X-Ray Fluorescence

March 13, 2018

1 X-Ray Fluorescence

In this lab, we will use an Amptek Mini-X X-Ray Tube in conjunction with the X-123 Complete Spectrometer to use X-Ray fluorescence to determine the composition of an unidentified sample.

1.1 6 March

We were unable to do any work on the experiment last week, as we were trying to modify the fluorescence setup to do X-Ray diffraction. However, after consulting with Jun Ye, we decided to abandon the diffraction experiment in favor of performing the fluorescence experiment.

SETUP * We first had to reinstall the device drivers for the spectrometer. All driver files downloaded to desktop. * Once the computer is talking to the devices, we were able to make a run on a Pb sample. * We follow the procedures given by the Experimenter's Kit Quick Start June 2012.pdf

CALIBRATION

- We have several different labeled samples to calibrate our equipment with: Cu, Pb, Sn, Al and we assume Steel.
- According to the X-123 Specification sheet, the Energy Range at $>$ 25% is 1.5 keV to 25 keV, although the manual claims it will detect outside this range at lower energy (http://amptek.com/products/x-123-complete-x-ray-spectrometer-with-si-pin-detector/)
- Combining this range with the K and L emission lines (http://amptek.com/pdf/xraychrt.pdf), we can use our samples to calibrate the energy axis of the detector. We will do this for each of the samples to get an idea of the precision and accuracy of the spectrometer.
- We follow the following calibration procedure:
  - Place Sample in spectrometer
  - Clear all data from Amptek DppMCA (Gain set to 30), time duration set to 30 s
  - Set X ray tube to 30 kV, 100 μA
  - Start X ray
  - Clear Spectrometer Data
  - Start Spectrometer collection
  - Turn off xray when data collection complete.
  - Compare sample peaks to range of spectrometer to determine which peaks correspond to which transistions

- Calibrate Spectrometer
- Note: To use the calibration feature of the Amptek software, we first define Regoins of Interest (ROI) by providing the min/max channel values of the windows defining the peaks.
- We then go into calibration, click on the peak of interest, click 'centroid' on the RHS, and enter the actual value in keV of the peak location. Then clinck 'add'
- Repeat for all peaks in the spectrum
- Once complete, click 'enable calibration.'

We repeat this procedure 3 times for each of our samples.
* A = -0.0311604 keV (Offset) * B = 0.0176715 keV (keV/Channel)

1.2 7 March

We had difficulty with the Analysis software from Amptek. We will repeat the calibration, and use the DppMCA interface with the spectrometer for data collection.

- We set the Voltage and current of the X-Ray tube to 40 kV and 100 μA
- We collect the spectra of Pb, Cu, and Sn for calibration.
- We calibrate the horizontal scale based on the known energies of the various known transition lines.

Through calibration, we have a way of characterizing the error in our spectrometer, as well as converting an arbitrary channel into an energy. Once this is complete, we have a way of determining the energies of peaks in various samples and comparing them to the 'fingerprints' of the constituent atoms. This should allow us to deterimine the composition of various samples.

