HPGe Detector Appendix

Alessandro Maraio & Samuel Morgan

Abstract

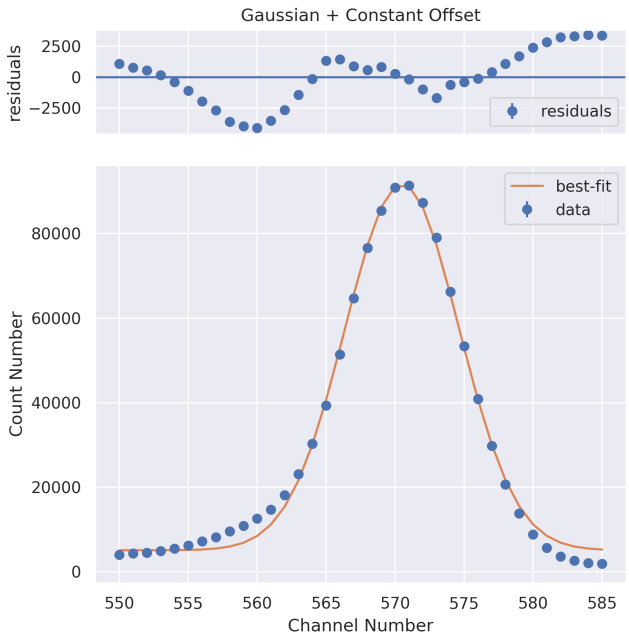
The objective of this project was to identify an unknown gamma ray source by using a high-purity germanium (HPGe) detector. We calibrated the detector by using three known sources, ^{22}Na, ^{60}Co & ^{133}Ba, and then measured the energy spectrum from the unknown source. We determined the unknown source 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.

A.1 Fitted Peaks

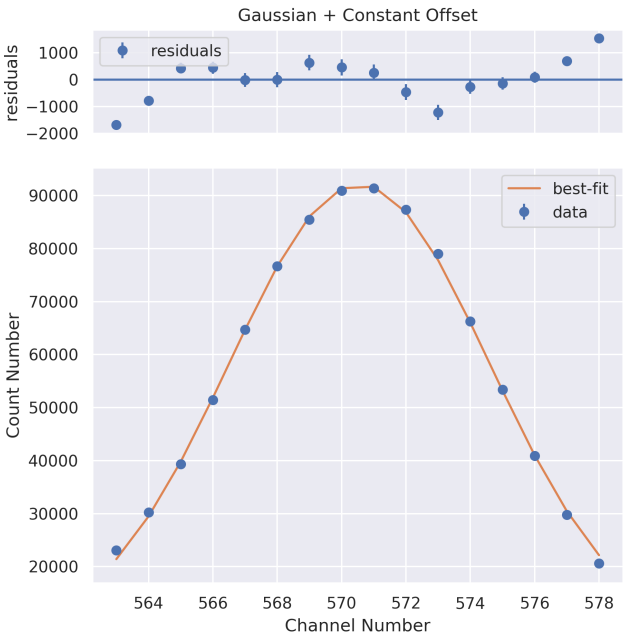
All of these are for the 24 hour data runs, hence the high count numbers

A.1.1 Barium-133

A.1.1.1 Gaussian Fit Only

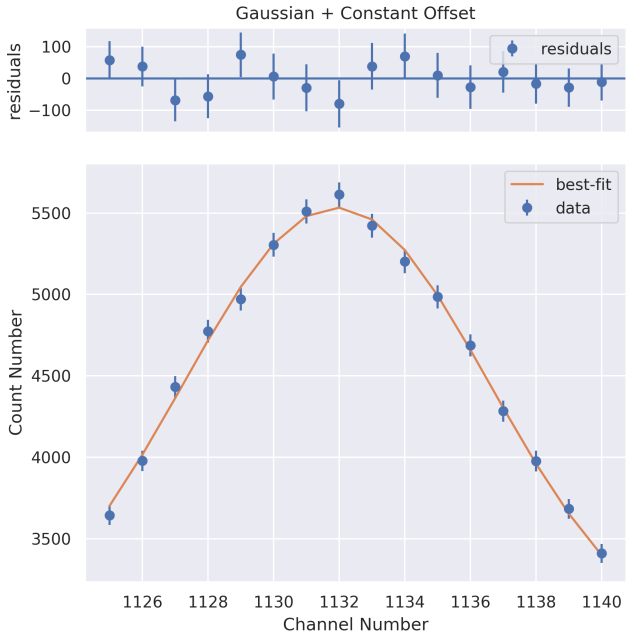


(a) Full peak with fit

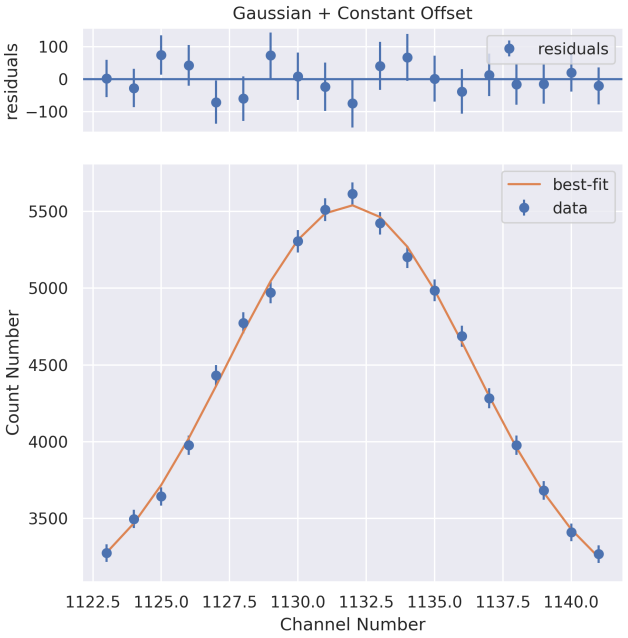


(b) Zoomed in peak with fit

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

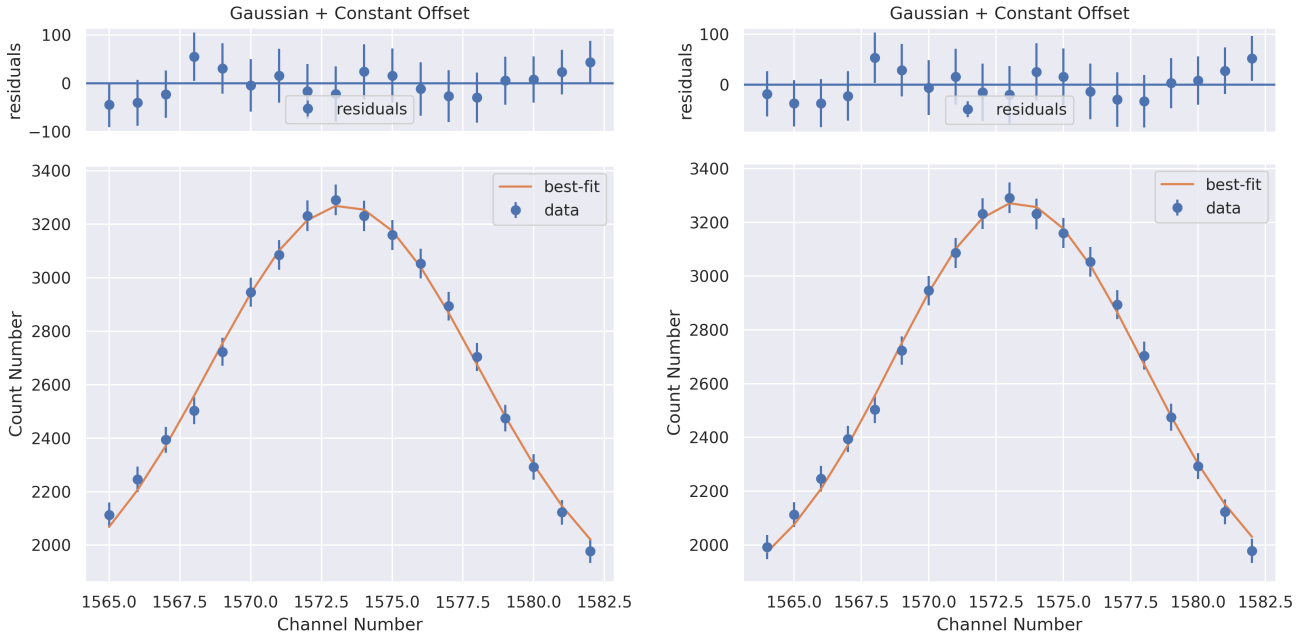


(a) Full peak with fit



(b) Zoomed in peak with fit

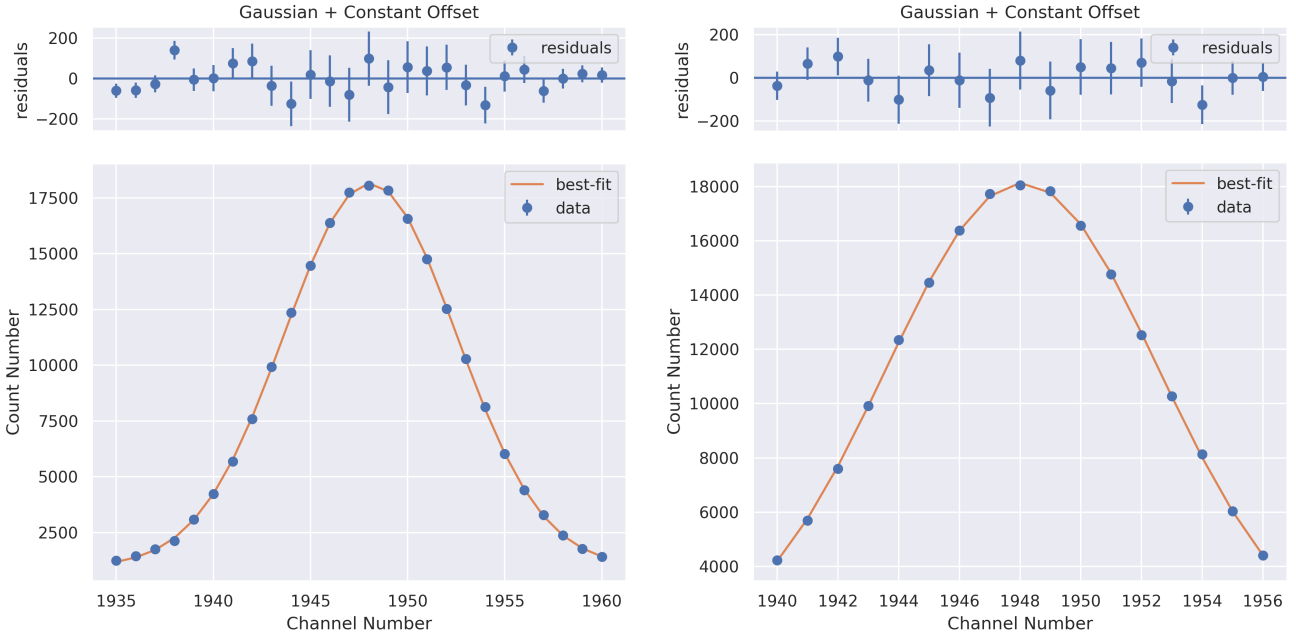
Figure A.2: Fit of full & zoomed in peak of ^{133}Ba 161 keV peak



(a) Full peak with fit

(b) Zoomed in peak with fit

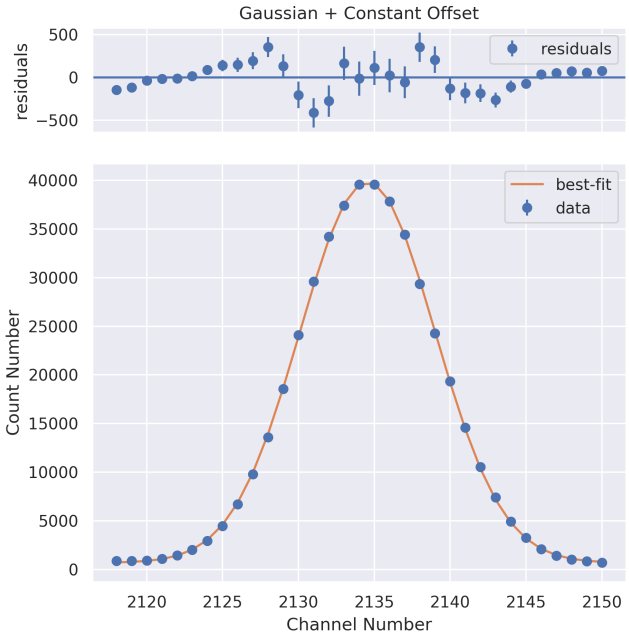
Figure A.3: Fit of full & zoomed in peak of ^{133}Ba 223 keV peak



