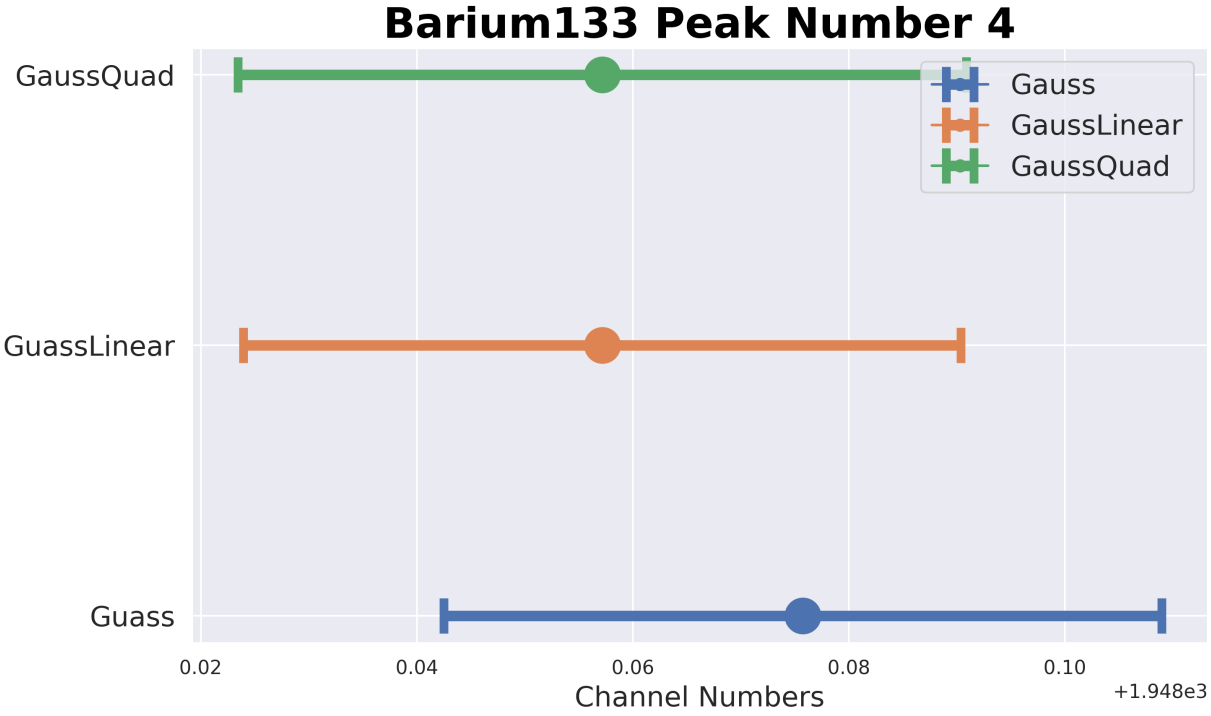


Figure 37: Fit comparisons for ^{133}Ba 276 keV peak

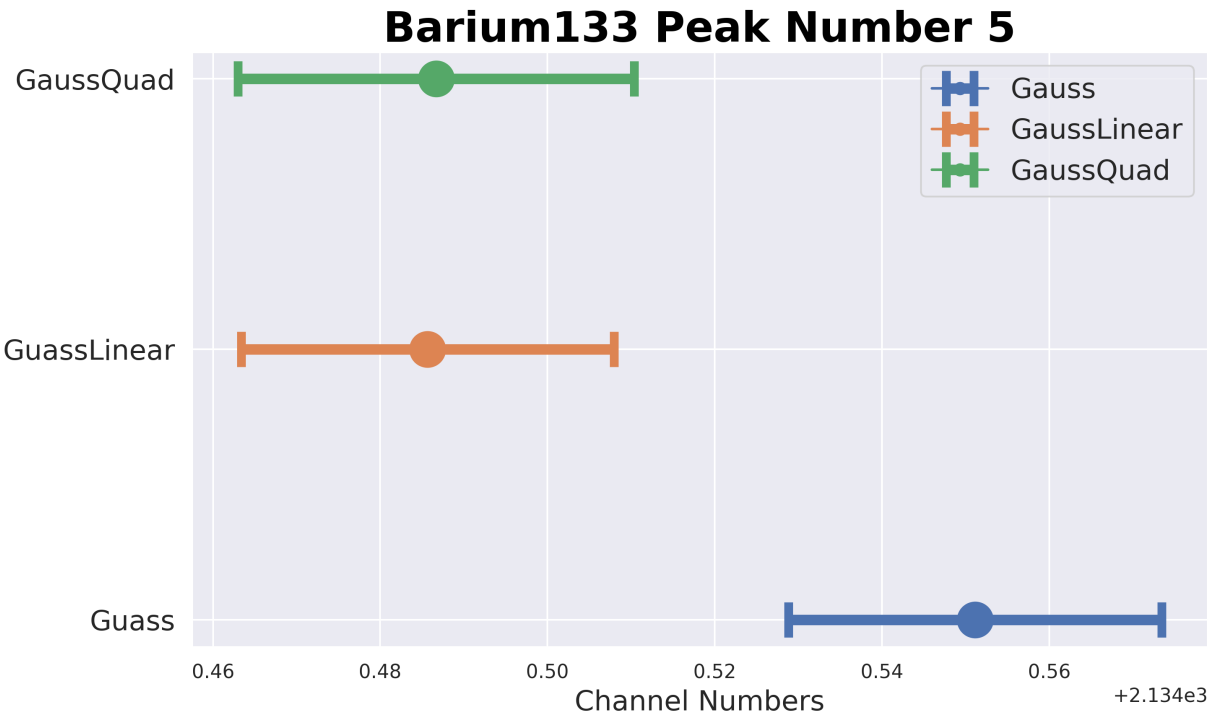


Figure 38: Fit comparisons for ^{133}Ba 303 keV peak

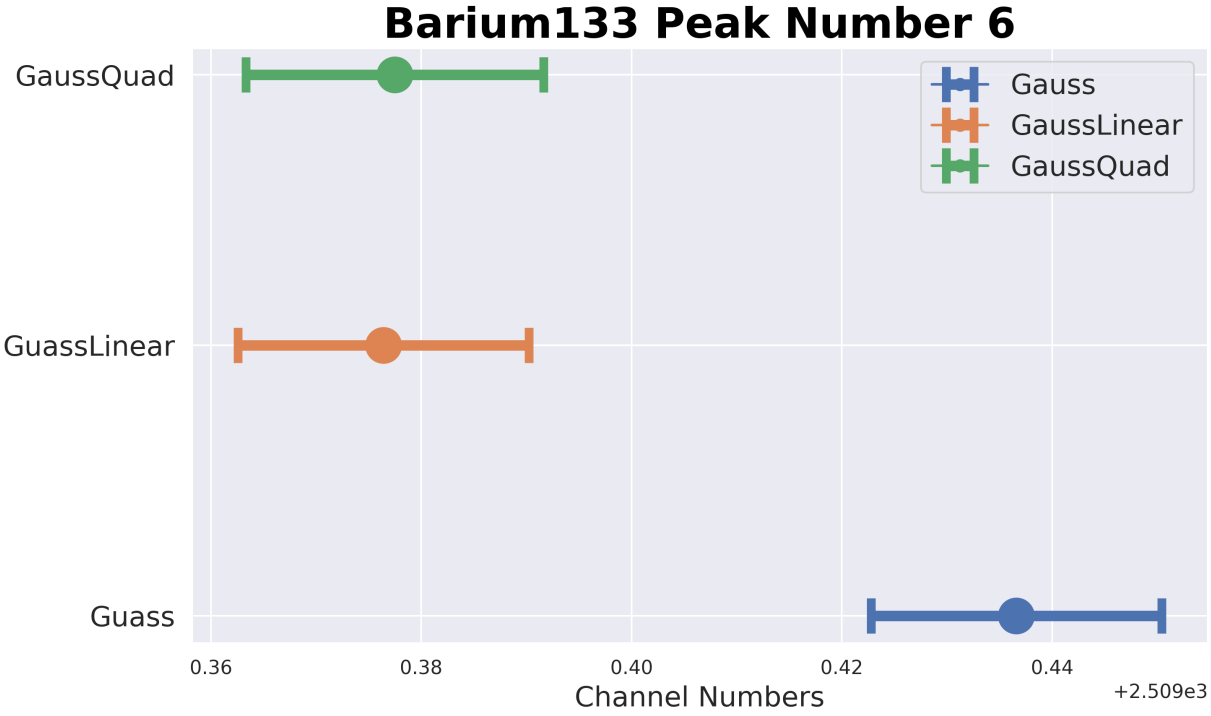


Figure 39: Fit comparisons for ^{133}Ba 356 keV peak

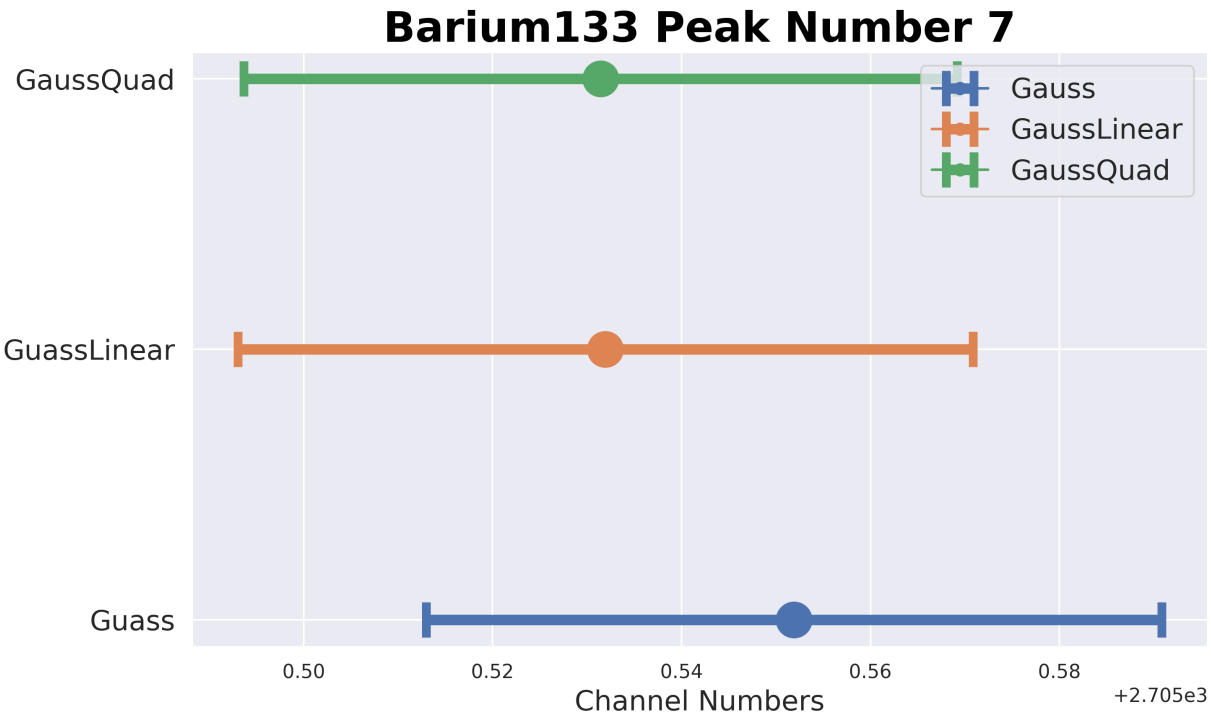


Figure 40: Fit comparisons for ^{133}Ba 384 keV peak

2.2 Cobalt-60

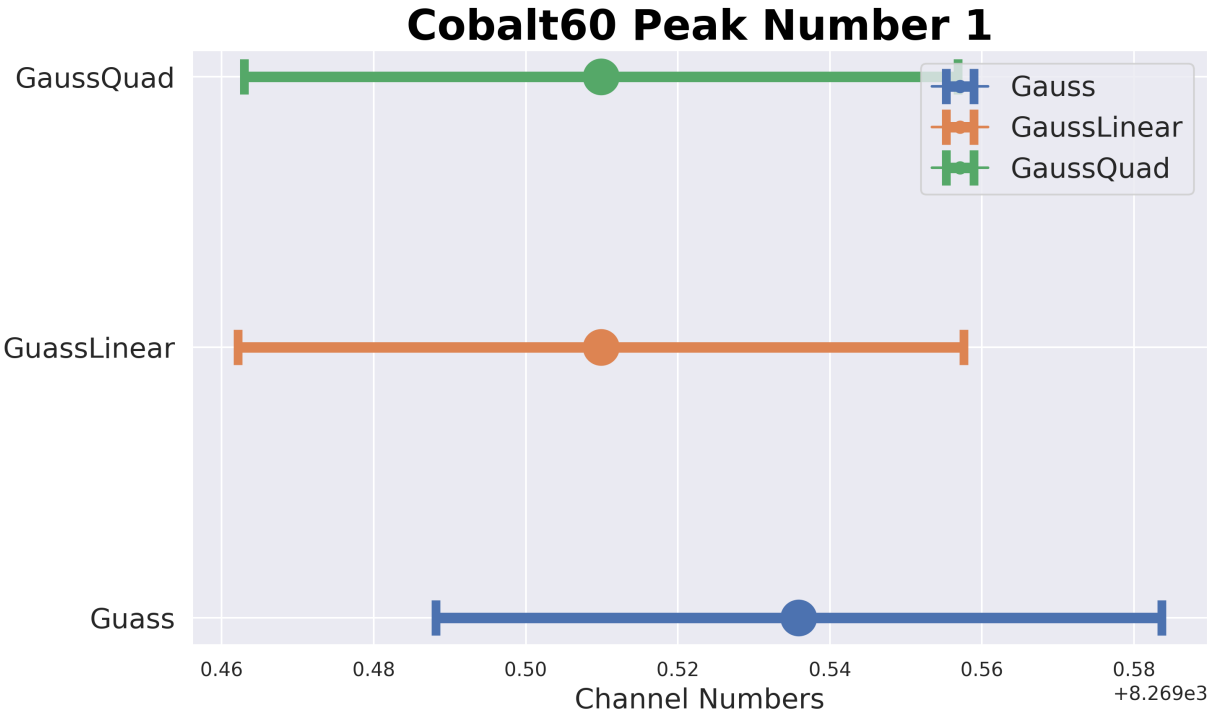


Figure 41: Fit comparisons for ^{60}Co 1173 keV peak

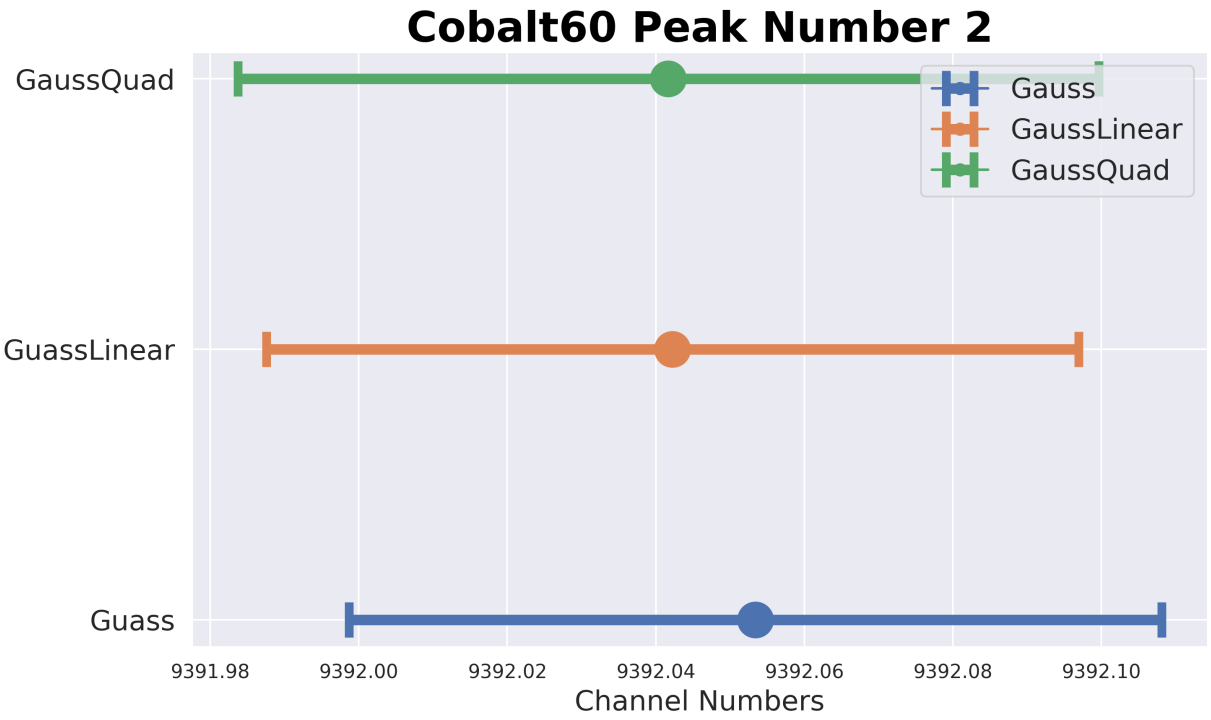


Figure 42: Fit comparisons for ^{60}Co 1332 keV peak

2.3 Sodium-22

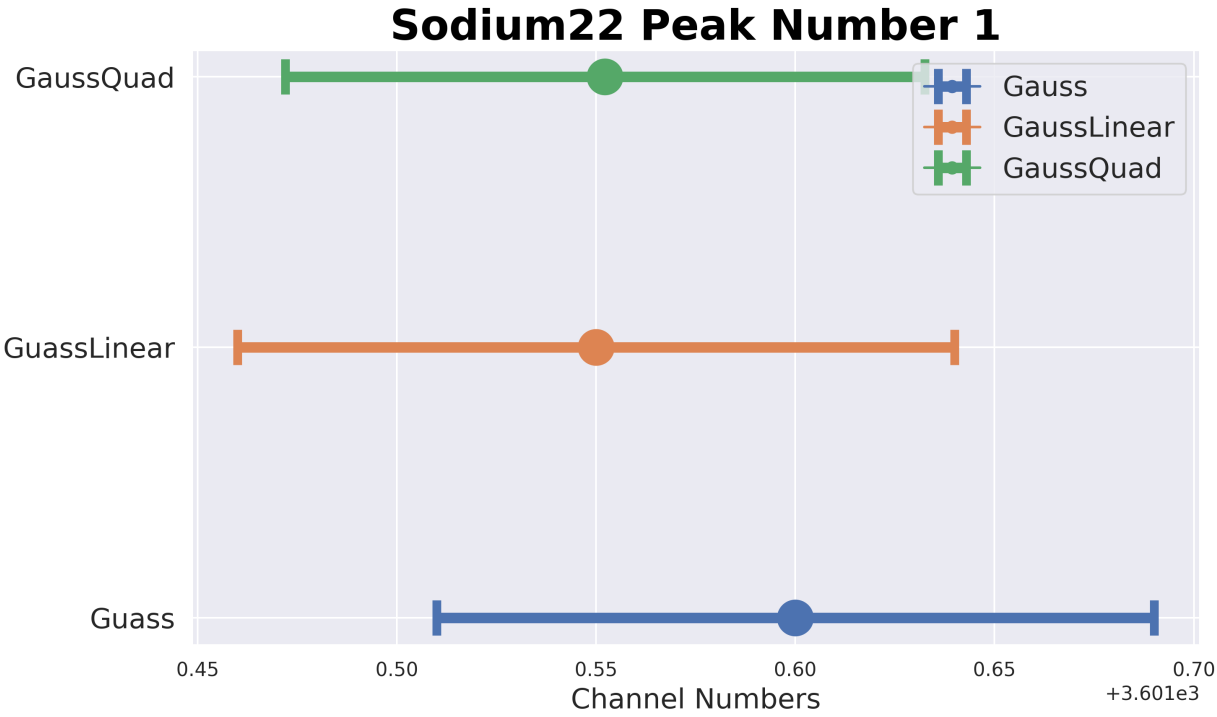


Figure 43: Fit comparisons for ^{22}Na 511 keV peak

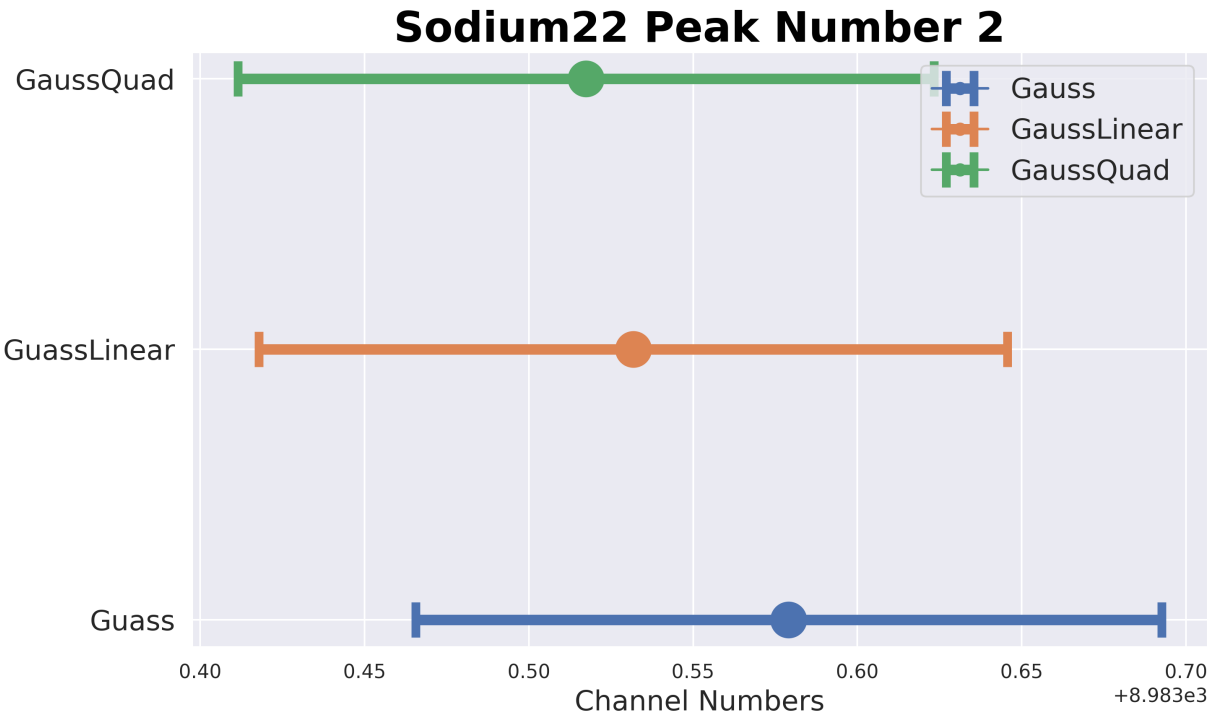


Figure 44: Fit comparisons for ^{22}Na 1274 keV peak

3 Python Code

3.1 GaussianFit.py

```
import numpy as np
import matplotlib.pylab as plt
import seaborn as sns
sns.set()
import scipy.optimize as sciopt
import scipy.stats as scistats
from lmfit import Model, Parameters
import pandas as pd

def ChiSqFunc(Measured,Fitted,Errors):
  ChiSquared = 0
  for i in range(len(Measured)):
    ChiSquared += ((Measured[i] - Fitted[i])**2.0) / ((Errors[i])**2.0)
  ReducedChiSq = ChiSquared/(len(Measured)-2)
  ProbChiSq = (1.0 - scistats.chi2.cdf(ChiSquared,len(Measured)-2) )  * 100.0
  return ChiSquared, ReducedChiSq, ProbChiSq

def Gauss(x,A,mean,sigma,a):
  return (A/(np.sqrt(2*np.pi) * sigma )) *np.exp(-((x-mean)**2.0)/(2.0*(sigma**2.0)))
   ↪  + a

def GaussLinear(x,A,mean,sigma,a,b):
  return (A/(np.sqrt(2*np.pi) * sigma ))*np.exp(-((x-mean)**2.0)/(2.0*(sigma**2.0)))
   ↪  + a + b*x

def GaussQuad(x,A,mean,sigma,a,b,c):
  return (A/(np.sqrt(2*np.pi) * sigma ))*np.exp(-((x-mean)**2.0)/(2.0*(sigma**2.0)))
   ↪  + a + b*x + c*x**2.0

def ChToEnergy(Ch,a,b,c):
  Energy = np.sqrt( ( (Ch - c + (b**2.0 / (4*a))) )/a ) - (b / (2*a))
  return Energy

SetSigma = 2

BariumList = [ [[550,585,570,81],[1125,1140,1132,161],[1565,1582,1574,223],[1935,1960
 ↪  ,1948,276],[2118,2150,2135,303],[2493,2526,2509,356],[2688,2723,2705,384]] , []
 ↪  ]
SodiumList = [ [ [3570,3630,3600,511] , [8966,9000,8984,1274]] , [] ]
CobaltList = [ [ [8250,8290,8270,1173] , [9370,9415,9392,1332]] , [] ]

SourceList = ["Barium133-24HrRun_006_eh_1","Sodium22-24HrData_002_eh_1","Cobalt60-24H
 ↪  rData_004_eh_1"]

PlotResolution = 300

PeakNo = int(1)

datadict = {'Element':[],'Fit type':[], 'Peak number':[], 'Peak type':[], 'Energy
 ↪  (keV)': [], 'Resolution':[], 'Min of range':[], 'Max of range':[], 'Mean':[],
 ↪  'A':[], 'Sigma':[], 'Error on mean':[], 'a':[], 'b':[], 'c':[], 'chisq':[],
 ↪  'Reduced chisq':[], 'Probchisq':[]}

# Fit parameters
```

```
a = 2.63977565e-06
b = 7.04767648
c = -1.91414134e-01

for source in SourceList:
  if source == "Barium133-24HrRun_006_eh_1":
    ElementList = BariumList
    Element = "Barium133"