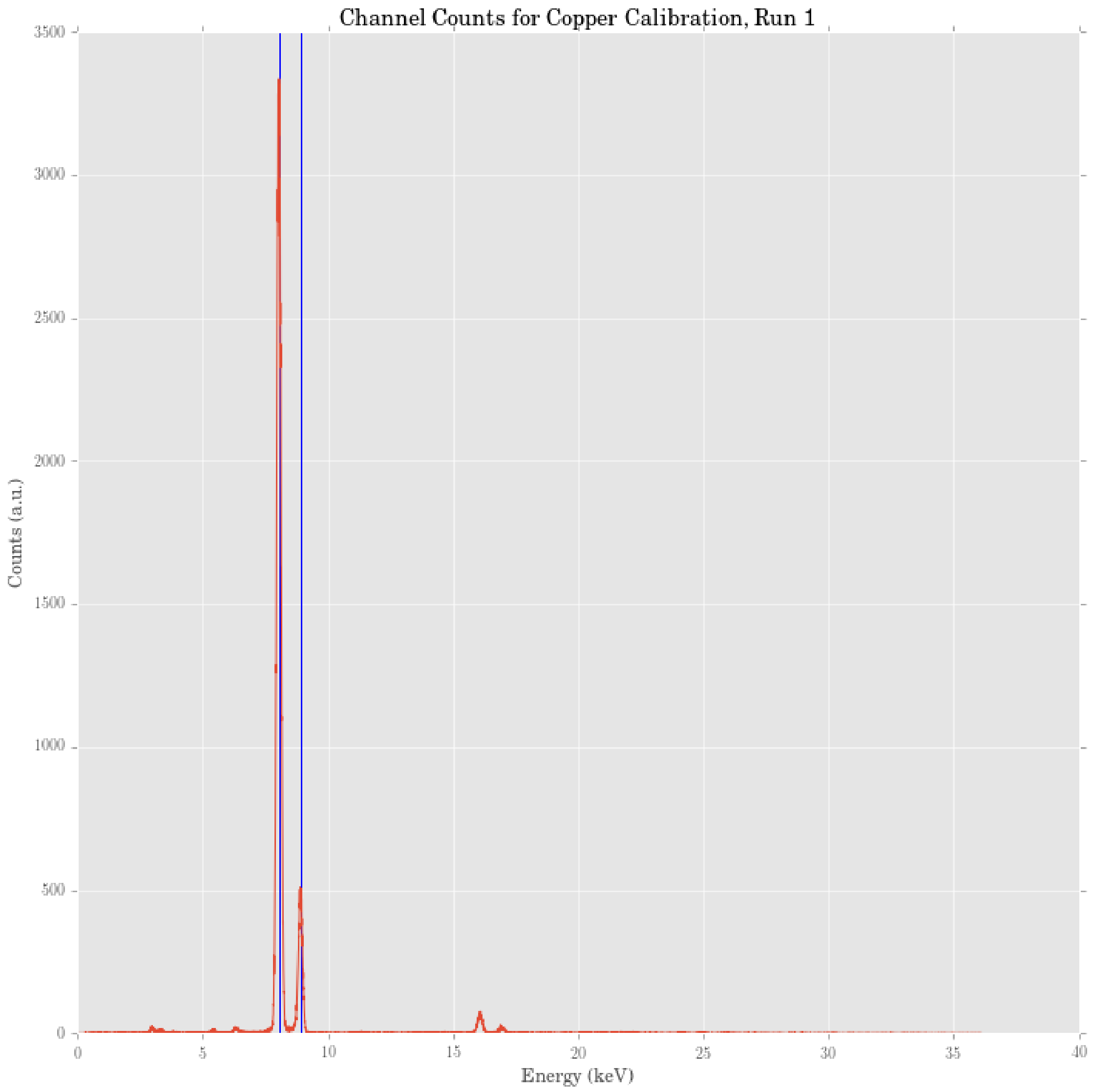
```
In [31]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         import os
         from scipy import signal
         from scipy import stats
         from scipy import optimize
         from ipywidgets import interact
         from scipy.stats import norm


         plt.style.use('ggplot')
         plt.rc('text', usetex=True)
         plt.rc('font', family='serif')
         plt.rcParams.update({'font.size': 18, 'text.usetex': True})


         %matplotlib inline

In [61]: class spectrograph:
             """
```

```
From a datafile containing the counts from an X-Ray Fluorescence Spectrograph, we r
the counts from a spectrometer for energies in a given channel.  See the 'MCAC' lin
for the specific number of channels used.  The ones we used here were for 8192 chan
"""

def __init__(self, file_path):
    self.file_path = file_path
    self.file = pd.read_csv(file_path, skiprows = 32, skipfooter = 71, names = ['Co
    self.cal_data = pd.read_csv(self.file_path, sep = ' ',skiprows = 13, skipfooter
                                names = ['Channel','Energy (keV)'], engine = 'pytho
    self.cal_fit_params, cal_fit_covar = np.polyfit(self.cal_data['Channel'], self.
                                                    1, cov=True)
    self.cal_err = np.sqrt(np.diag(cal_fit_covar))
    self.calm_report = self.reported_values(self.cal_fit_params[0], self.cal_err[0]