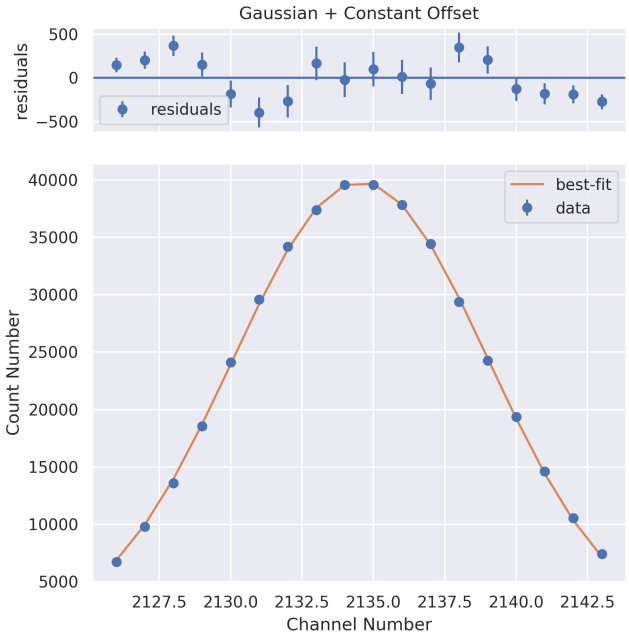
(a) Full peak with fit

(b) Zoomed in peak with fit

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

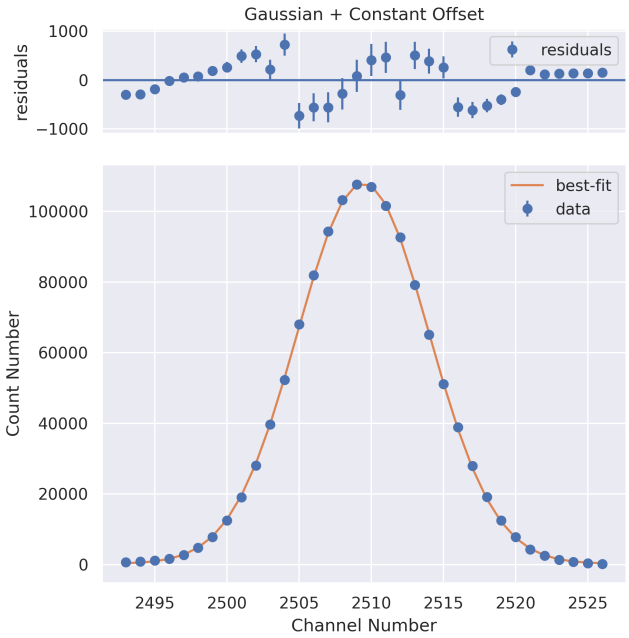


(a) Full peak with fit

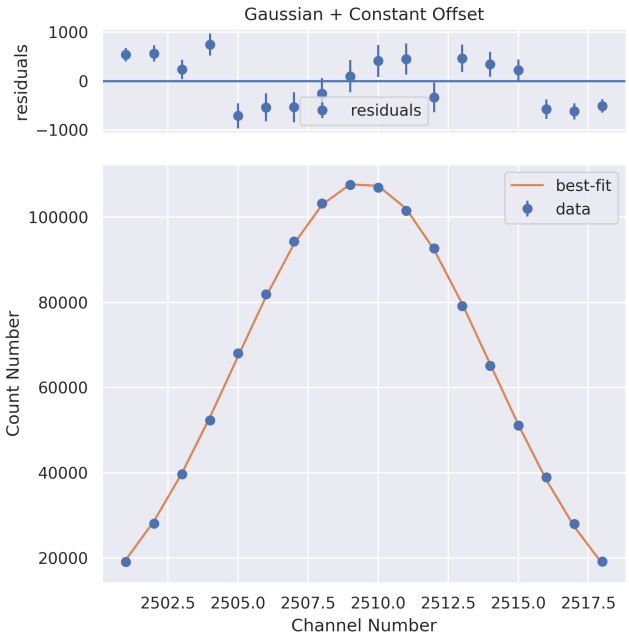


(b) Zoomed in peak with fit

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

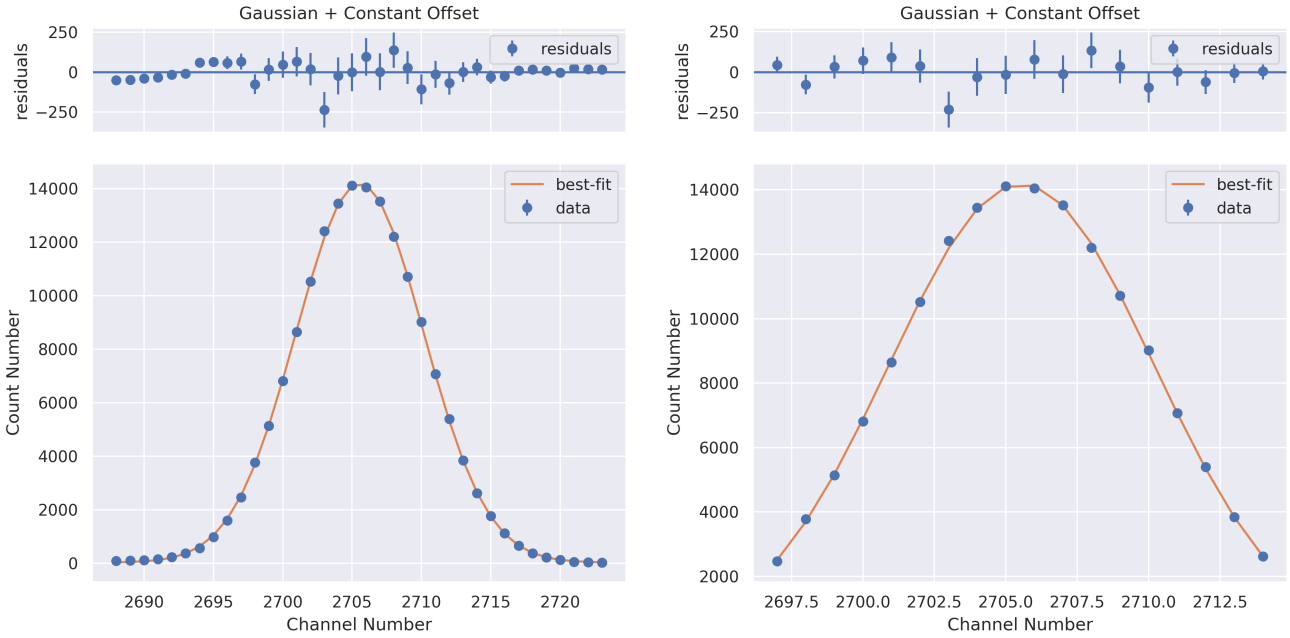


(a) Full peak with fit



(b) Zoomed in peak with fit

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

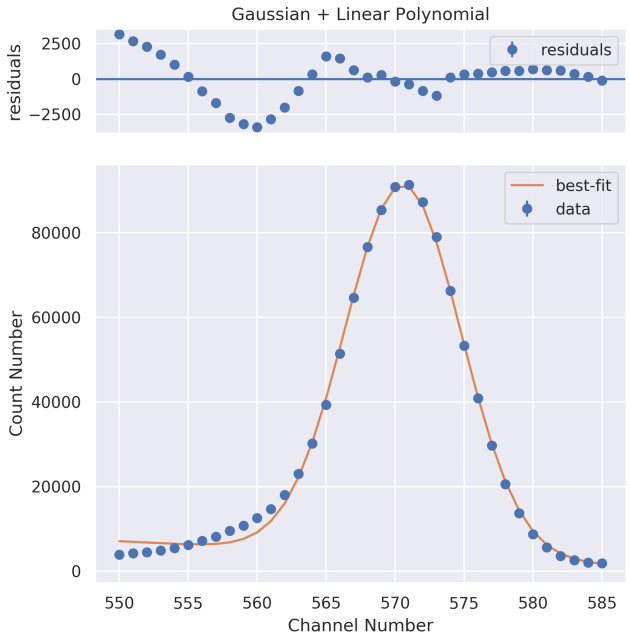


(a) Full peak with fit

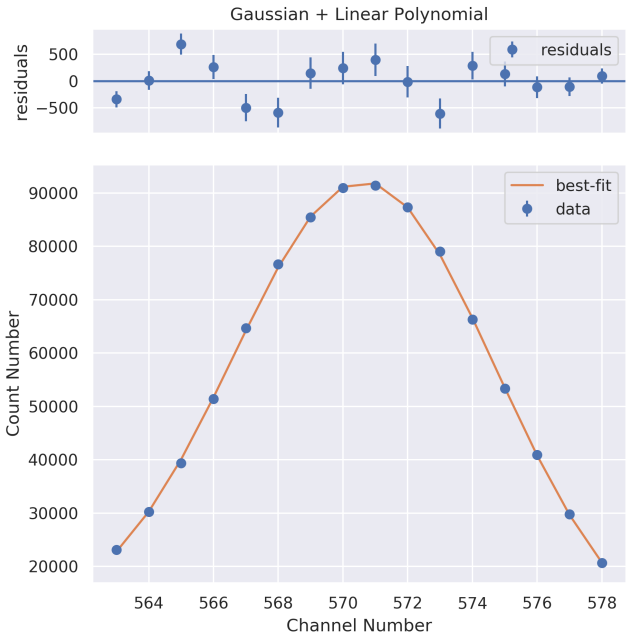
(b) Zoomed in peak with fit

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

A.1.1.2 Linear + Gaussian Fit

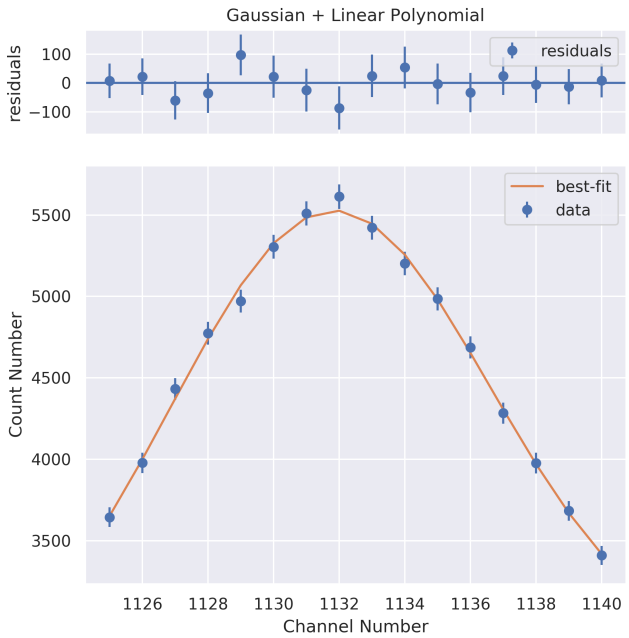


(a) Full peak with fit

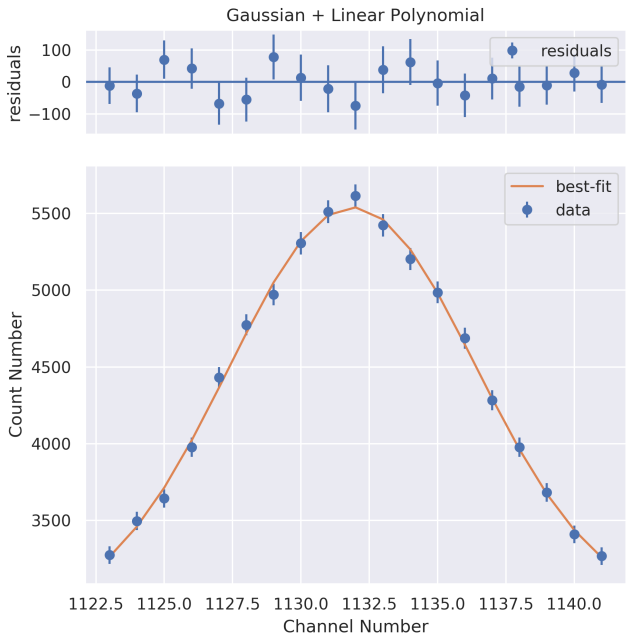


(b) Zoomed in peak with fit

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

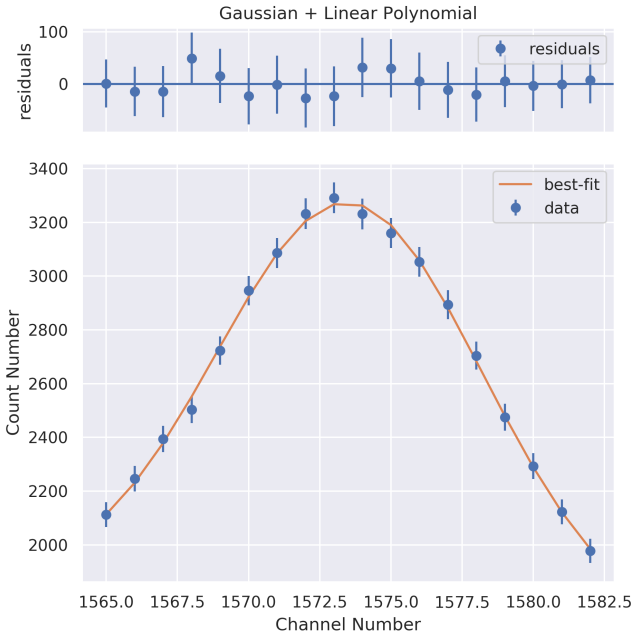


(a) Full peak with fit

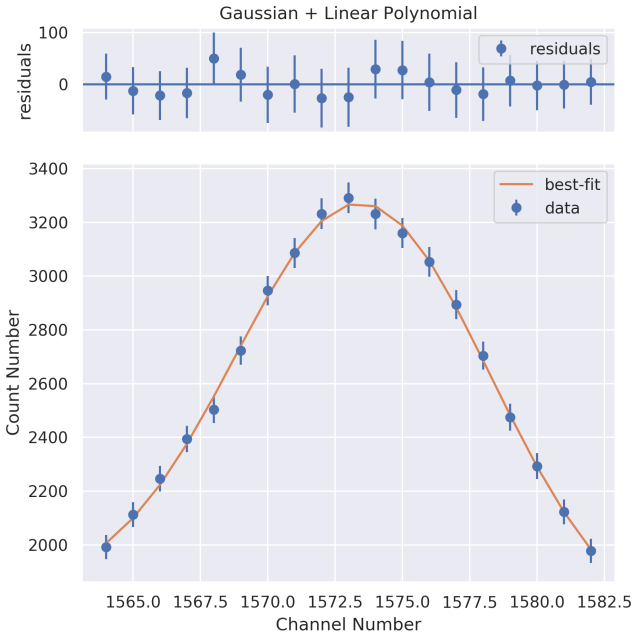


(b) Zoomed in peak with fit

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