  elif source == "Sodium22-24HrData_002_eh_1":
    ElementList = SodiumList
    Element = "Sodium22"
  else:
    ElementList = CobaltList
    Element = "Cobalt60"

  for item in ElementList:
    for peak in item:
      #! Initialises the source and ranges for run

      if item == ElementList[0]:
        PeakType = "Full"
      else:
        PeakType = "Zoom"

      df = pd.read_csv(source + ".dat", sep = r"\s+", names = ['channel number','count number'])

      df['count errors'] = np.sqrt(df['count number'])
      df['count errors'] = df['count errors'].replace(0,1)

      MinValue = peak[0]
      MaxValue = peak[1]
      MeanValue = peak[2]
      PeakEnergy = peak[3]

      data = df[(MinValue<=df['channel number']) & (df['channel number']<=MaxValue)]
      #?plt.plot(data['channel number'],data['count number'])
      #?plt.show()
      #-----------------------------------------------------------------------------------------
      #! Gaussian + Offset model only

      GaussModel = Model(Gauss)
      Params = Parameters()

      Params.add('A',value=max(data['count number']),vary=True,min=0)
      Params.add('mean',value=MeanValue,vary=True)
      Params.add('sigma',value=1,vary=True)
      Params.add('a',value=1,vary=True)

      FitResult = GaussModel.fit(data['count number'],params=Params,x=data['channel number'])
      TempList = [ FitResult.best_values['mean'] - SetSigma*FitResult.best_values['sigma'] , FitResult.best_values['mean'] + SetSigma*FitResult.best_values['sigma'] , FitResult.best_values['mean'] , PeakEnergy ]
      if item == ElementList[0]:
```

```
97          ElementList[1].append(TempList)
98      #print(FitResult.best_values)
99      FitResult.plot(yerr=data['count errors'],xlabel='Channel Number',ylabel='Count
        ↪  Number',title='Gaussian + Constant Offset')
100
101     thingy = ChiSqFunc(list(data['count
        ↪  number']),list(FitResult.best_fit),list(data['count errors']))
102     #print(thingy)
103
104     #?plt.plot(data['channel number'],data['count number'])
105     plt.tight_layout()
106     plt.savefig(f'Plots/{Element}/Gauss/Gauss_{PeakNo}_{PeakType}.png',
        ↪  format='png', dpi=PlotResolution)
107     plt.close('all')
108     #plt.show()
109
110     CountMax1 = FitResult.best_values['A'] / ( FitResult.best_values['sigma'] *
        ↪  np.sqrt(2 * np.pi ) ) + FitResult.best_values['a']
111     ErrorOnMean1 = FitResult.best_values['sigma'] / np.sqrt(CountMax1)
112
113     datadict['Element'].append(Element)
114     datadict['Fit type'].append('Gauss')
115     datadict['Energy (keV)'].append(PeakEnergy)
116     FWHM_Energy = 2 * np.sqrt(2*np.log(2)) *
        ↪  ChToEnergy(FitResult.best_values['sigma'],a,b,c)
117     datadict['Resolution'].append( FWHM_Energy / PeakEnergy)
118     #datadict['Resolution'].append( ( (2 *
        ↪  np.sqrt(2*np.log(2))*FitResult.best_values['sigma'] - b ) / a ) /