    self.calb_report = self.reported_values(self.cal_fit_params[1], self.cal_err[1]
    self.cal_energy = np.poly1d(self.cal_fit_params)(np.arange(len(self.file)))
    self.cal_chi2, self.cal_p = stats.chisquare(self.cal_data['Energy (keV)'],\
                                                np.poly1d(self.cal_fit_params)(self
    self.file.insert(0, 'Energy (keV)', self.cal_energy)

def print_data(self):
    return self.file

def get_counts(self):
    return self.file['Counts']

def reported_values(self, value, error):
    a = int(np.floor(np.log10(np.abs(error))))
    report_Err = round(error, -a)
    report_Val = round(value, -a)
    return report_Val, report_Err


def plot_calib(self):
    cal_fit_x = np.linspace(0,9000,9000)
    cal_fit_y = np.poly1d(self.cal_fit_params)(cal_fit_x)
    fig, ax = plt.subplots(figsize = (10,10))
    ax.scatter(self.cal_data['Channel'], self.cal_data['Energy (keV)'], color = 'b'
    ax.plot(cal_fit_x, cal_fit_y, \
            label = str('Best Fit:\n m = {}' + r'$\pm$' + '{} keV/Channel \n b = {}
                        r'$\pm$' + '{} keV\n p-val = {}')\
            .format(self.calm_report[0], self.calm_report[1],\
                    self.calb_report[0], self.calb_report[1],self.cal_p))
    ax.set_title('Channel to Energy Calibration')
    ax.set_xlabel('Channel Number')
    ax.set_ylabel('Energy (keV)')
    ax.axhline(y=0, color = 'k')
    ax.axvline(x=0, color = 'k')
```