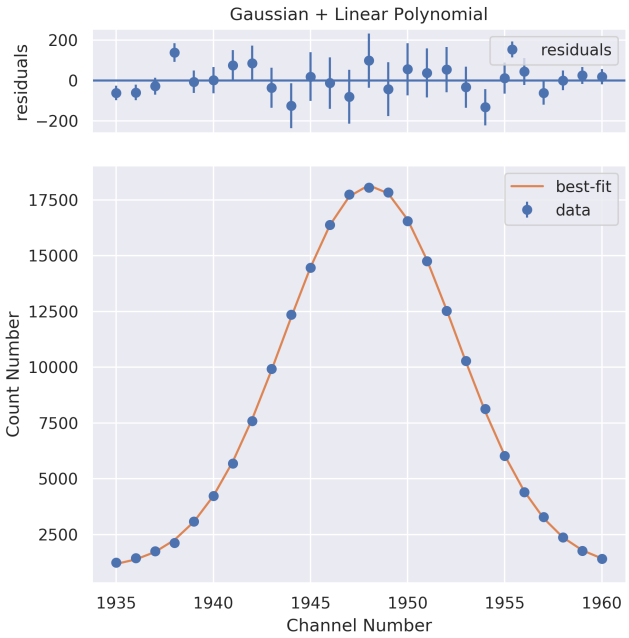


(a) Full peak with fit

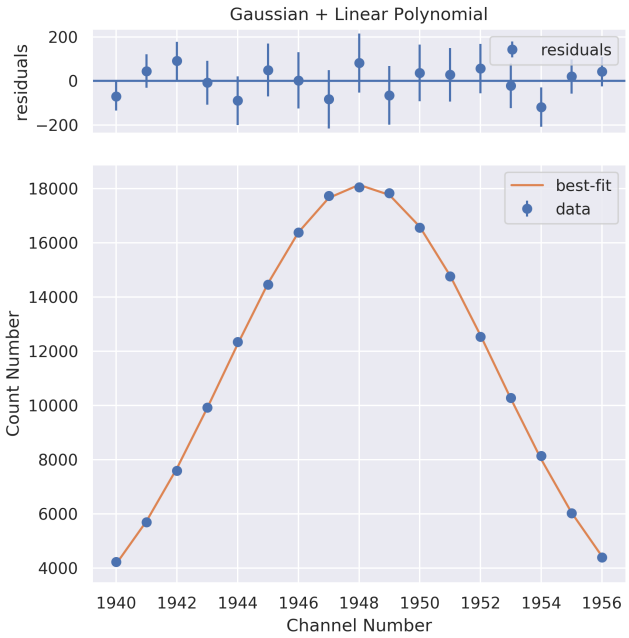


(b) Zoomed in peak with fit

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

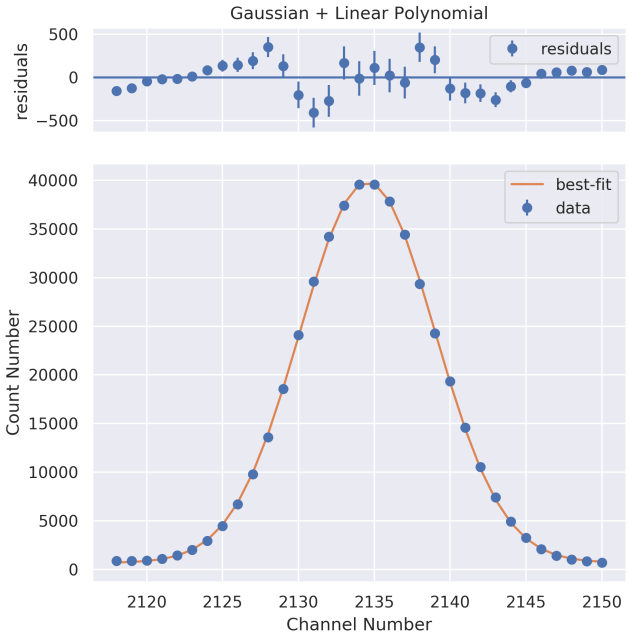


(a) Full peak with fit

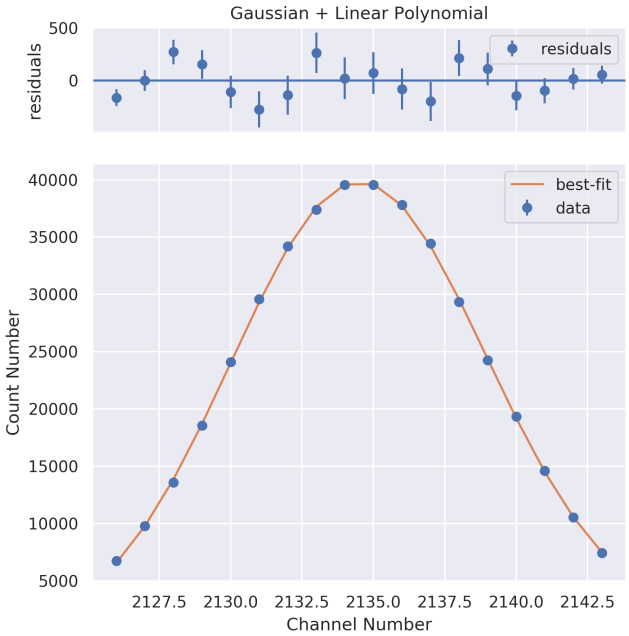


(b) Zoomed in peak with fit

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

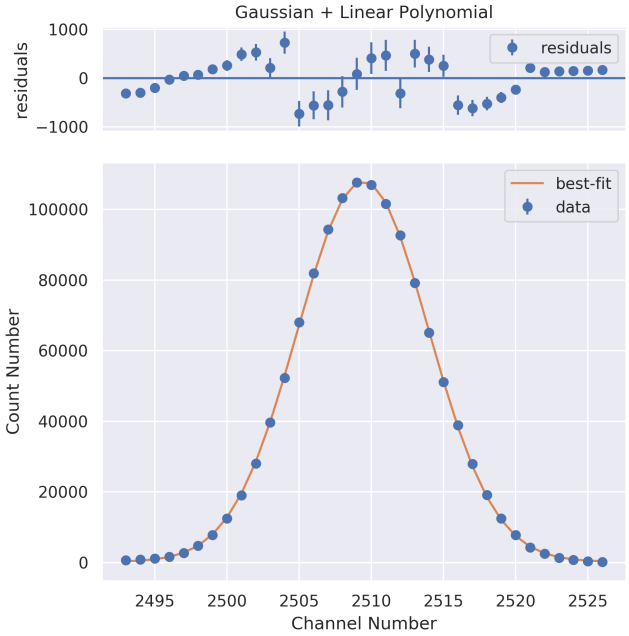


(a) Full peak with fit

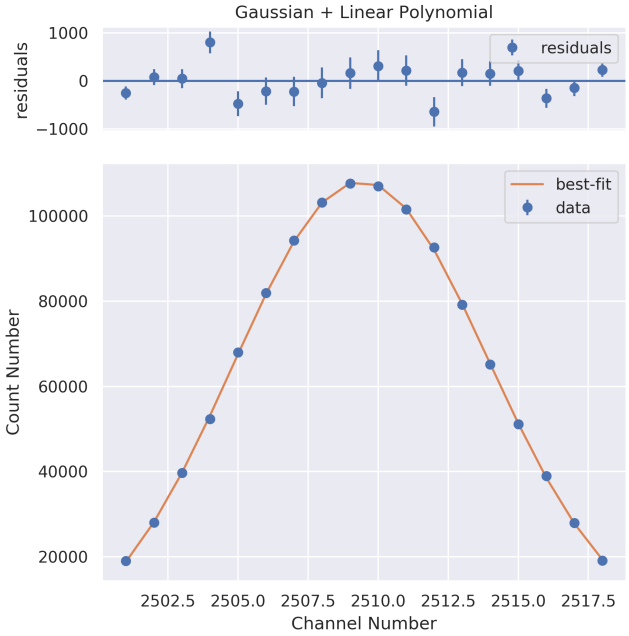


(b) Zoomed in peak with fit

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

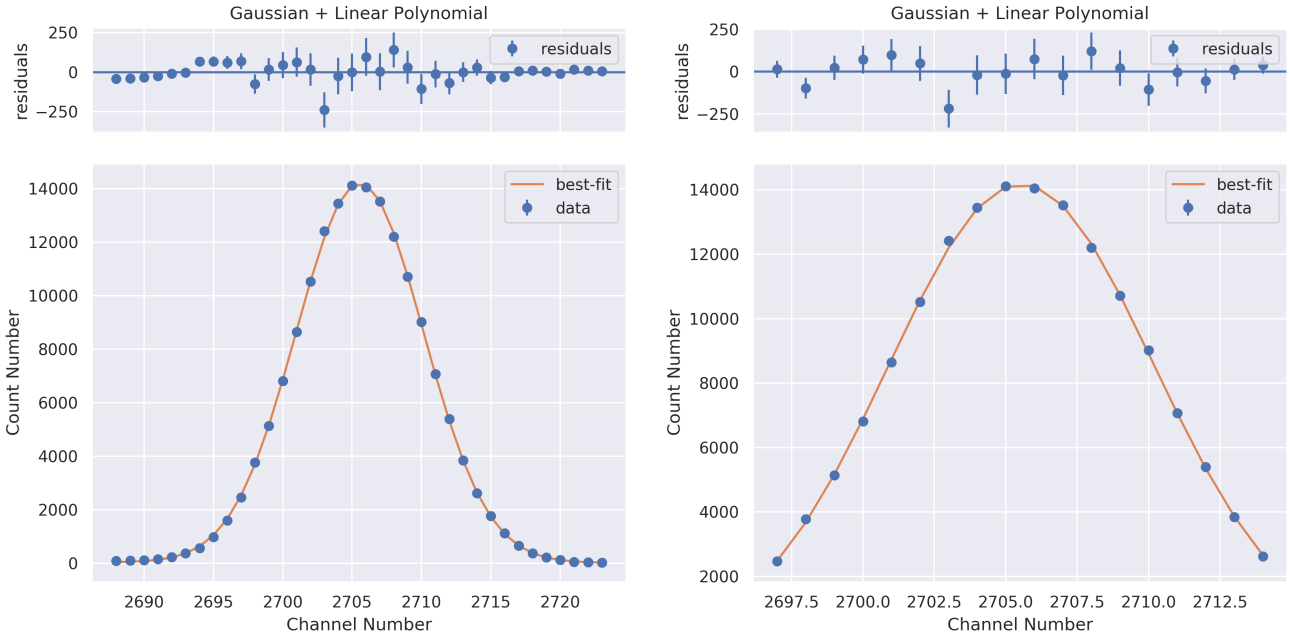


(a) Full peak with fit



(b) Zoomed in peak with fit

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

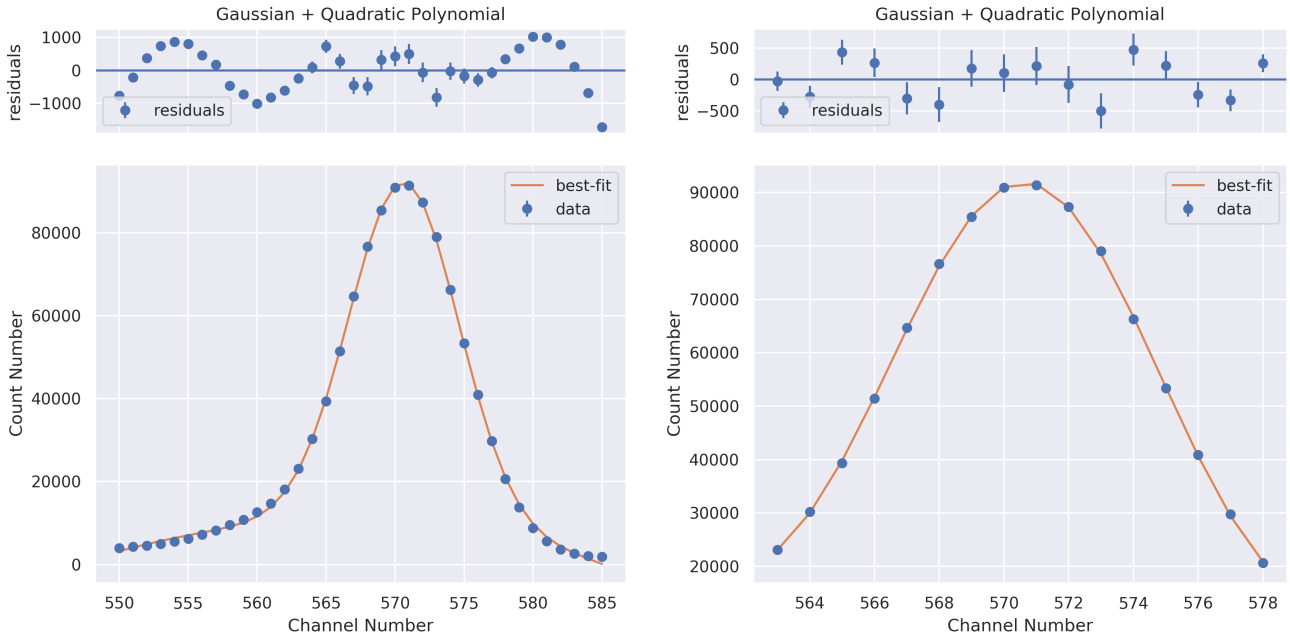


(a) Full peak with fit

(b) Zoomed in peak with fit

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

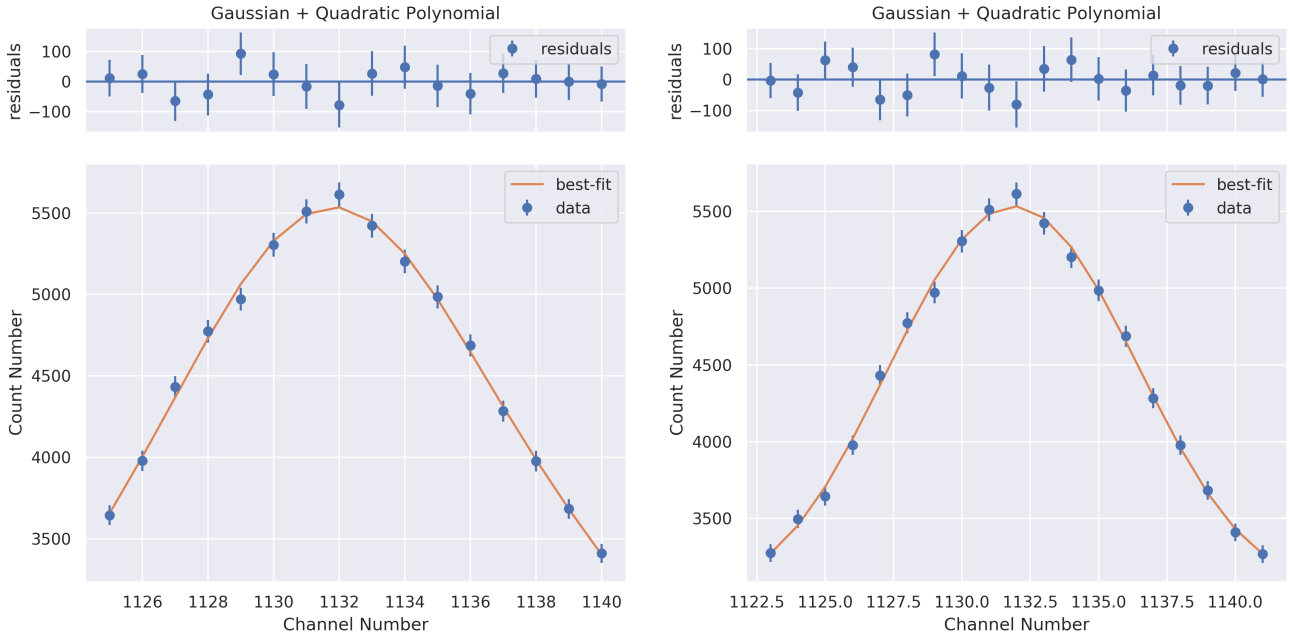
A.1.1.3 Quadratic + Gaussian Fit



(a) Full peak with fit

(b) Zoomed in peak with fit

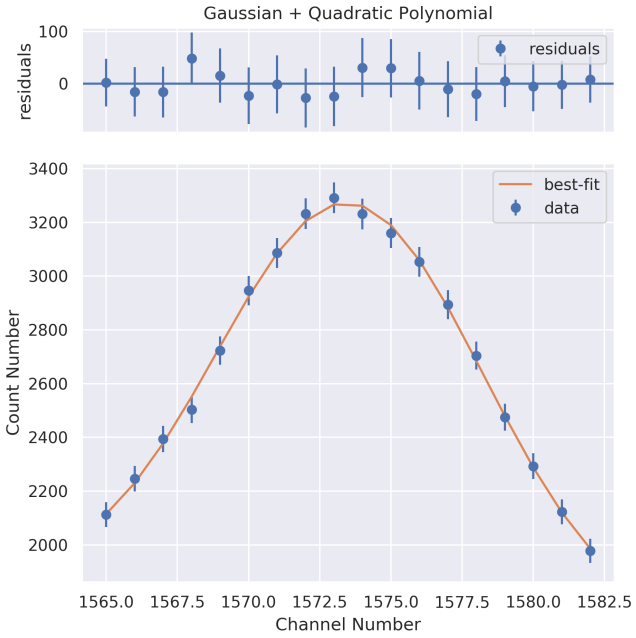
Figure A.15: Fit of full & zoomed in peak of ^{133}Ba 81 keV peak