        ↪  PeakEnergy)
119     datadict['Peak number'].append(PeakNo)
120     datadict['Peak type'].append(PeakType)
121     datadict['Min of range'].append(MinValue)
122     datadict['Max of range'].append(MaxValue)
123     datadict['Mean'].append(FitResult.best_values['mean'])
124     datadict['A'].append(FitResult.best_values['A'])
125     datadict['Sigma'].append(FitResult.best_values['sigma'])
126     datadict['Error on mean'].append(1)
127     datadict['a'].append(FitResult.best_values['a'])
128     datadict['b'].append(0)
129     datadict['c'].append(0)
130     datadict['chisq'].append(thingy[0])
131     datadict['Reduced chisq'].append(thingy[1])
132     datadict['Probchisq'].append(thingy[2])
133
134
135     #-------------------------------------------------------------------------
        ↪  ---------------------------
136     #! Gaussain + Linear model
137
138     GaussModel2 = Model(GaussLinear)
139     Params2 = Parameters()
140
141     Params2.add('A',value=max(data['count number']),vary=True,min=0)
142     Params2.add('mean',value=MeanValue,vary=True)
143     Params2.add('sigma',value=1,vary=True)
144     Params2.add('a',value=1,vary=True)
145     Params2.add('b',value=1,vary=True)
```

```
146
147         FitResult2 = GaussModel2.fit(data['count
            ↪  number'],params=Params2,x=data['channel number'])
148         #TempList2 = [ FitResult2.best_values['mean'] - SetSigma
            ↪  *FitResult2.best_values['sigma'] , FitResult2.best_values['mean'] +
            ↪  SetSigma *FitResult2.best_values['sigma'] , FitResult2.best_values['mean']
            ↪  ]
149         #print(FitResult2.best_values)
150         FitResult2.plot(yerr=data['count errors'],xlabel='Channel Number',ylabel='Count
            ↪  Number',title='Gaussian + Linear Polynomial')
151
152         thingy2 = ChiSqFunc(list(data['count
            ↪  number']),list(FitResult2.best_fit),list(data['count errors']))
153
154         #?plt.plot(data['channel number'],data['count number'])
155         plt.tight_layout()
156         plt.savefig(f'Plots/{Element}/Linear/Linear_{PeakNo}_{PeakType}.png',
            ↪  format='png', dpi=PlotResolution)
157         plt.close('all')
158
159         CountMax2 = FitResult2.best_values['A'] / ( FitResult2.best_values['sigma'] *
            ↪  np.sqrt(2 * np.pi ) ) + FitResult2.best_values['a'] +
            ↪  FitResult2.best_values['b'] * FitResult2.best_values['mean']
160         ErrorOnMean2 = FitResult2.best_values['sigma'] / np.sqrt(CountMax2)
161
162         datadict['Element'].append(Element)
163         datadict['Fit type'].append('Linear')
164         datadict['Energy (keV)'].append(PeakEnergy)
165         FWHM_Energy = 2 * np.sqrt(2*np.log(2)) *
            ↪  ChToEnergy(FitResult2.best_values['sigma'],a,b,c)
166         datadict['Resolution'].append( FWHM_Energy / PeakEnergy)
167         datadict['Peak number'].append(PeakNo)