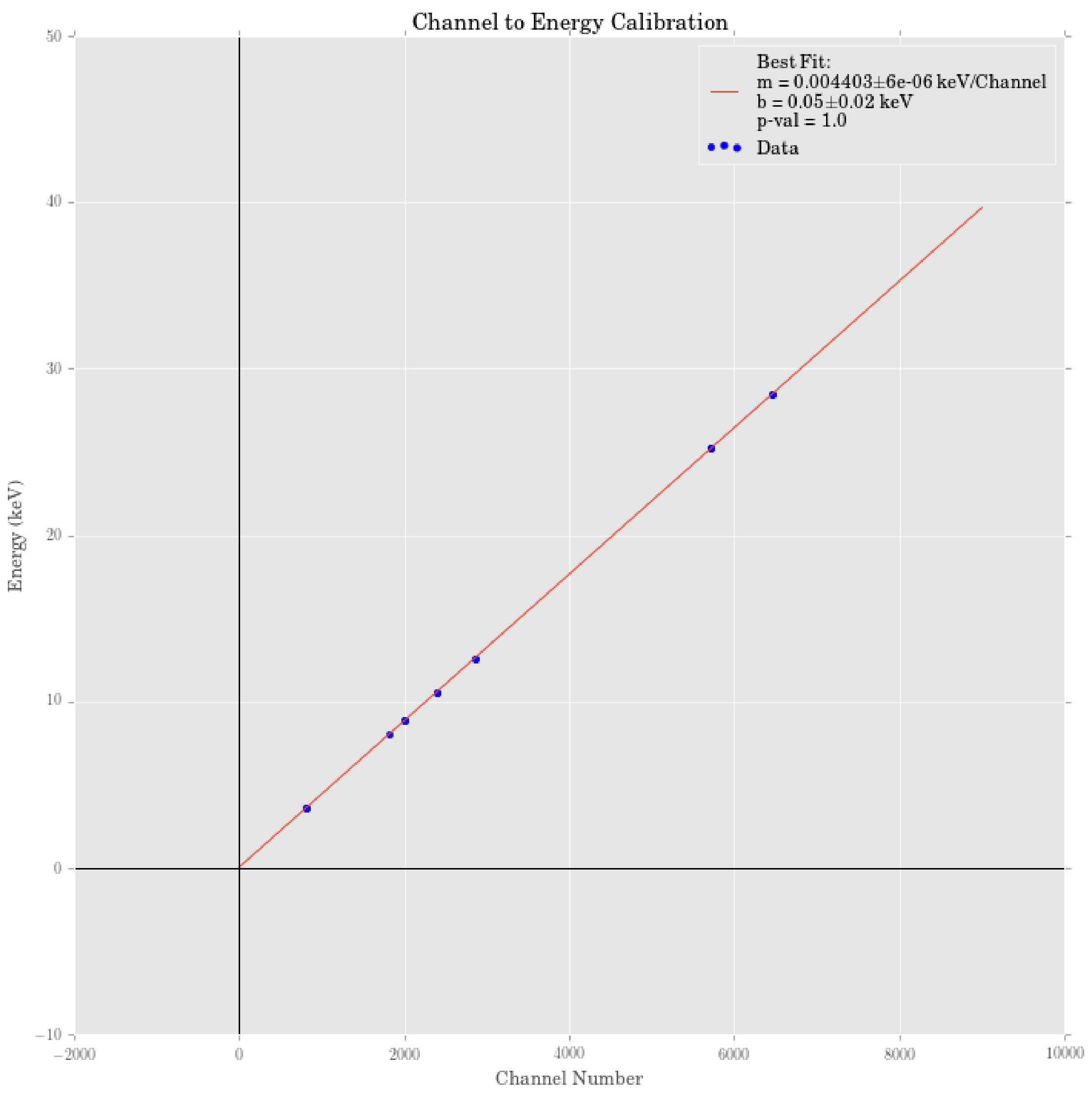
```
            ax.legend()
            fig.tight_layout()


        def plot_spectrum(self, dataName):
            self.fig, self.ax = plt.subplots(figsize = (10,10))
            self.ax.plot(self.file['Counts'])
            self.ax.plot(self.file['Energy (keV)'], self.file['Counts'])
            self.ax.set_title('Channel Counts for {}'.format(str(dataName)))
    #         self.ax.set_xlabel('Channel (a.u)')
            self.ax.set_xlabel('Energy (keV)')
            self.ax.set_ylabel('Counts (a.u.)')
            self.ax.axhline(y=0, color = 'k')
            self.ax.axvline(x=0, color = 'k')
            self.fig.tight_layout()

        def find_peaks(self, peak_width, snr):
            self.peaks = self.file['Energy (keV)'][signal.find_peaks_cwt(self.file['Counts'
                                                                           np.arange(1,peak_w

        def fit_peaks(self, peaklocs, peakhts, peakwd):
            peak_info = pd.DataFrame(np.zeros(len(peaklocs),6),\
                                 columns =['Peak Location','Peak Width','Peak Height',\
                                           'Peak Location Err','Peak Width Err', 'Peak Heig
            for i in np.arange(len(peaklocs)):
                optimize.curve_fit(gauss_fit, )




    def gauss_fit(x, x0, height, width):
        return height*width*np.sqrt(2*np.pi)*norm.pdf(x, loc = x0, scale = width)


    def make_peak(x0, ):
        x = np.linspace(0, 9000,9000)
        y = height**np.sqrt(2*np.pi)*norm.pdf(x, loc = x0, scale = )
        fig, ax = plt.subplots(figsize = (8,8))
        ax.plot(x,y)
        fig.tight_layout()
```

```
In [33]: a = spectrograph('SumCrossXRF/Calibration/7Mar_Cu1.txt')
         a.find_peaks(100, 20)
         a.plot_spectrum('Copper Calibration, Run 1')
         a.ax.vlines(a.peaks, 0, 3500, color = 'b')
         # a.peaks
         # np.poly1d(a.cal_fit_params)(100)
```