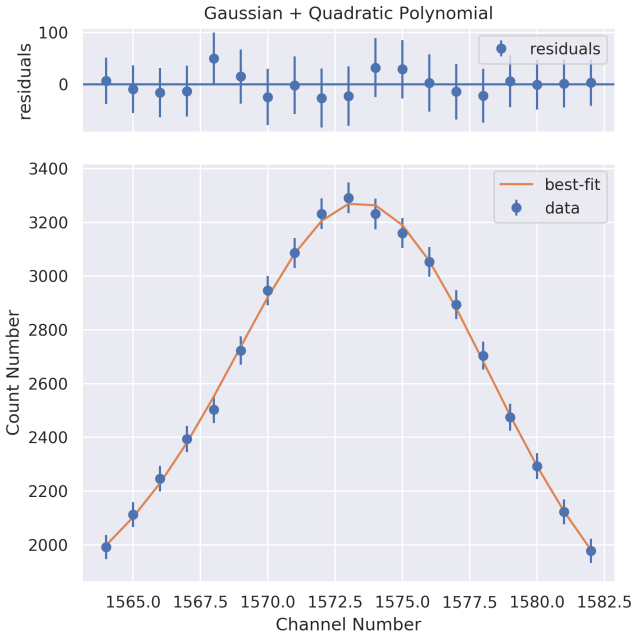
(a) Full peak with fit

(b) Zoomed in peak with fit

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

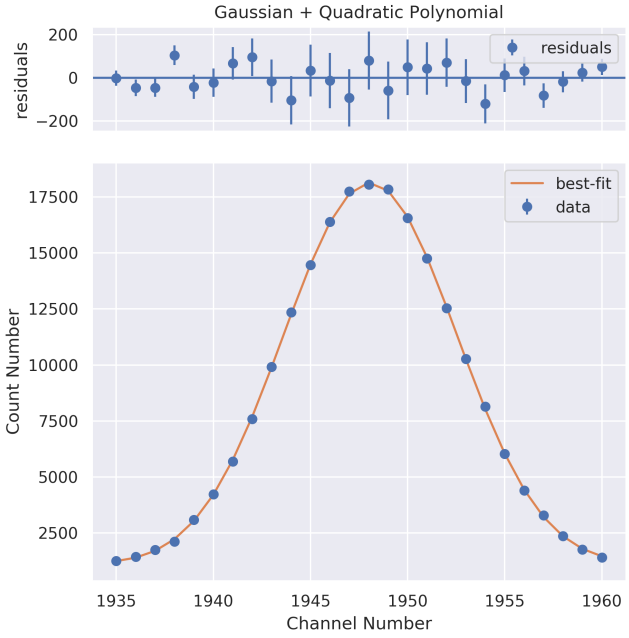


(a) Full peak with fit

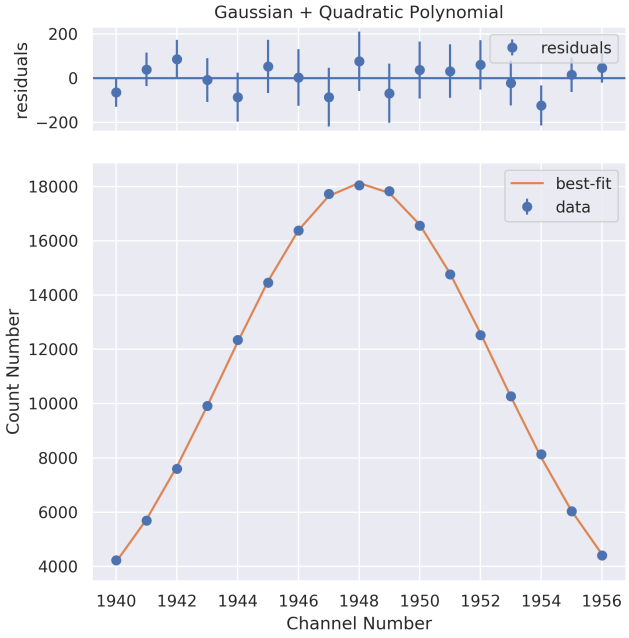


(b) Zoomed in peak with fit

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

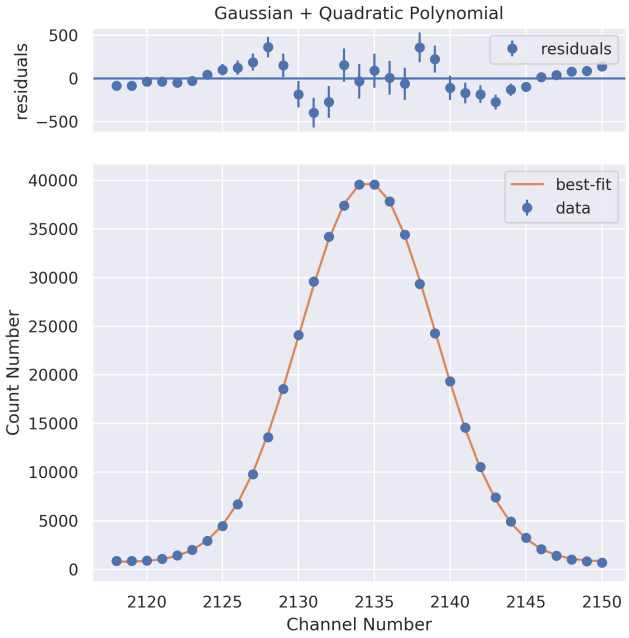


(a) Full peak with fit

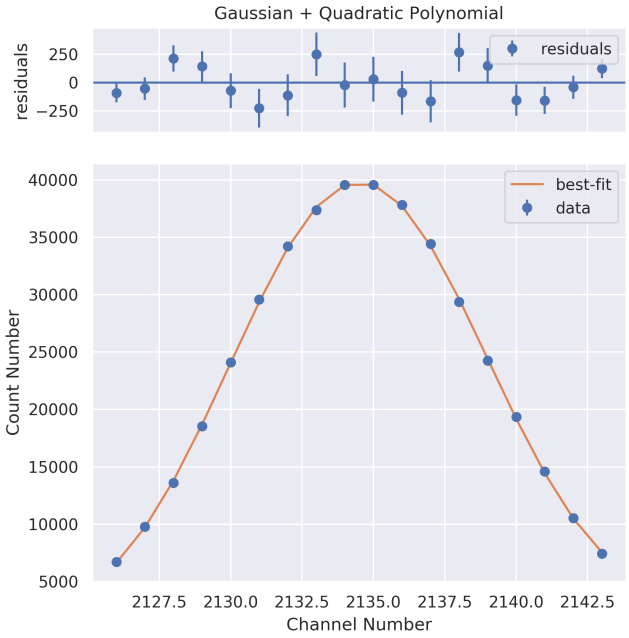


(b) Zoomed in peak with fit

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

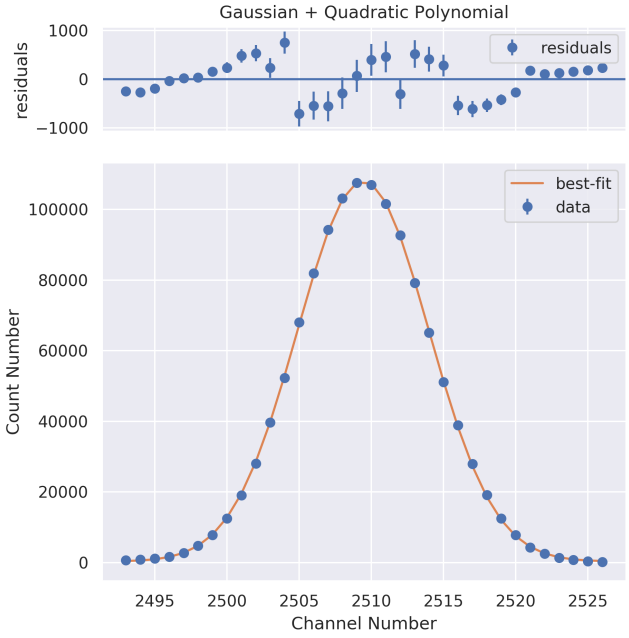


(a) Full peak with fit

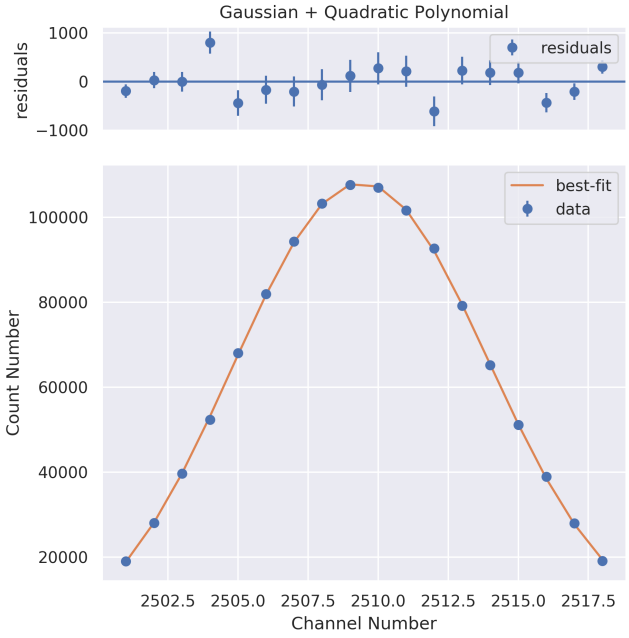


(b) Zoomed in peak with fit

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

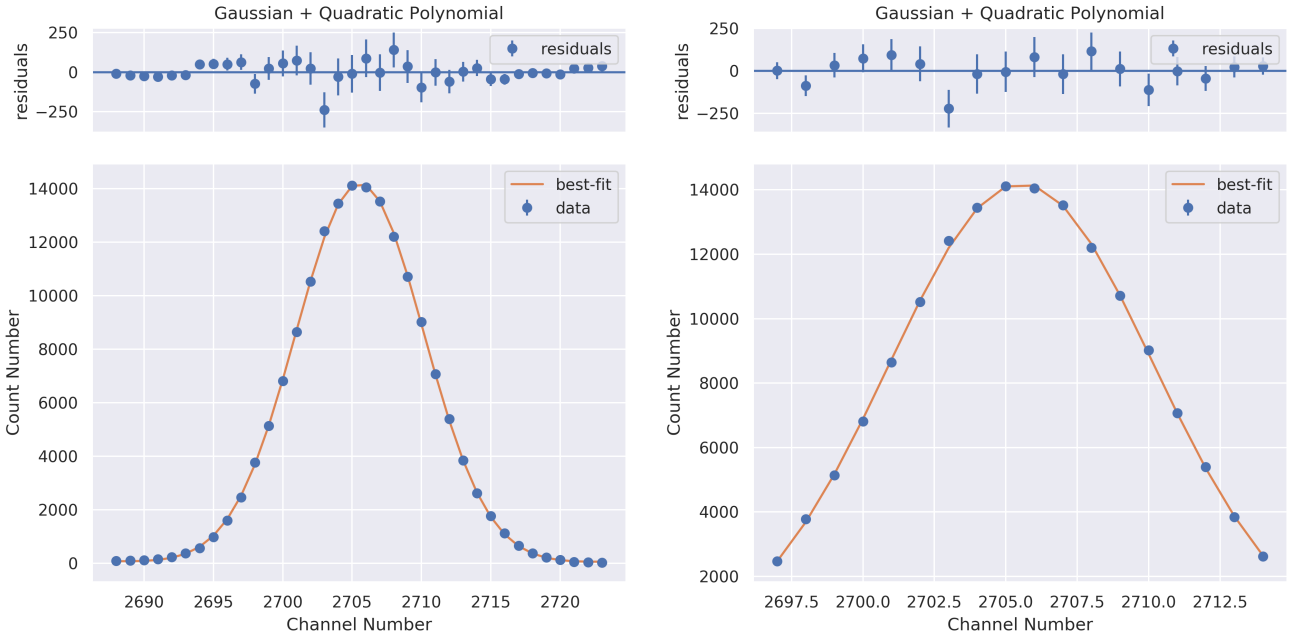


(a) Full peak with fit



(b) Zoomed in peak with fit

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



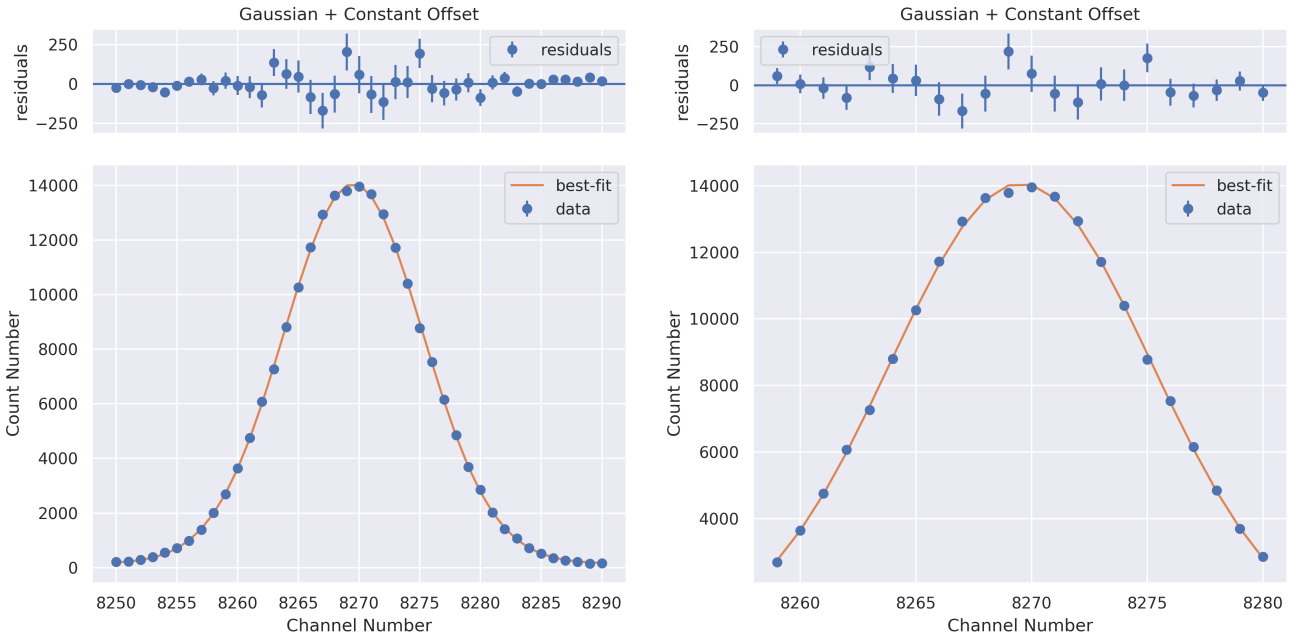
(a) Full peak with fit

(b) Zoomed in peak with fit

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

A.1.2 Cobalt-60

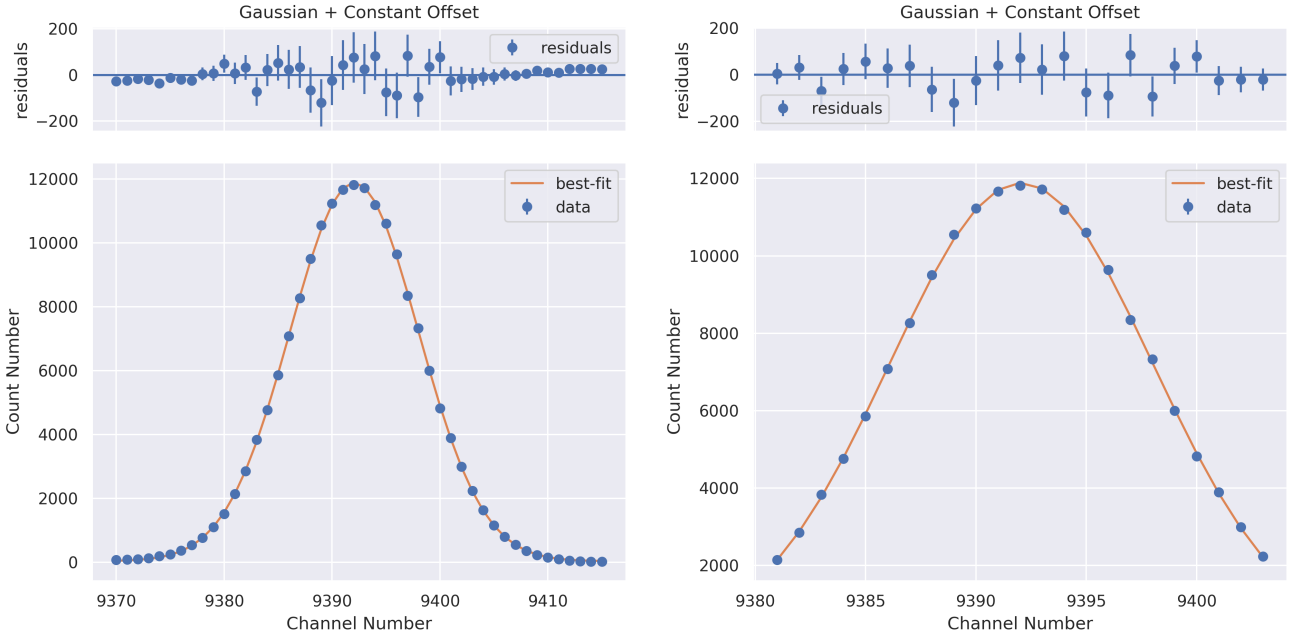
A.1.2.1 Gaussian Fit Only



(a) Full peak with fit

(b) Zoomed in peak with fit

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

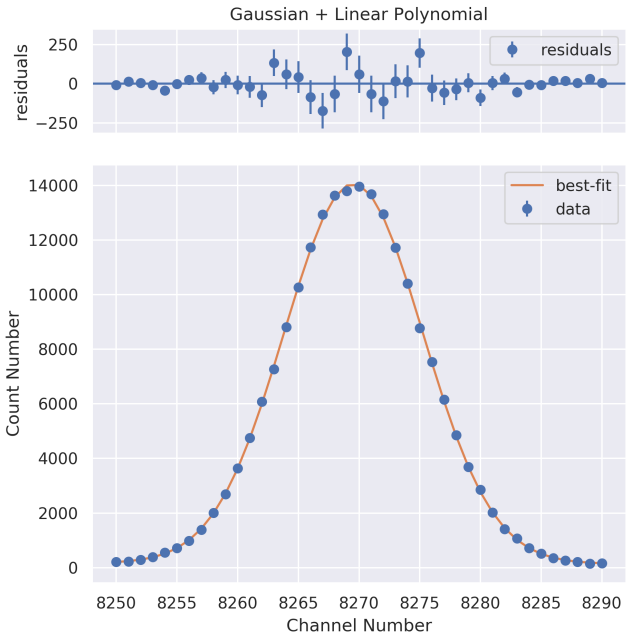


(a) Full peak with fit

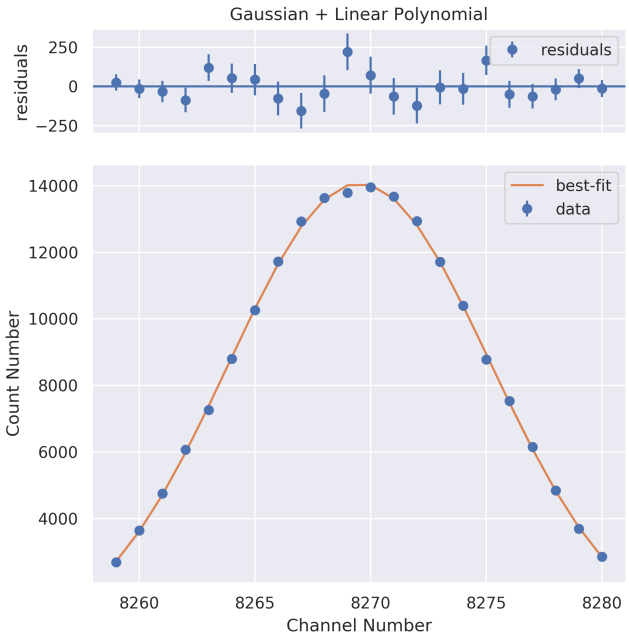
(b) Zoomed in peak with fit