168         datadict['Peak type'].append(PeakType)
169         datadict['Min of range'].append(MinValue)
170         datadict['Max of range'].append(MaxValue)
171         datadict['Mean'].append(FitResult2.best_values['mean'])
172         datadict['A'].append(FitResult2.best_values['A'])
173         datadict['Sigma'].append(FitResult2.best_values['sigma'])
174         datadict['Error on mean'].append(1)
175         datadict['a'].append(FitResult2.best_values['a'])
176         datadict['b'].append(FitResult2.best_values['b'])
177         datadict['c'].append(0)
178         datadict['chisq'].append(thingy2[0])
179         datadict['Reduced chisq'].append(thingy2[1])
180         datadict['Probchisq'].append(thingy2[2])
181
182         #-------------------------------------------------------------------------
            ↪  ---------------------------
183         #! Gaussain + Quadratic model
184
185         GaussModel3 = Model(GaussQuad)
186         Params3 = Parameters()
187
188         Params3.add('A',value=max(data['count number']),vary=True,min=0)
189         Params3.add('mean',value=MeanValue,vary=True)
190         Params3.add('sigma',value=1,vary=True)
191         Params3.add('a',value=1,vary=True)
```

```
192         Params3.add('b',value=1,vary=True)
193         Params3.add('c',value=1,vary=True)
194 
195         FitResult3 = GaussModel3.fit(data['count
            ↪  number'],params=Params3,x=data['channel number'])
196         #TempList3 = [ FitResult3.best_values['mean'] - SetSigma
            ↪  *FitResult3.best_values['sigma'] , FitResult3.best_values['mean'] +
            ↪  SetSigma *FitResult3.best_values['sigma'] , FitResult3.best_values['mean']
            ↪  ]
197         #print(FitResult3.best_values)
198         FitResult3.plot(yerr=data['count errors'],xlabel='Channel Number',ylabel='Count
            ↪  Number',title='Gaussian + Quadratic Polynomial')
199 
200         thingy3 = ChiSqFunc(list(data['count
            ↪  number']),list(FitResult3.best_fit),list(data['count errors']))
201         #print(thingy3)
202 
203         #?plt.plot(data['channel number'],data['count number'])
204         plt.tight_layout()
205         plt.savefig(f'Plots/{Element}/Quad/Quad_{PeakNo}_{PeakType}.png', format='png',
            ↪  dpi=PlotResolution)
206         plt.close('all')
207 
208         CountMax3 = FitResult3.best_values['A'] / ( FitResult3.best_values['sigma'] *
            ↪  np.sqrt(2 * np.pi ) ) + FitResult3.best_values['a'] +
            ↪  FitResult3.best_values['b'] * FitResult3.best_values['mean'] +
            ↪  FitResult3.best_values['c'] * (FitResult3.best_values['mean'])** 2.0
209         ErrorOnMean3 = FitResult3.best_values['sigma'] / np.sqrt(CountMax3)
210 
211         datadict['Element'].append(Element)
212         datadict['Fit type'].append('Quad')
213         datadict['Energy (keV)'].append(PeakEnergy)