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

A.1.2.2 Linear + Gaussian Fit

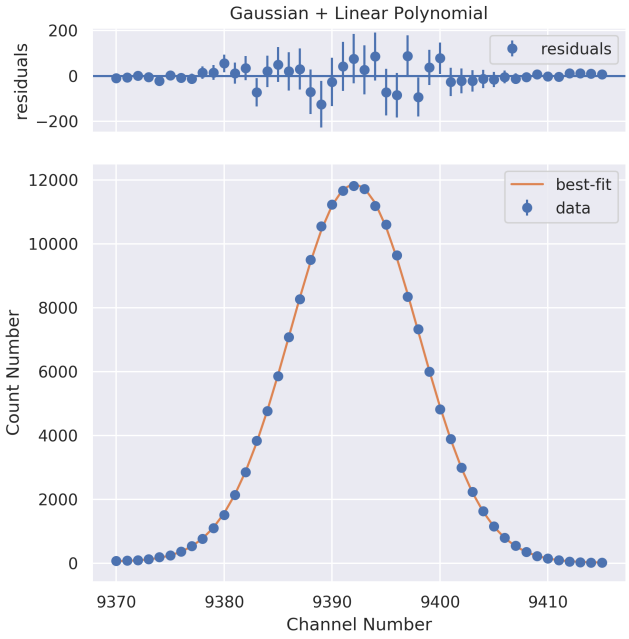


(a) Full peak with fit

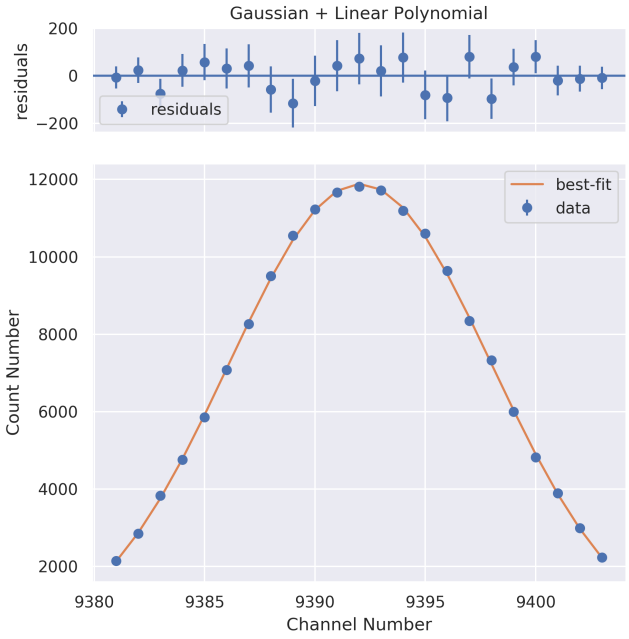


(b) Zoomed in peak with fit

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



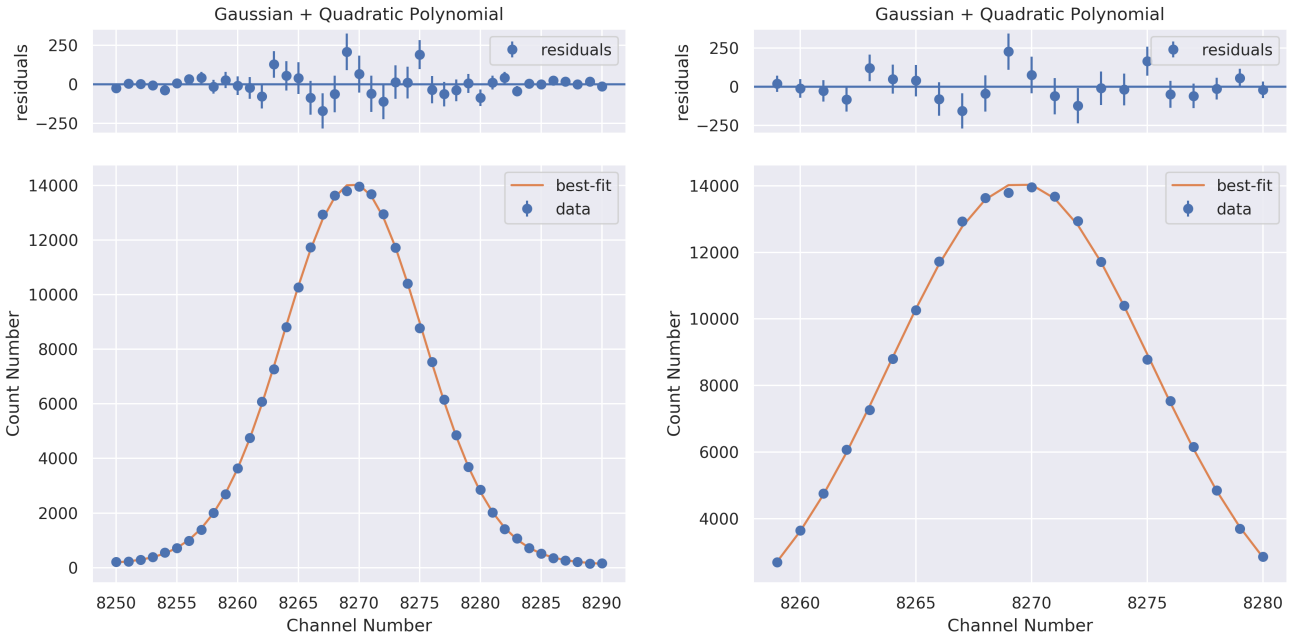
(a) Full peak with fit



(b) Zoomed in peak with fit

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

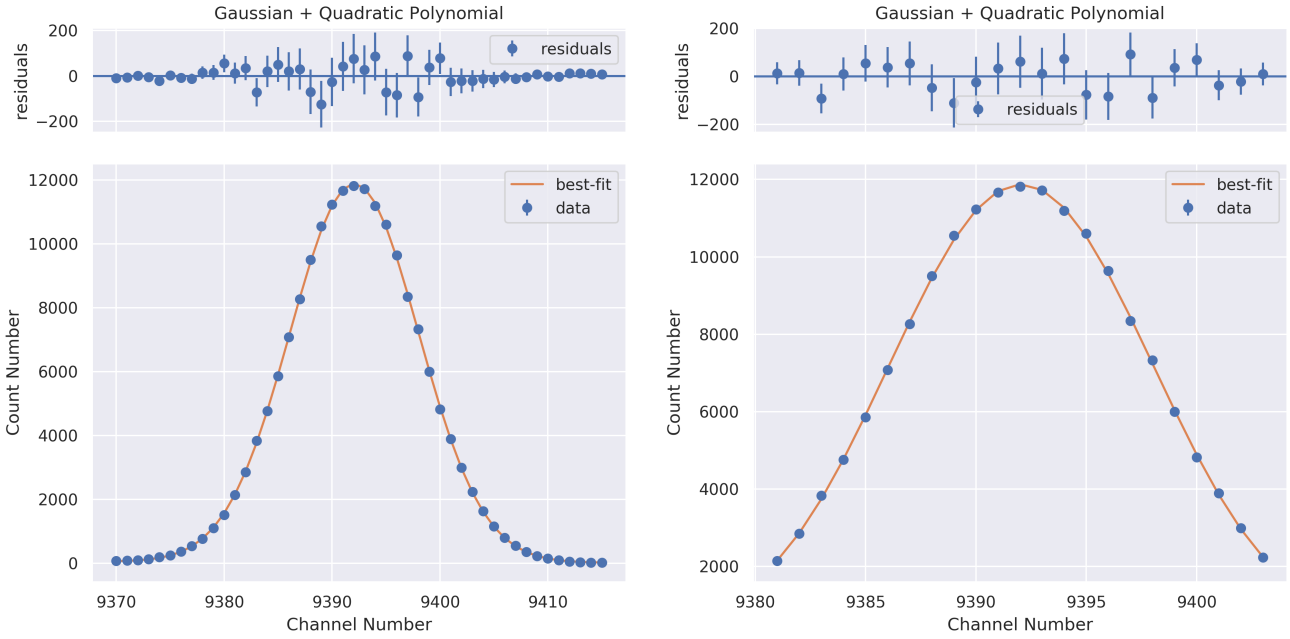
A.1.2.3 Quadratic + Gaussian Fit



(a) Full peak with fit

(b) Zoomed in peak with fit

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



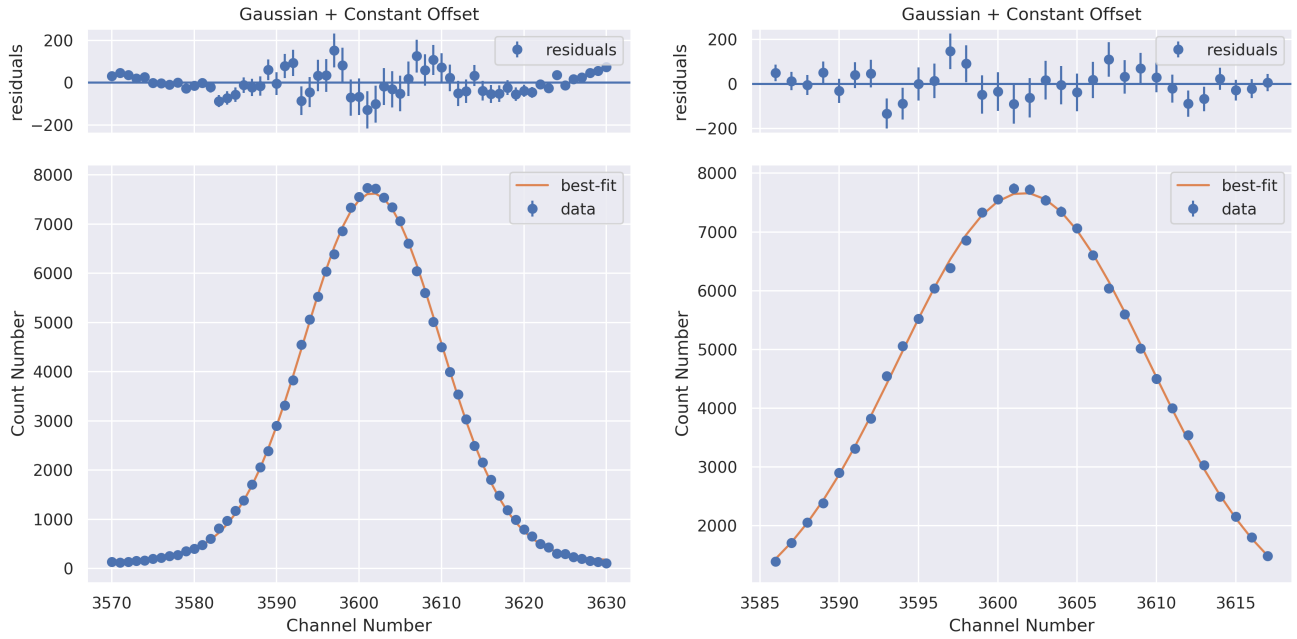
(a) Full peak with fit

(b) Zoomed in peak with fit

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

A.1.3 Sodium-22

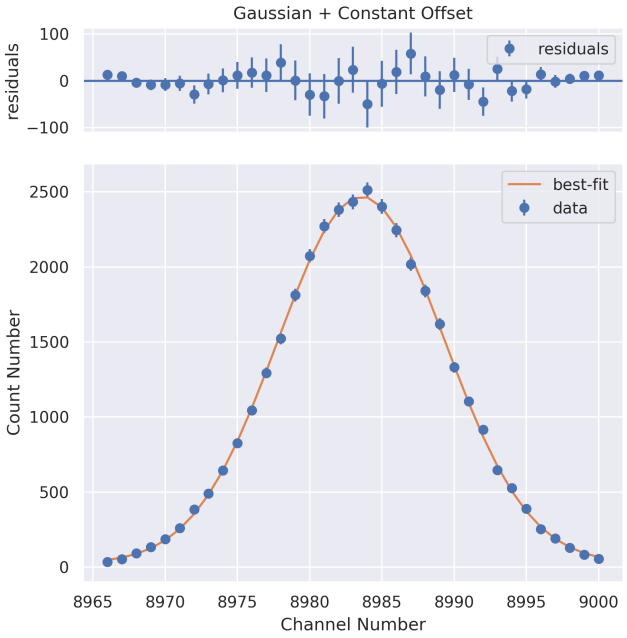
A.1.3.1 Gaussian Fit Only



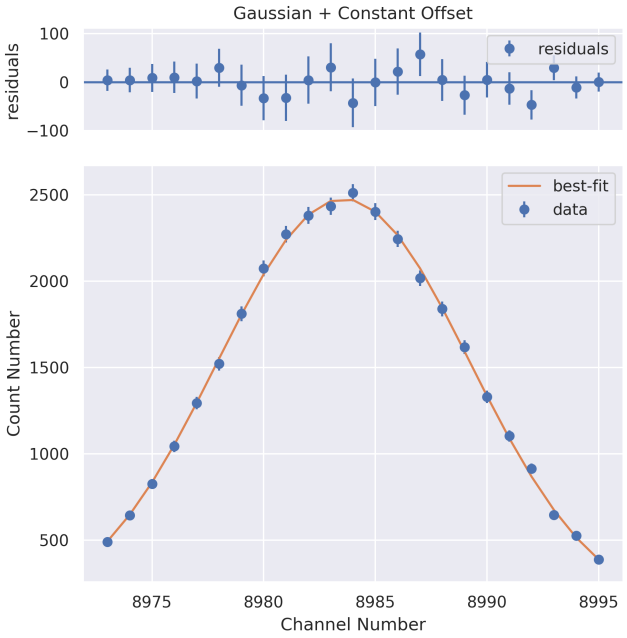
(a) Full peak with fit

(b) Zoomed in peak with fit

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



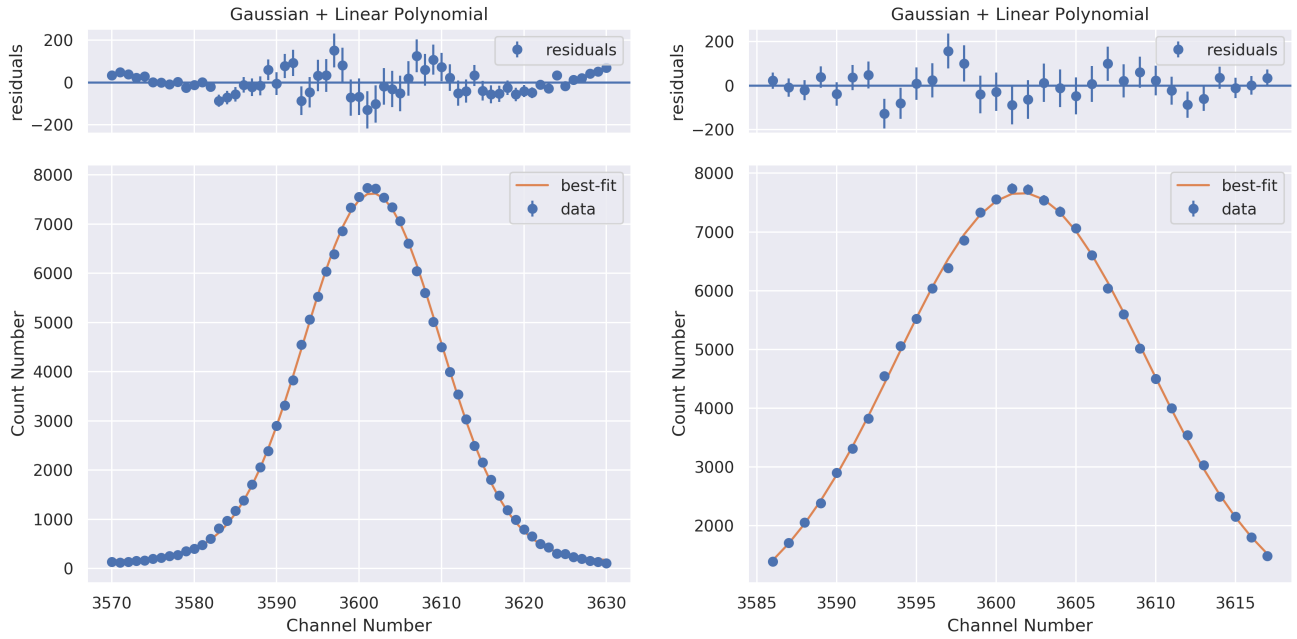
(a) Full peak with fit



(b) Zoomed in peak with fit

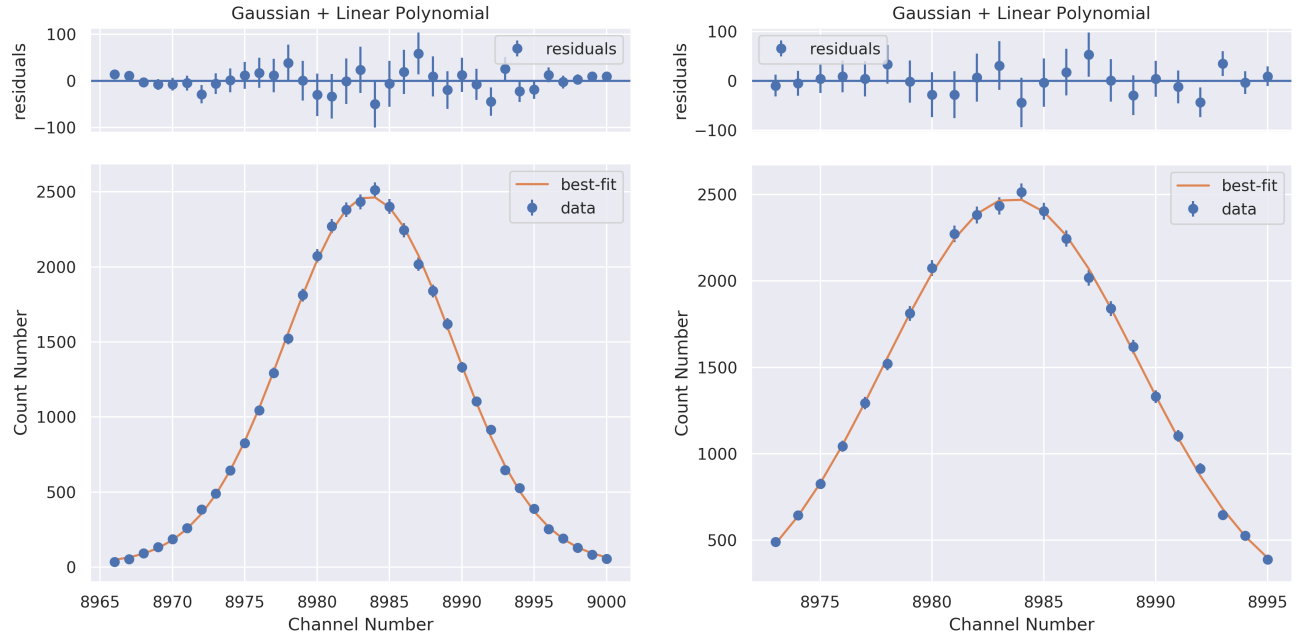
Figure A.29: Fit of full & zoomed in peak of ^{22}Na 1274 keV peak

A.1.3.2 Linear + Gaussian Fit



(a) Full peak with fit **(b)** Zoomed in peak with fit

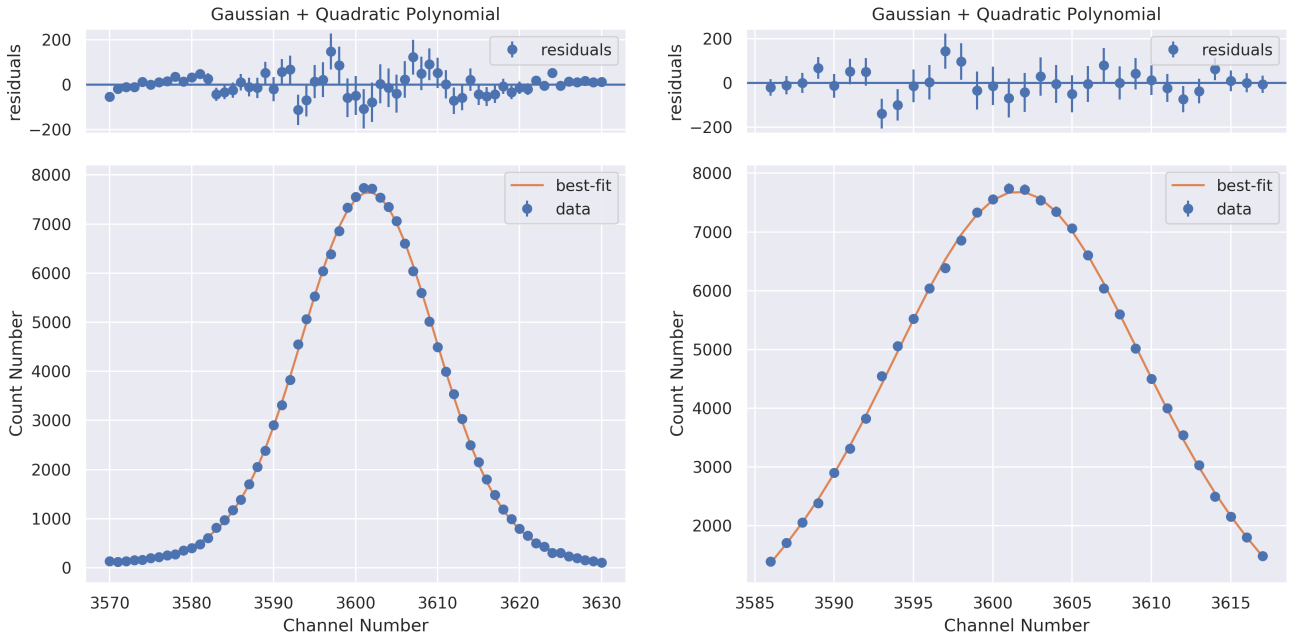
Figure A.30: Fit of full & zoomed in peak of ^{60}Na 511 keV peak



(a) Full peak with fit **(b)** Zoomed in peak with fit

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

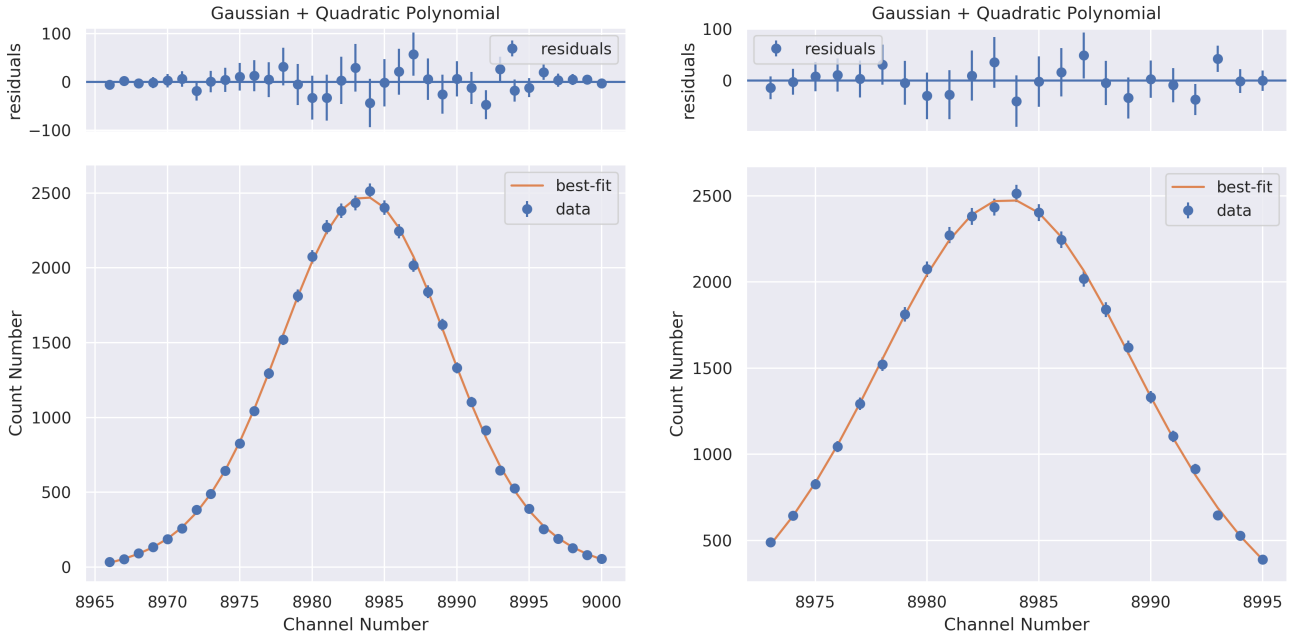
A.1.3.3 Quadratic + Gaussian Fit



(a) Full peak with fit

(b) Zoomed in peak with fit

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

A.2 Fit Comparisons

A.2.1 Barium-133

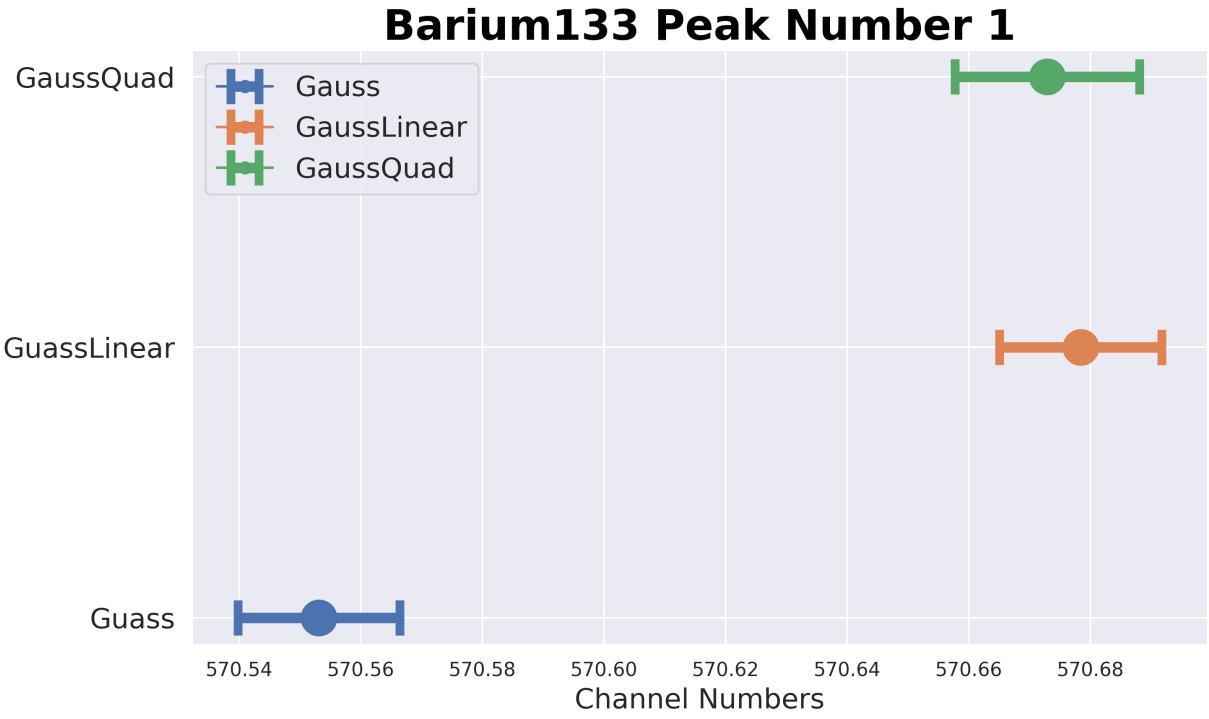


Figure A.34: Fit comparisons for ^{133}Ba 81 keV peak

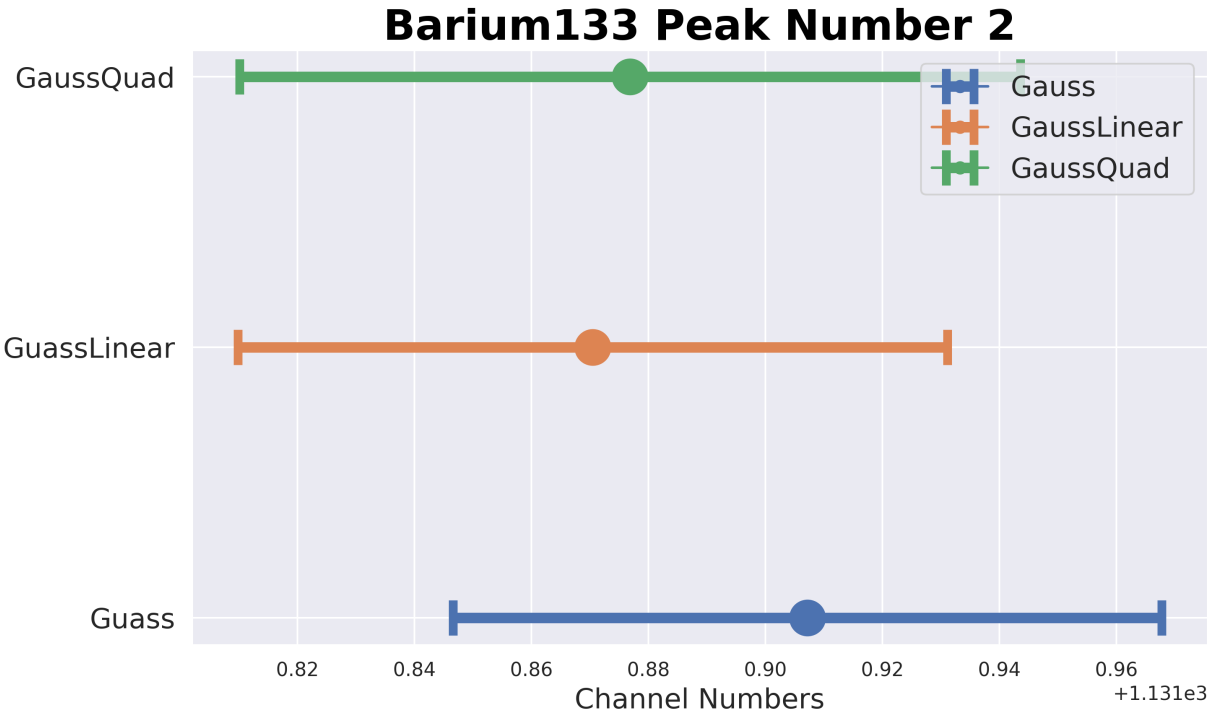


Figure A.35: Fit comparisons for ^{133}Ba 161 keV peak

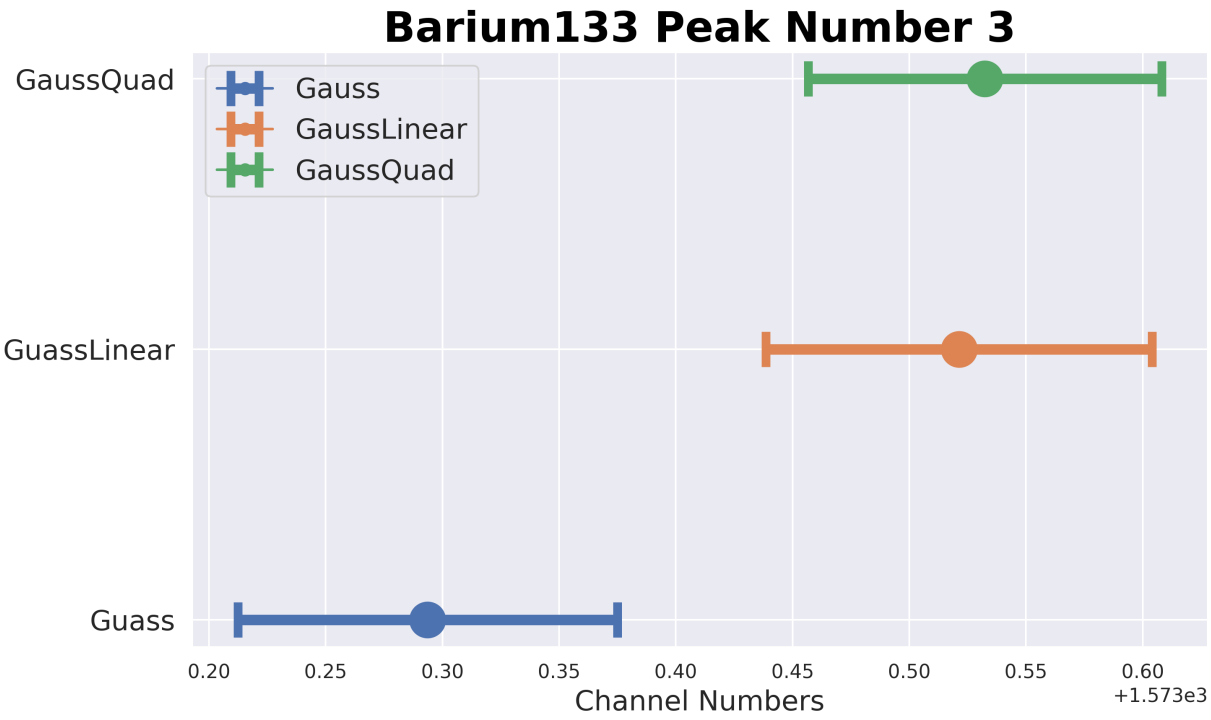


Figure A.36: Fit comparisons for ^{133}Ba 223 keV peak

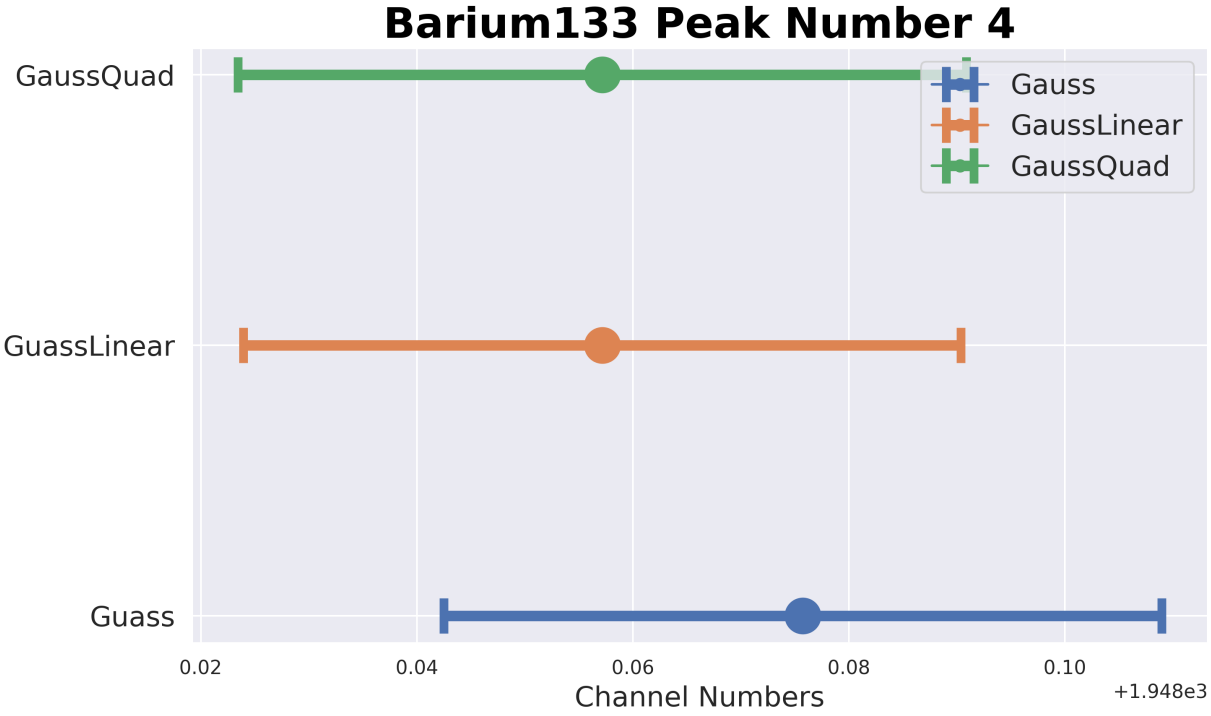


Figure A.37: Fit comparisons for ^{133}Ba 276 keV peak

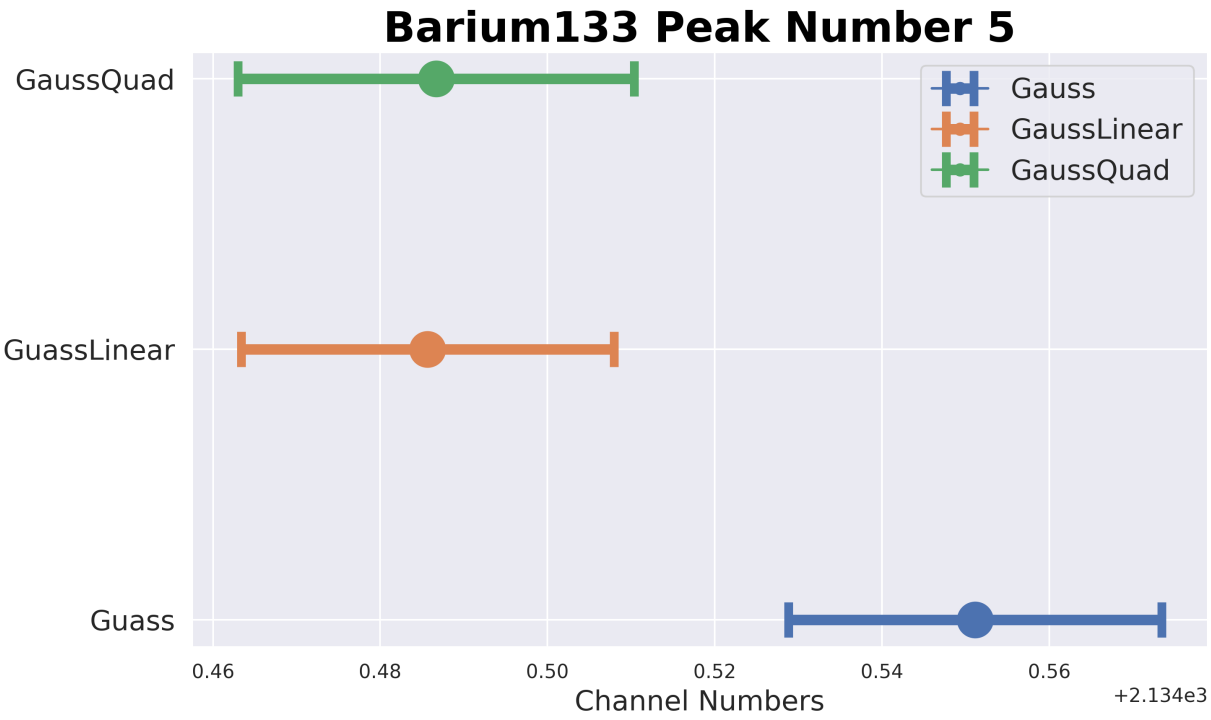


Figure A.38: Fit comparisons for ^{133}Ba 303 keV peak

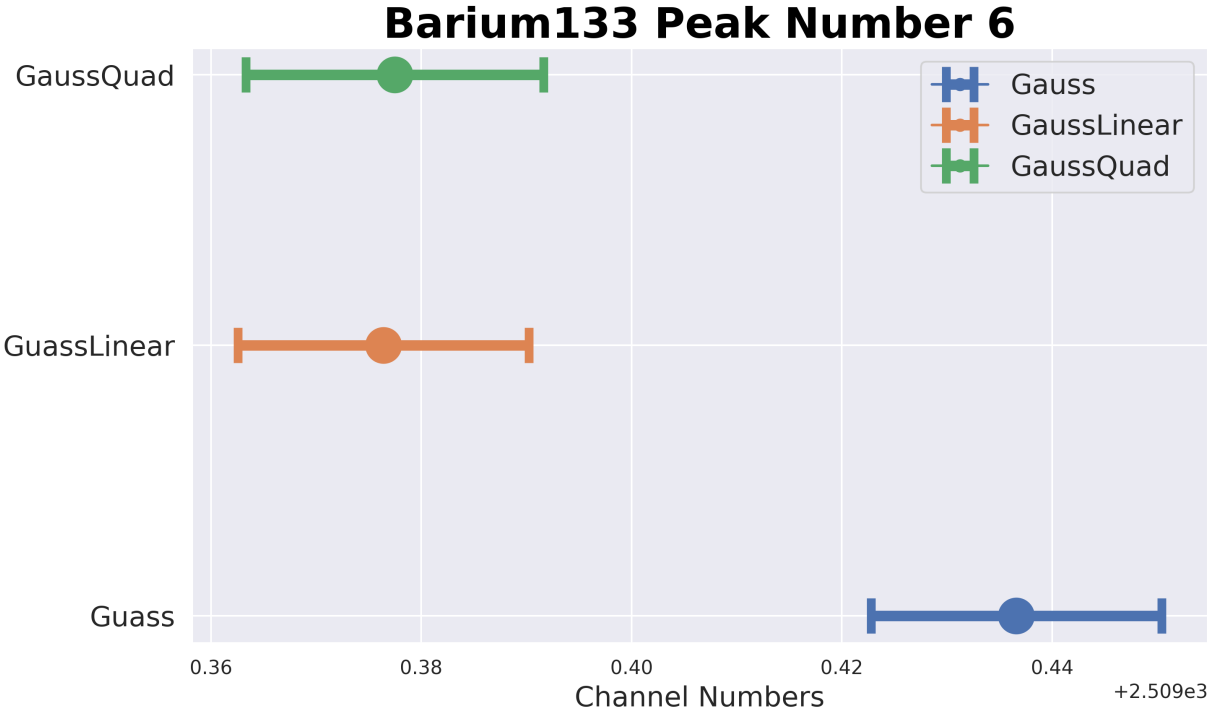


Figure A.39: Fit comparisons for ^{133}Ba 356 keV peak