214         FWHM_Energy = 2 * np.sqrt(2*np.log(2)) *
            ↪  ChToEnergy(FitResult3.best_values['sigma'],a,b,c)
215         datadict['Resolution'].append( FWHM_Energy / PeakEnergy)
216         datadict['Peak number'].append(PeakNo)
217         datadict['Peak type'].append(PeakType)
218         datadict['Min of range'].append(MinValue)
219         datadict['Max of range'].append(MaxValue)
220         datadict['Mean'].append(FitResult3.best_values['mean'])
221         datadict['A'].append(FitResult3.best_values['A'])
222         datadict['Sigma'].append(FitResult3.best_values['sigma'])
223         datadict['Error on mean'].append(1)
224         datadict['a'].append(FitResult3.best_values['a'])
225         datadict['b'].append(FitResult3.best_values['b'])
226         datadict['c'].append(FitResult3.best_values['c'])
227         datadict['chisq'].append(thingy3[0])
228         datadict['Reduced chisq'].append(thingy3[1])
229         datadict['Probchisq'].append(thingy3[2])
230 
231 
232         Delta12 = np.abs(FitResult.best_values['mean'] - FitResult2.best_values['mean'])
233         ErrorDelta12 = np.sqrt( ErrorOnMean1**2.0 + ErrorOnMean2**2.0 )
234 
235         Delta23 = np.abs(FitResult2.best_values['mean'] -
            ↪  FitResult3.best_values['mean'])
236         ErrorDelta23 = np.sqrt( ErrorOnMean2**2.0 + ErrorOnMean3**2.0 )
```

```
237
238
239         if PeakType == "Zoom":
240           if abs(ErrorDelta23) > abs(Delta23):
241             print('best fit')
242           else:
243             print('not best fit')
244
245           TitleFont = {'size':'24', 'color':'black', 'weight':'bold'}
246           AxTitleFont = {'size':'16'}
247           plt.figure(figsize=(10,6))
248           plt.errorbar(FitResult.best_values['mean'],1,xerr=ErrorOnMean1,elinewidth=6,c
              ↪  apsize=10,capthick=5,marker='o',markersize=20,label="Gauss")
249           plt.errorbar(FitResult2.best_values['mean'],1.25,xerr=ErrorOnMean2,elinewidth
              ↪  =6,capsize=10,capthick=5,marker='o',markersize=20,label="GaussLinear")
250           plt.errorbar(FitResult3.best_values['mean'],1.5,xerr=ErrorOnMean3,elinewidth=
              ↪  6,capsize=10,capthick=5,marker='o',markersize=20,label="GaussQuad")
251           plt.legend(fontsize=16,markerscale=0.33)
252           plt.title(str(Element) + ' Peak Number ' + str(PeakNo),**TitleFont)
253           plt.xlabel('Channel Numbers',**AxTitleFont)
254           plt.yticks([1,1.25,1.5],['Guass','GuassLinear','GaussQuad'],**AxTitleFont)
255           plt.tight_layout()
256           plt.savefig(f'Plots/{Element}/FitComparison_Peak{PeakNo}.png', format='png',
              ↪  dpi=PlotResolution)
257           plt.close('all')
258
259         PeakNo +=1
260
261       PeakNo = 1
262
263   Fitdf = pd.DataFrame(datadict)
264   Fitdf.to_csv('Gamma_Peak_Stats_and_Params.csv')
```