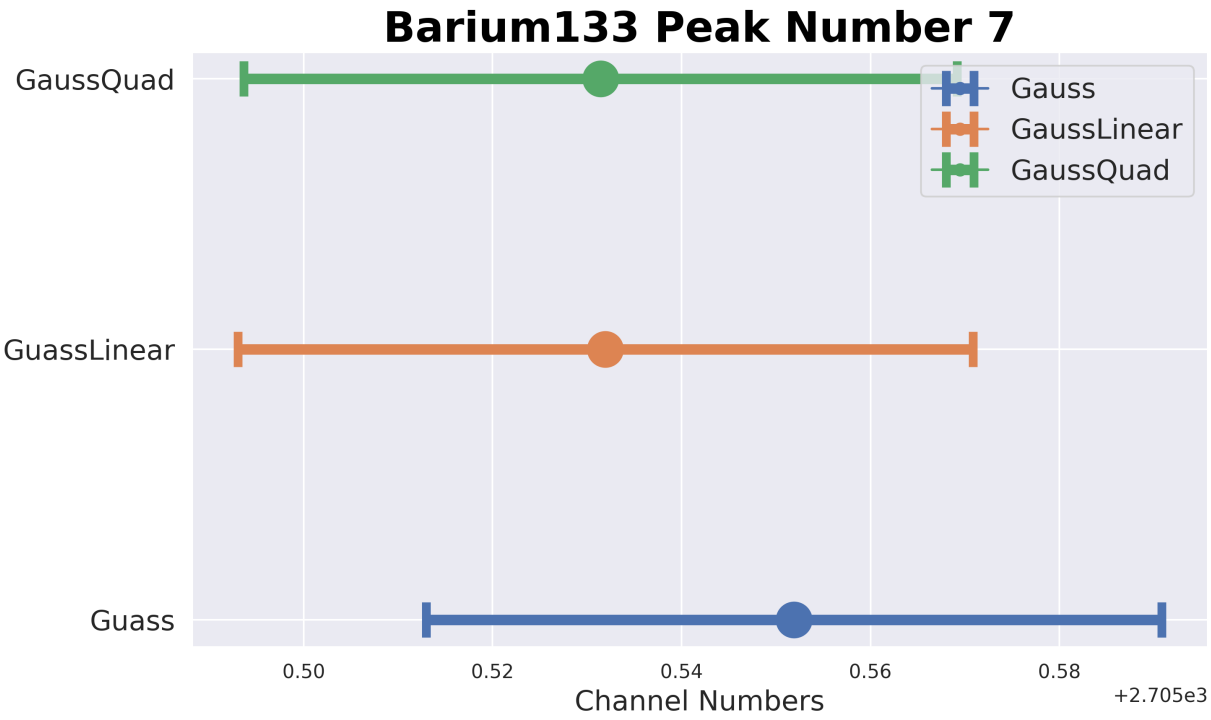


Figure A.40: Fit comparisons for ^{133}Ba 384 keV peak

A.2.2 Cobalt-60

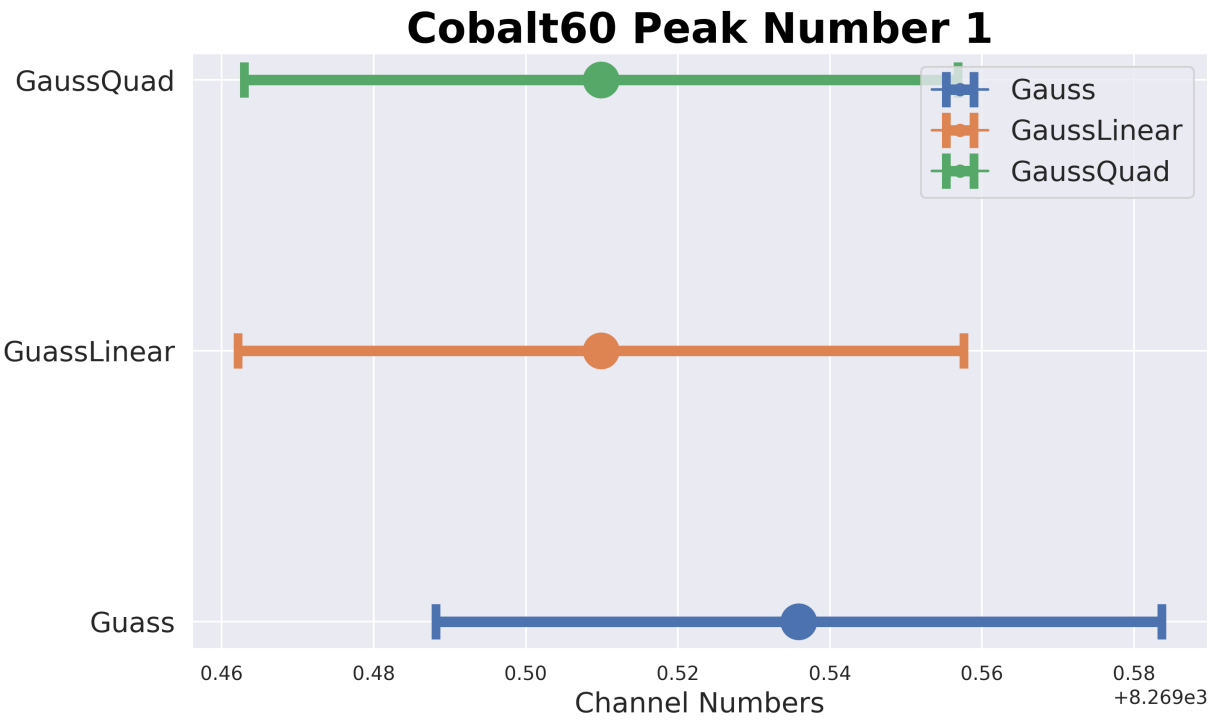


Figure A.41: Fit comparisons for ^{60}Co 1173 keV peak

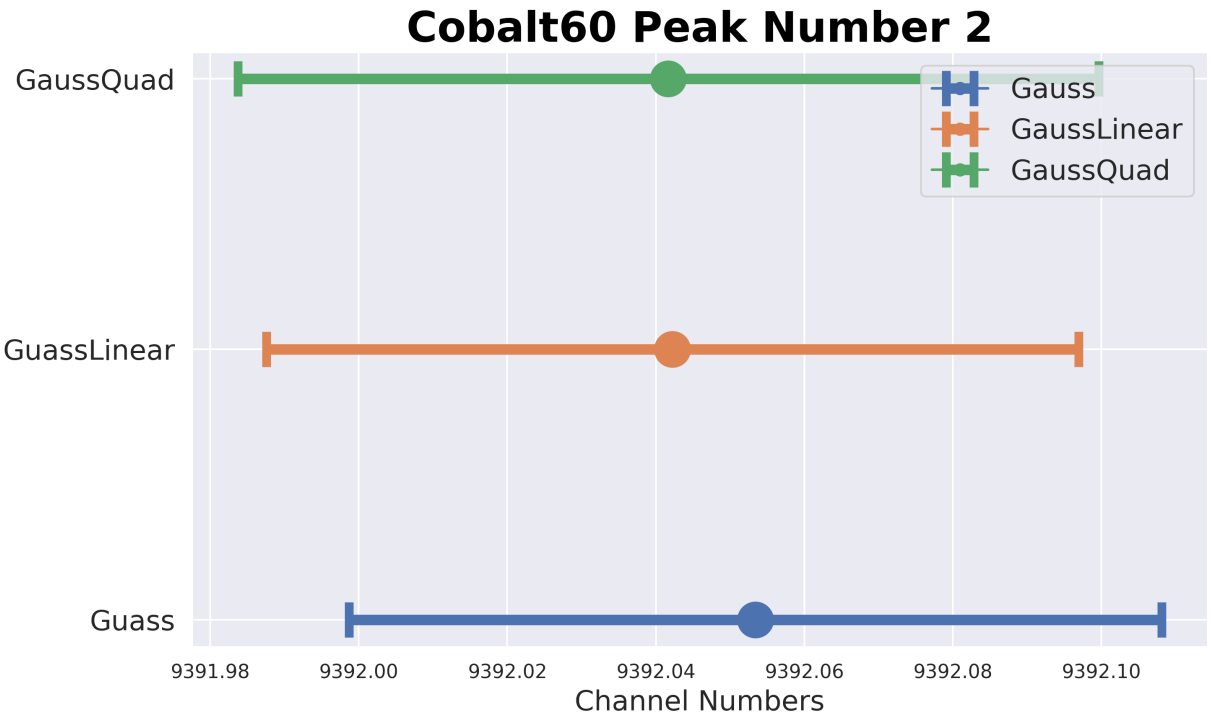


Figure A.42: Fit comparisons for ^{60}Co 1332 keV peak

A.2.3 Sodium-22

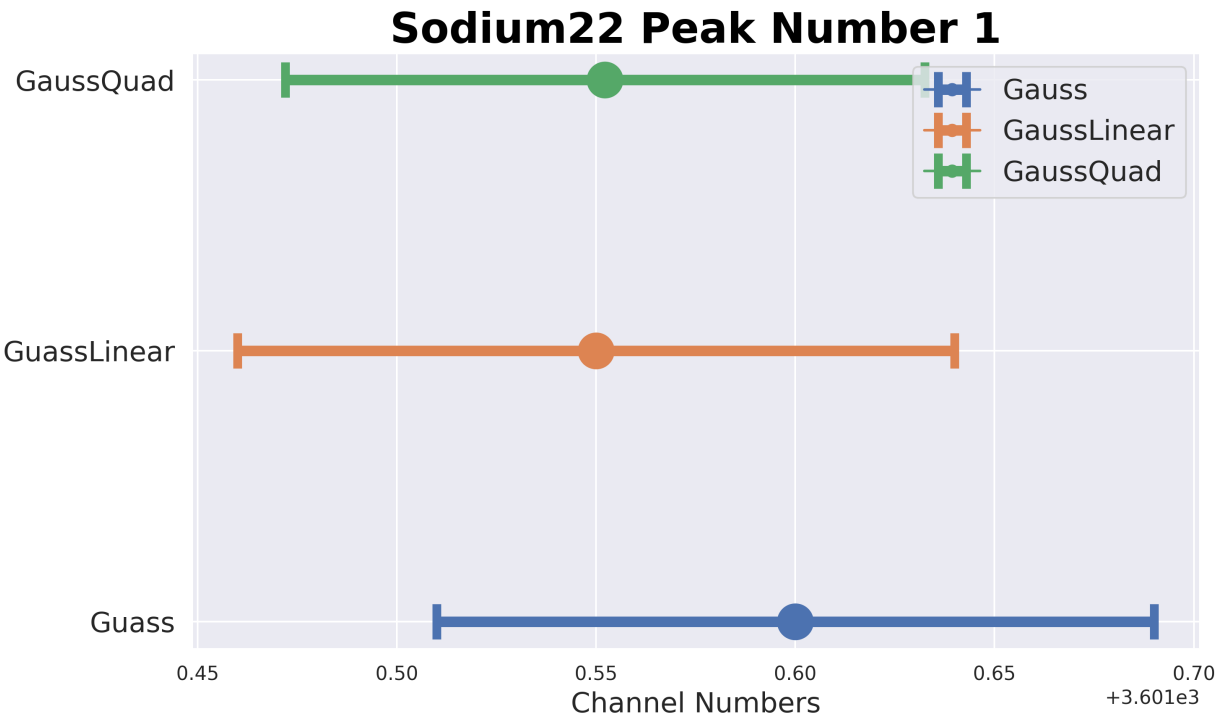


Figure A.43: Fit comparisons for ^{22}Na 511 keV peak

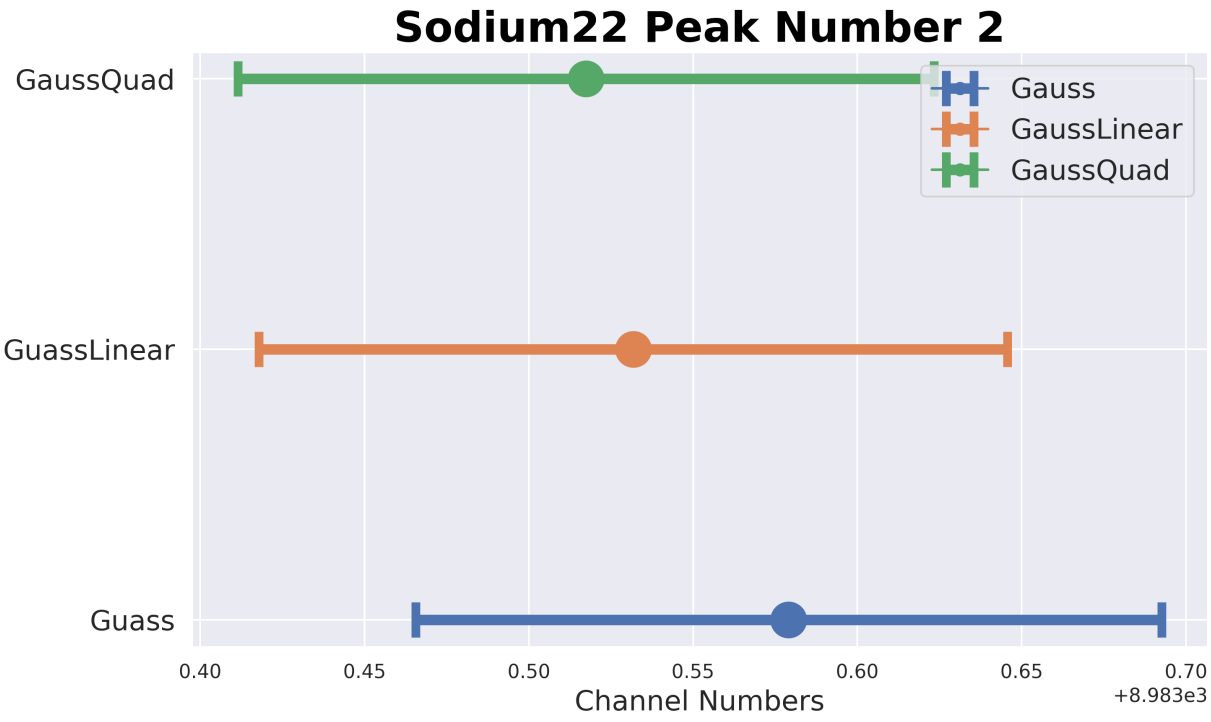


Figure A.44: Fit comparisons for ^{22}Na 1274 keV peak

A.3 Python Code

Below is the complete collection of our Python codes that we wrote during the four weeks of our experiment to run various data analysis tools on the raw data, including curve fitting, energy calibration and detector resolution finding. They are included in this appendix so that way we have them on records, in case something tragic happens to the digital copies stored on the central university file system.

However, we have also mitigated against file loss by uploading our Python files, raw data, and this appendix to a GitHub repository that is hosted externally. In addition to being used as a version tracker tool helping us identify changes made to the codes over time (as we add, change & remove features), it also provides mitigation against file loss as the current builds can simply be downloaded from the repo again. This also provides an easy way for others to double check figures or results, if they so desire – improving transparency for our experiment, which is something that all scientists should be striving to do.

GitHub: `https://github.com/AlexMaraio/HPGeDetector`

A.3.1 GaussianFit.py

This file is the file responsible for the fitting of the individual Gaussians to the peaks in the data. Here, we also add various order polynomials in addition to the Gaussian, starting from a simple offset to a quadratic order polynomial. This adjusts the way the fit works in subtle ways and we needed to see how fitting the various functions worked. Of course any arbitrary function can be approximated as a linear polynomial of arbitrary degree, so naturally going to a higher order polynomial will increase the goodness of fit, however detract from what can be extracted from the fit.

By looking how the mean changes between the different fit types we can determine if we are justified in increasing the order of the polynomial, and after our analysis thanks to the following code, we can not justify going beyond a linear polynomial in addition to the Gaussian function.

```
1  import numpy as np
2  import matplotlib.pylab as plt
3  import seaborn as sns
4  sns.set()
5  import scipy.optimize as sciopt
6  import scipy.stats as scistats
7  from lmfit import Model, Parameters
8  import pandas as pd
9
10 def ChiSqFunc(Measured,Fitted,Errors):
11   ChiSquared = 0
12   for i in range(len(Measured)):
13     ChiSquared += ((Measured[i] - Fitted[i])**2.0) / ((Errors[i])**2.0)
14   ReducedChiSq = ChiSquared/(len(Measured)-2)
15   ProbChiSq = (1.0 - scistats.chi2.cdf(ChiSquared,len(Measured)-2) )  * 100.0
16   return ChiSquared, ReducedChiSq, ProbChiSq
17
18 def Gauss(x,A,mean,sigma,a):
19   return (A/(np.sqrt(2*np.pi) * sigma )) *np.exp(-((x-mean)**2.0)/(2.0*(sigma**2.0)))
     ↪  + a
20
21 def GaussLinear(x,A,mean,sigma,a,b):
```

```
22    return (A/(np.sqrt(2*np.pi) * sigma ))*np.exp(-((x-mean)**2.0)/(2.0*(sigma**2.0)))
      ↪  + a + b*x
23
24  def GaussQuad(x,A,mean,sigma,a,b,c):
25    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
26
27  def ChToEnergy(Ch,a,b,c):
28    Energy = np.sqrt( ( (Ch - c + (b**2.0 / (4*a))) )/a ) - (b / (2*a))
29    return Energy
30
31  SetSigma = 2
32
33  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]] , []
    ↪  ]
34  SodiumList = [ [ [3570,3630,3600,511] , [8966,9000,8984,1274]] , [] ]
35  CobaltList = [ [ [8250,8290,8270,1173] , [9370,9415,9392,1332]] , [] ]
36
37  SourceList = ["Barium133-24HrRun_006_eh_1","Sodium22-24HrData_002_eh_1","Cobalt60-24H
    ↪  rData_004_eh_1"]
38
39  PlotResolution = 300
40
41  PeakNo = int(1)
42
43  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':[]}
44
45  # Fit parameters
46  a = 2.63977565e-06
47  b = 7.04767648
48  c = -1.91414134e-01
49
50  for source in SourceList:
51    if source == "Barium133-24HrRun_006_eh_1":
52      ElementList = BariumList
53      Element = "Barium133"
54    elif source == "Sodium22-24HrData_002_eh_1":
55      ElementList = SodiumList
56      Element = "Sodium22"
57    else:
58      ElementList = CobaltList
59      Element = "Cobalt60"
60
61    for item in ElementList:
62      for peak in item:
63        #! Initialises the source and ranges for run
64
65        if item == ElementList[0]:
66          PeakType = "Full"
67        else:
68          PeakType = "Zoom"
69
```

```
70          df = pd.read_csv(source + ".dat", sep = r"\s+", names = ['channel
            ↪  number','count number'])
71
72          df['count errors'] = np.sqrt(df['count number'])
73          df['count errors'] = df['count errors'].replace(0,1)
74
75          MinValue = peak[0]
76          MaxValue = peak[1]
77          MeanValue = peak[2]
78          PeakEnergy = peak[3]
79
80          data = df[(MinValue<=df['channel number']) & (df['channel number']<=MaxValue)]
81          #-------------------------------------------------------------------------
82          #! Gaussian + Offset model only
83
84          GaussModel = Model(Gauss)
85          Params = Parameters()
86
87          Params.add('A',value=max(data['count number']),vary=True,min=0)
88          Params.add('mean',value=MeanValue,vary=True)
89          Params.add('sigma',value=1,vary=True)
90          Params.add('a',value=1,vary=True)
91
92          FitResult = GaussModel.fit(data['count number'],params=Params,x=data['channel
            ↪  number'])
93          TempList = [ FitResult.best_values['mean'] - SetSigma
            ↪  *FitResult.best_values['sigma'] , FitResult.best_values['mean'] + SetSigma
            ↪  *FitResult.best_values['sigma'] , FitResult.best_values['mean'] ,
            ↪  PeakEnergy ]
94          if item == ElementList[0]:
95            ElementList[1].append(TempList)
96          #print(FitResult.best_values)
97          FitResult.plot(yerr=data['count errors'],xlabel='Channel Number',ylabel='Count
            ↪  Number',title='Gaussian + Constant Offset')
98
99          thingy = ChiSqFunc(list(data['count
            ↪  number']),list(FitResult.best_fit),list(data['count errors']))
100
101         plt.tight_layout()
102         plt.savefig(f'Plots/{Element}/Gauss/Gauss_{PeakNo}_{PeakType}.png',
            ↪  format='png', dpi=PlotResolution)
103         plt.close('all')
104
105         CountMax1 = FitResult.best_values['A'] / ( FitResult.best_values['sigma'] *
            ↪  np.sqrt(2 * np.pi ) ) + FitResult.best_values['a']
106         ErrorOnMean1 = FitResult.best_values['sigma'] / np.sqrt(CountMax1)
107
108         datadict['Element'].append(Element)
109         datadict['Fit type'].append('Gauss')
110         datadict['Energy (keV)'].append(PeakEnergy)
111         FWHM_Energy = 2 * np.sqrt(2*np.log(2)) *
            ↪  ChToEnergy(FitResult.best_values['sigma'],a,b,c)
112         datadict['Resolution'].append( FWHM_Energy / PeakEnergy)
113         #datadict['Resolution'].append( ( (2 *
            ↪  np.sqrt(2*np.log(2))*FitResult.best_values['sigma'] - b ) / a ) /
            ↪  PeakEnergy)
114         datadict['Peak number'].append(PeakNo)
```

```
115         datadict['Peak type'].append(PeakType)
116         datadict['Min of range'].append(MinValue)
117         datadict['Max of range'].append(MaxValue)
118         datadict['Mean'].append(FitResult.best_values['mean'])
119         datadict['A'].append(FitResult.best_values['A'])
120         datadict['Sigma'].append(FitResult.best_values['sigma'])
121         datadict['Error on mean'].append(1)
122         datadict['a'].append(FitResult.best_values['a'])
123         datadict['b'].append(0)
124         datadict['c'].append(0)
125         datadict['chisq'].append(thingy[0])
126         datadict['Reduced chisq'].append(thingy[1])
127         datadict['Probchisq'].append(thingy[2])
128
129
130         #-------------------------------------------------------------------------
131         #! Gaussain + Linear model
132
133         GaussModel2 = Model(GaussLinear)
134         Params2 = Parameters()
135
136         Params2.add('A',value=max(data['count number']),vary=True,min=0)
137         Params2.add('mean',value=MeanValue,vary=True)
138         Params2.add('sigma',value=1,vary=True)
139         Params2.add('a',value=1,vary=True)
140         Params2.add('b',value=1,vary=True)
141
142         FitResult2 = GaussModel2.fit(data['count
            ↪  number'],params=Params2,x=data['channel number'])
143         FitResult2.plot(yerr=data['count errors'],xlabel='Channel Number',ylabel='Count
            ↪  Number',title='Gaussian + Linear Polynomial')
144
145         thingy2 = ChiSqFunc(list(data['count
            ↪  number']),list(FitResult2.best_fit),list(data['count errors']))
146
147         plt.tight_layout()
148         plt.savefig(f'Plots/{Element}/Linear/Linear_{PeakNo}_{PeakType}.png',
            ↪  format='png', dpi=PlotResolution)
149         plt.close('all')
150
151         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']
152         ErrorOnMean2 = FitResult2.best_values['sigma'] / np.sqrt(CountMax2)
153
154         datadict['Element'].append(Element)
155         datadict['Fit type'].append('Linear')
156         datadict['Energy (keV)'].append(PeakEnergy)
157         FWHM_Energy = 2 * np.sqrt(2*np.log(2)) *
            ↪  ChToEnergy(FitResult2.best_values['sigma'],a,b,c)
158         datadict['Resolution'].append( FWHM_Energy / PeakEnergy)
159         datadict['Peak number'].append(PeakNo)
160         datadict['Peak type'].append(PeakType)
161         datadict['Min of range'].append(MinValue)
162         datadict['Max of range'].append(MaxValue)
163         datadict['Mean'].append(FitResult2.best_values['mean'])
164         datadict['A'].append(FitResult2.best_values['A'])
```

```
        datadict['Sigma'].append(FitResult2.best_values['sigma'])
        datadict['Error on mean'].append(1)
        datadict['a'].append(FitResult2.best_values['a'])
        datadict['b'].append(FitResult2.best_values['b'])
        datadict['c'].append(0)
        datadict['chisq'].append(thingy2[0])
        datadict['Reduced chisq'].append(thingy2[1])
        datadict['Probchisq'].append(thingy2[2])

        #-------------------------------------------------------------------------
        #! Gaussain + Quadratic model

        GaussModel3 = Model(GaussQuad)
        Params3 = Parameters()

        Params3.add('A',value=max(data['count number']),vary=True,min=0)
        Params3.add('mean',value=MeanValue,vary=True)
        Params3.add('sigma',value=1,vary=True)
        Params3.add('a',value=1,vary=True)
        Params3.add('b',value=1,vary=True)
        Params3.add('c',value=1,vary=True)

        FitResult3 = GaussModel3.fit(data['count
        ↪  number'],params=Params3,x=data['channel number'])
        FitResult3.plot(yerr=data['count errors'],xlabel='Channel Number',ylabel='Count
        ↪  Number',title='Gaussian + Quadratic Polynomial')

        thingy3 = ChiSqFunc(list(data['count
        ↪  number']),list(FitResult3.best_fit),list(data['count errors']))

        plt.tight_layout()
        plt.savefig(f'Plots/{Element}/Quad/Quad_{PeakNo}_{PeakType}.png', format='png',
        ↪  dpi=PlotResolution)
        plt.close('all')

        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
        ErrorOnMean3 = FitResult3.best_values['sigma'] / np.sqrt(CountMax3)

        datadict['Element'].append(Element)
        datadict['Fit type'].append('Quad')
        datadict['Energy (keV)'].append(PeakEnergy)
        FWHM_Energy = 2 * np.sqrt(2*np.log(2)) *
        ↪  ChToEnergy(FitResult3.best_values['sigma'],a,b,c)
        datadict['Resolution'].append( FWHM_Energy / PeakEnergy)
        datadict['Peak number'].append(PeakNo)
        datadict['Peak type'].append(PeakType)
        datadict['Min of range'].append(MinValue)
        datadict['Max of range'].append(MaxValue)
        datadict['Mean'].append(FitResult3.best_values['mean'])
        datadict['A'].append(FitResult3.best_values['A'])
        datadict['Sigma'].append(FitResult3.best_values['sigma'])
        datadict['Error on mean'].append(1)
        datadict['a'].append(FitResult3.best_values['a'])
        datadict['b'].append(FitResult3.best_values['b'])
```

```
        datadict['c'].append(FitResult3.best_values['c'])
        datadict['chisq'].append(thingy3[0])
        datadict['Reduced chisq'].append(thingy3[1])
        datadict['Probchisq'].append(thingy3[2])


        Delta12 = np.abs(FitResult.best_values['mean'] - FitResult2.best_values['mean'])
        ErrorDelta12 = np.sqrt( ErrorOnMean1**2.0 + ErrorOnMean2**2.0 )

        Delta23 = np.abs(FitResult2.best_values['mean'] -
        ↪  FitResult3.best_values['mean'])
        ErrorDelta23 = np.sqrt( ErrorOnMean2**2.0 + ErrorOnMean3**2.0 )


        if PeakType == "Zoom":
          if abs(ErrorDelta23) > abs(Delta23):
            print('best fit')
          else:
            print('not best fit')

          TitleFont = {'size':'24', 'color':'black', 'weight':'bold'}
          AxTitleFont = {'size':'16'}
          plt.figure(figsize=(10,6))
          plt.errorbar(FitResult.best_values['mean'],1,xerr=ErrorOnMean1,elinewidth=6,c
          ↪  apsize=10,capthick=5,marker='o',markersize=20,label="Gauss")
          plt.errorbar(FitResult2.best_values['mean'],1.25,xerr=ErrorOnMean2,elinewidth
          ↪  =6,capsize=10,capthick=5,marker='o',markersize=20,label="GaussLinear")
          plt.errorbar(FitResult3.best_values['mean'],1.5,xerr=ErrorOnMean3,elinewidth=
          ↪  6,capsize=10,capthick=5,marker='o',markersize=20,label="GaussQuad")
          plt.legend(fontsize=16,markerscale=0.33)
          plt.title(str(Element) + ' Peak Number ' + str(PeakNo),**TitleFont)
          plt.xlabel('Channel Numbers',**AxTitleFont)
          plt.yticks([1,1.25,1.5],['Guass','GuassLinear','GaussQuad'],**AxTitleFont)
          plt.tight_layout()
          plt.savefig(f'Plots/{Element}/FitComparison_Peak{PeakNo}.png', format='png',
          ↪  dpi=PlotResolution)
          plt.close('all')

        PeakNo +=1

      PeakNo = 1

Fitdf = pd.DataFrame(datadict)
Fitdf.to_csv('Gamma_Peak_Stats_and_Params.csv')
```

A.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)
```

A.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()

```

```
47
48  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')
97  TitleFont = {'size':'25', 'color':'black', 'weight':'bold'}
98  AxTitleFont = {'size':'22'}
99  plt.xlabel('Gamma Energy [keV]',**AxTitleFont)
100 plt.ylabel('Log-10 of count number',**AxTitleFont)
101 plt.title('Barium 133, no background, energies',**TitleFont)
102 plt.xlim(0)
103 plt.ylim(1)
104 plt.legend()
105
106
107 plt.figure(3)
108 plt.semilogy(CobaltDF['Energies'],CobaltDF['count number'],label="Cobalt 60
    ↪  data",color='blue')
109 TitleFont = {'size':'25', 'color':'black', 'weight':'bold'}
110 AxTitleFont = {'size':'22'}
111 plt.xlabel('Gamma Energy [keV]',**AxTitleFont)
112 plt.ylabel('Log-10 of count number',**AxTitleFont)
113 plt.title('Cobalt 60, no background, energies',**TitleFont)
114 plt.xlim(0)
115 plt.ylim(1)
116 plt.legend()
117 plt.show()
```

A.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)
```