3.2 Calibration.py

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as sciopt
import seaborn as sns
import pandas as pd
sns.set()
import scipy.stats as scistats

source = "MysterySource-2HrRun_001_eh_1"
df = pd.read_csv(source+ ".dat", sep=r"\s+",names = ['channel number','count number'])

TitleFont = {'size':'25', 'color':'black', 'weight':'bold'}
AxTitleFont = {'size':'22'}

def Linear(E,a,b):
    return E*a + b

def Quadratic(E,a,b,c):
    return a*E**2.0 + E*b + c

def ChiSqFunc(Measured,Fitted,Errors,Params):
  ChiSquared = 0
  for i in range(len(Measured)):
    ChiSquared += ((Measured[i] - Fitted[i])**2.0) / ((Errors[i])**2.0)
  ReducedChiSq = ChiSquared/(len(Measured)-len(Params))
  ProbChiSq = (1.0 - scistats.chi2.cdf(ChiSquared,len(Measured)-len(Params)) )  *
   ↪  100.0
  return ChiSquared, ReducedChiSq, ProbChiSq


FitDF = pd.read_csv('Gamma_Peak_Stats_and_Params.csv', names=['Element', 'Fit type',
 ↪  'Peak number',  'Peak type', 'Energy (keV)', 'Resolution', 'Min', 'Max', 'Mean',
 ↪  'A', 'Sigma', 'Error' ,'a', 'b', 'chisq', 'Reduced chisq',
 ↪  'Probchisq'],usecols=list(range(1,18)))

FitDF = FitDF[FitDF['Fit type'] == 'Linear']
FitDF = FitDF[FitDF['Peak type'] == 'Zoom']

FitDF['Mean'] = FitDF['Mean'].astype(float)
FitDF['Error'] = FitDF['Error'].astype(float)

FitDF['Energy (keV)'] = FitDF['Energy (keV)'].astype(float)

plt.plot(FitDF['Energy (keV)'],FitDF['Mean'],'bo')

plt.show()

DataBa = FitDF[FitDF['Element'] == 'Barium133']
DataCo = FitDF[FitDF['Element'] == 'Cobalt60']
DataNa = FitDF[FitDF['Element'] == 'Sodium22']


ParamsLinear, ErrorsLinear = sciopt.curve_fit(Quadratic,FitDF['Energy
 ↪  (keV)'],FitDF['Mean'])
print(ParamsLinear)

```

```
rough_EC_a = a = ParamsLinear[0]
rough_EC_b = b = ParamsLinear[1]
rough_EC_c = c = ParamsLinear[2]


Energies = np.sqrt( (FitDF['Mean'] - c + (b**2.0/ (4 * a)))/a ) - (b/(2*a))

plt.errorbar(DataBa['Energy (keV)'],DataBa['Mean'], fmt='ro',label="Barium-133",
    markersize=10.5,yerr=DataBa['Error'].values)
plt.errorbar(DataCo['Energy (keV)'],DataCo['Mean'],fmt='bo',label="Cobalt-60",
    markersize=10.5,yerr=DataCo['Error'].values)
plt.errorbar(DataNa['Energy (keV)'],DataNa['Mean'],fmt='yo',label="Sodium-22",
    markersize=10.5,yerr=DataNa['Error'].values)

plt.plot(Energies,Quadratic(Energies,*ParamsLinear),label='Fit')
plt.title('Energy calibration plot with fit',**TitleFont)
plt.xlabel('Gamma Energy [keV]',**AxTitleFont)
plt.ylabel('Channel Number',**AxTitleFont)
plt.legend()
plt.show()


plt.errorbar(DataBa['Energy (keV)'],[(DataBa['Mean'].iloc[i] -
    Quadratic(DataBa['Energy (keV)'].iloc[i],*ParamsLinear)) for i in
    range(len(DataBa['Energy (keV)']))],fmt='ro',label="Barium-133",
    markersize=7.5,yerr=DataBa['Error'].values)
plt.errorbar(DataCo['Energy (keV)'],[(DataCo['Mean'].iloc[i] -
    Quadratic(DataCo['Energy (keV)'].iloc[i],*ParamsLinear)) for i in
    range(len(DataCo['Energy (keV)']))],fmt='bo',label="Cobalt-60",
    markersize=7.5,yerr=DataCo['Error'].values)
plt.errorbar(DataNa['Energy (keV)'],[(DataNa['Mean'].iloc[i] -
    Quadratic(DataNa['Energy (keV)'].iloc[i],*ParamsLinear)) for i in
    range(len(DataNa['Energy (keV)']))],fmt='yo',label="Sodium-22",
    markersize=7.5,yerr=DataNa['Error'].values)
plt.title('Energy calibration residual plot',**TitleFont)
plt.xlabel('Gamma Energy [keV]',**AxTitleFont)
plt.ylabel('Channel Number Residual',**AxTitleFont)
plt.legend()
plt.show()


ChiStats = ChiSqFunc(list(FitDF['Mean']),list(Quadratic(FitDF['Energy
    (keV)'],*ParamsLinear)),list(FitDF['Error']),ParamsLinear)
print(ChiStats)
```

3.3 Subtraction.py

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import scipy as sc
import pandas as pd

# If statements that control what the code does!
PlotMysterySource = True
PlotKnownSources = True

# Plot controlls
TitleFont = {'size':'23', 'color':'black', 'weight':'bold'}
AxTitleFont = {'size':'20'}

if PlotMysterySource:
  BgWithShield = "WeekendBgWithShield_003_eh_1"
  BgWithShieldDF = pd.read_csv(BgWithShield+ ".dat", sep=r"\s+",names = ['channel number','count number'])

  MysterySource = "MysterySource-24HrRun_001_eh_1"
  MysterySourceDF = pd.read_csv(MysterySource+ ".dat", sep=r"\s+",names = ['channel number','count number'])

  # Normalises the timings of the two runs to 24 hours each
  BgWithShieldDF['count number2'] = BgWithShieldDF['count number']
  BgWithShieldDF['count number'] = BgWithShieldDF['count number'] / ( 239068  ) * 86400

  MysterySourceDF['count number'] = MysterySourceDF['count number'] - BgWithShieldDF['count number']

  a = 2.63977565e-06
  b = 7.04767648
  c = -1.91414134e-01

  MysterySourceDF['Energies'] =  np.sqrt( (BgWithShieldDF['channel number'] - c + (b**2.0/ (4 * a)))/a ) - (b/(2*a))
  BgWithShieldDF['Energies'] =  np.sqrt( (BgWithShieldDF['channel number'] - c + (b**2.0/ (4 * a)))/a ) - (b/(2*a))


  plt.semilogy(MysterySourceDF['Energies'],MysterySourceDF['count number'],label="Mystery Source")
  plt.semilogy(BgWithShieldDF['Energies'],BgWithShieldDF['count number2'],label="Background With Shield")
  plt.xlabel('Gamma Energy [keV]',**AxTitleFont)
  plt.ylabel('Log-10 of count number',**AxTitleFont)
  plt.title('Mystery Source With Background Subtracted',**TitleFont)
  plt.xlim(0,2216)
  plt.ylim(1)
  plt.legend()
  plt.show()


if PlotKnownSources:
```

```
49  BgWithShield = "WeekendBgWithShield_003_eh_1"
50  BgWithShieldDF = pd.read_csv(BgWithShield+ ".dat", sep=r"\s+",names = ['channel
    ↪  number','count number'])
51
52  Sodium = 'Sodium22-24HrData_002_eh_1'
53  SodiumDF = pd.read_csv(Sodium+ ".dat", sep=r"\s+",names = ['channel number','count
    ↪  number'])
54
55  Barium = 'Barium133-24HrRun_006_eh_1'
56  BariumDF = pd.read_csv(Barium+ ".dat", sep=r"\s+",names = ['channel number','count
    ↪  number'])
57
58  Cobalt = 'Cobalt60-24HrData_004_eh_1'
59  CobaltDF = pd.read_csv(Cobalt+ ".dat", sep=r"\s+",names = ['channel number','count
    ↪  number'])
60
61  # Normalises the background to the correct timings
62  BgWithShieldDFCobalt = BgWithShieldDF.copy()
63  BgWithShieldDF['count number'] = BgWithShieldDF['count number'] / ( 239068  ) *
    ↪  86400
64  BgWithShieldDFCobalt['count number'] = BgWithShieldDF['count number'] / ( 239068  )
    ↪  * (86400/12.0)
65
66  # Subtracts the background from the original data
67  SodiumDF['count number'] = SodiumDF['count number'] - BgWithShieldDF['count number']
68  BariumDF['count number'] = BariumDF['count number'] - BgWithShieldDF['count number']
69  CobaltDF['count number'] = CobaltDF['count number'] - BgWithShieldDFCobalt['count
    ↪  number']
70
71  # Fitting parameters
72  a = 7.01164642
73  b = 8.1697501
74
75  # Converting channel numbers to energies
76  SodiumDF['Energies'] = ( SodiumDF['channel number'] - b )/a
77  BariumDF['Energies'] = ( BariumDF['channel number'] - b )/a
78  CobaltDF['Energies'] = ( CobaltDF['channel number'] - b )/a
79
80
81  # Now plotting the different elements on different log-y plots.
82
83  plt.figure(1)
84  plt.semilogy(SodiumDF['Energies'],SodiumDF['count number'],label="Sodium 22
    ↪  data",color='yellow')
85  TitleFont = {'size':'25', 'color':'black', 'weight':'bold'}
86  AxTitleFont = {'size':'22'}
87  plt.xlabel('Gamma Energy [keV]',**AxTitleFont)
88  plt.ylabel('Log-10 of count number',**AxTitleFont)
89  plt.title('Sodium 22, no background, energies',**TitleFont)
90  plt.xlim(0)
91  plt.ylim(1)
92  plt.legend()
93
94
95  plt.figure(2)
96  plt.semilogy(BariumDF['Energies'],BariumDF['count number'],label="Barium 133
    ↪  data",color='green')
```

```
    TitleFont = {'size':'25', 'color':'black', 'weight':'bold'}
    AxTitleFont = {'size':'22'}
    plt.xlabel('Gamma Energy [keV]',**AxTitleFont)
    plt.ylabel('Log-10 of count number',**AxTitleFont)
    plt.title('Barium 133, no background, energies',**TitleFont)
    plt.xlim(0)
    plt.ylim(1)
    plt.legend()


    plt.figure(3)
    plt.semilogy(CobaltDF['Energies'],CobaltDF['count number'],label="Cobalt 60
    ↪  data",color='blue')
    TitleFont = {'size':'25', 'color':'black', 'weight':'bold'}
    AxTitleFont = {'size':'22'}
    plt.xlabel('Gamma Energy [keV]',**AxTitleFont)
    plt.ylabel('Log-10 of count number',**AxTitleFont)
    plt.title('Cobalt 60, no background, energies',**TitleFont)
    plt.xlim(0)
    plt.ylim(1)
    plt.legend()
    plt.show()
```

3.4 Resolution.py

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as sciopt
import seaborn as sns
import pandas as pd
sns.set()
import scipy.stats as scistats


def ResolutionFit(E,a,b):
  return a/np.sqrt(E) + b

FitDF = pd.read_csv('Gamma_Peak_Stats_and_Params.csv', names=['Element', 'Fit type', 'Peak number',  'Peak type', 'Energy (keV)', 'Resolution', 'Min', 'Max', 'Mean', 'A', 'Sigma', 'Error' ,'a', 'b', 'chisq', 'Reduced chisq', 'Probchisq'],usecols=list(range(1,18)))

FitDF = FitDF[FitDF['Fit type'] == 'Linear']
FitDF = FitDF[FitDF['Peak type'] == 'Zoom']

FitDF['Energy (keV)'] = FitDF['Energy (keV)'].astype(float)
FitDF['Resolution'] = FitDF['Resolution'].astype(float).multiply(100)
SodiumDF = FitDF[(FitDF['Energy (keV)'] == 511)]
FitDF = FitDF[~(FitDF['Energy (keV)'] == 511)]


DataBa = FitDF[FitDF['Element'] == 'Barium133']
DataCo = FitDF[FitDF['Element'] == 'Cobalt60']
DataNa = FitDF[FitDF['Element'] == 'Sodium22']


Fit, Errors = sciopt.curve_fit(ResolutionFit,FitDF['Energy (keV)'],FitDF['Resolution'])
print(Fit)

TitleFont = {'size':'20', 'color':'black', 'weight':'bold'}
AxTitleFont = {'size':'18'}

plt.figure(1)
Energies = np.linspace(55,1400,14000)
plt.plot(Energies,ResolutionFit(Energies,*Fit),label="Fit")

plt.errorbar(DataBa['Energy (keV)'],DataBa['Resolution'],fmt="o",color='red',label="Barium-133",markersize=7,yerr=(7.05/DataBa['Energy (keV)']),elinewidth=3,capsize=5,capthick=3,marker='o')
plt.errorbar(DataCo['Energy (keV)'],DataCo['Resolution'],fmt="o",color='blue',label="Cobalt-60",markersize=7,yerr=(7.05/DataCo['Energy (keV)']),elinewidth=3,capsize=5,capthick=3,marker='o')
plt.errorbar(DataNa['Energy (keV)'],DataNa['Resolution'],fmt="o",color='yellow',label="Sodium-22",markersize=7,yerr=(7.05/DataNa['Energy (keV)']),elinewidth=3,capsize=5,capthick=3,marker='o')
plt.errorbar(SodiumDF['Energy (keV)'],SodiumDF['Resolution'],fmt="o",color='yellow',label="",markersize=7,yerr=(7.05/SodiumDF['Energy (keV)']),elinewidth=3,capsize=5,capthick=3,marker='o')
plt.xlabel('Peak Energy [keV]',**AxTitleFont)
plt.ylabel('Energy Resolution [%]',**AxTitleFont)
```

```
plt.title('Energy Resolution as a Function of Gamma Energy',**TitleFont)
plt.legend(fontsize=16,fancybox=True,shadow=False)

plt.figure(2)
plt.errorbar(DataBa['Energy (keV)'],DataBa['Resolution']-ResolutionFit(DataBa['Energy (keV)'],*Fit),fmt="o",color='red',label="Barium-133",markersize=7,yerr=(7.05/DataBa['Energy (keV)']),elinewidth=3,capsize=5,capthick=3,marker='o')
plt.errorbar(DataCo['Energy (keV)'],DataCo['Resolution']-ResolutionFit(DataCo['Energy (keV)'],*Fit),fmt="o",color='blue',label="Cobalt-60",markersize=7,yerr=(7.05/DataCo['Energy (keV)']),elinewidth=3,capsize=5,capthick=3,marker='o')
plt.errorbar(DataNa['Energy (keV)'],DataNa['Resolution']-ResolutionFit(DataNa['Energy (keV)'],*Fit),fmt="o",color='yellow',label="Sodium-22",markersize=7,yerr=(7.05/DataNa['Energy (keV)']),elinewidth=3,capsize=5,capthick=3,marker='o')
plt.errorbar(SodiumDF['Energy (keV)'],SodiumDF['Resolution']-ResolutionFit(SodiumDF['Energy (keV)'],*Fit),fmt="o",color='yellow',label="",markersize=7,yerr=(7.05/SodiumDF['Energy (keV)']),elinewidth=3,capsize=5,capthick=3,marker='o')
plt.xlabel('Peak Energy [keV]',**AxTitleFont)
plt.ylabel('Energy Resolution Residual [%]',**AxTitleFont)
plt.title('Energy Resolution Residuals',**TitleFont)
plt.legend(fontsize=16,fancybox=True,shadow=False)
plt.show()

def ChiSqFunc(Measured,Fitted,Errors,Params):
  ChiSquared = 0
  for i in range(len(Measured)):
    ChiSquared += ((Measured[i] - Fitted[i])**2.0) / ((Errors[i])**2.0)
  ReducedChiSq = ChiSquared/(len(Measured)-len(Params))
  ProbChiSq = (1.0 - scistats.chi2.cdf(ChiSquared,len(Measured)-len(Params)) )  * 100.0
  return ChiSquared, ReducedChiSq, ProbChiSq


MeasuredThungys = list(DataBa['Resolution'])+list(DataCo['Resolution']) + list(DataNa['Resolution'])
FittedThingys = list(ResolutionFit(DataBa['Energy (keV)'],*Fit)) + list(ResolutionFit(DataCo['Energy (keV)'],*Fit)) + list(ResolutionFit(DataNa['Energy (keV)'],*Fit))
ErrorThingys = list(7.05/DataBa['Energy (keV)']) + list(7.05/DataCo['Energy (keV)']) + list(7.05/DataBa['Energy (keV)'])

ChiStats = ChiSqFunc(MeasuredThungys, FittedThingys, ErrorThingys,Fit)
print(ChiStats)
```