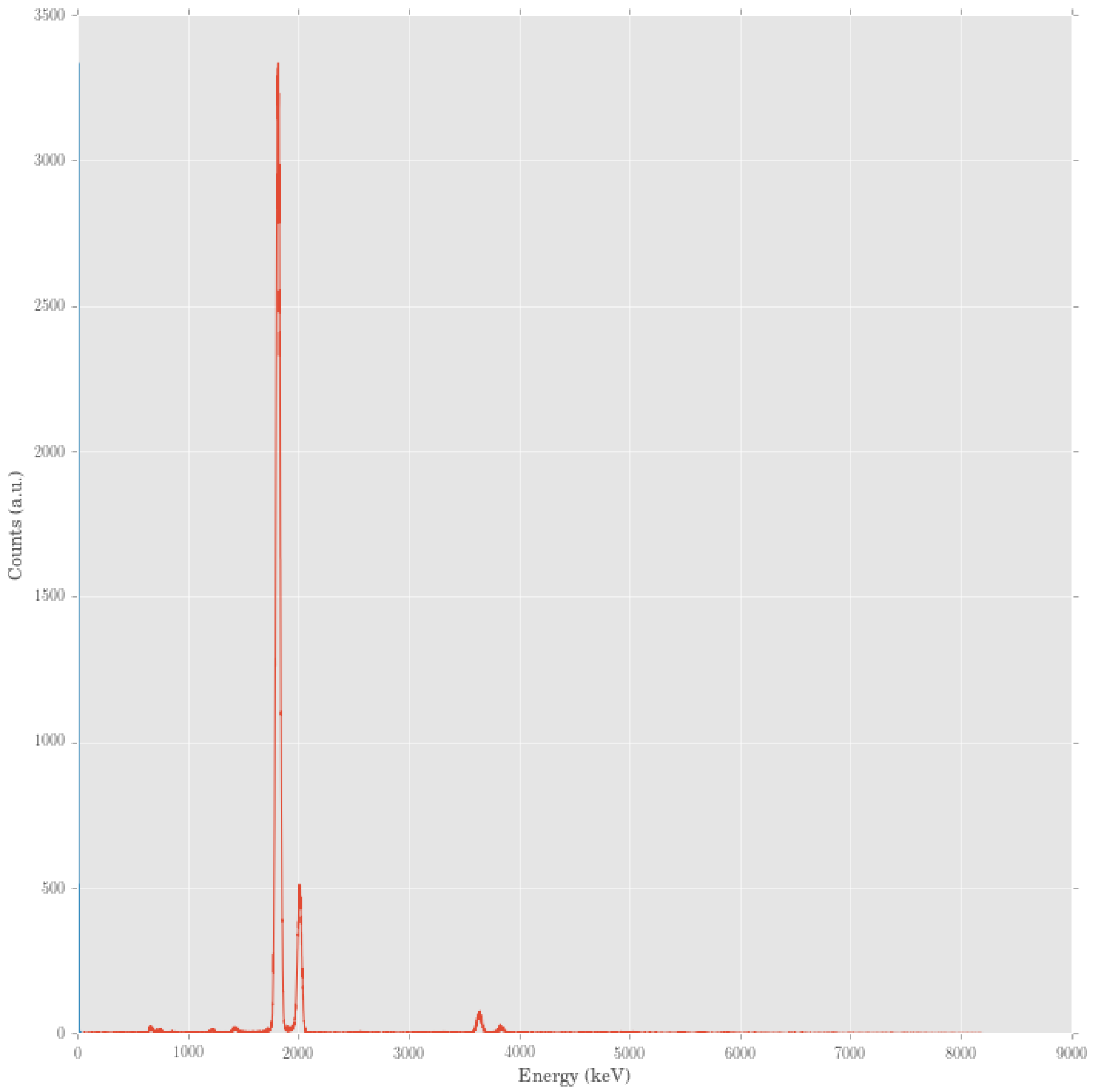
```
Out[33]: <matplotlib.collections.LineCollection at 0x7f16d67952b0>
```



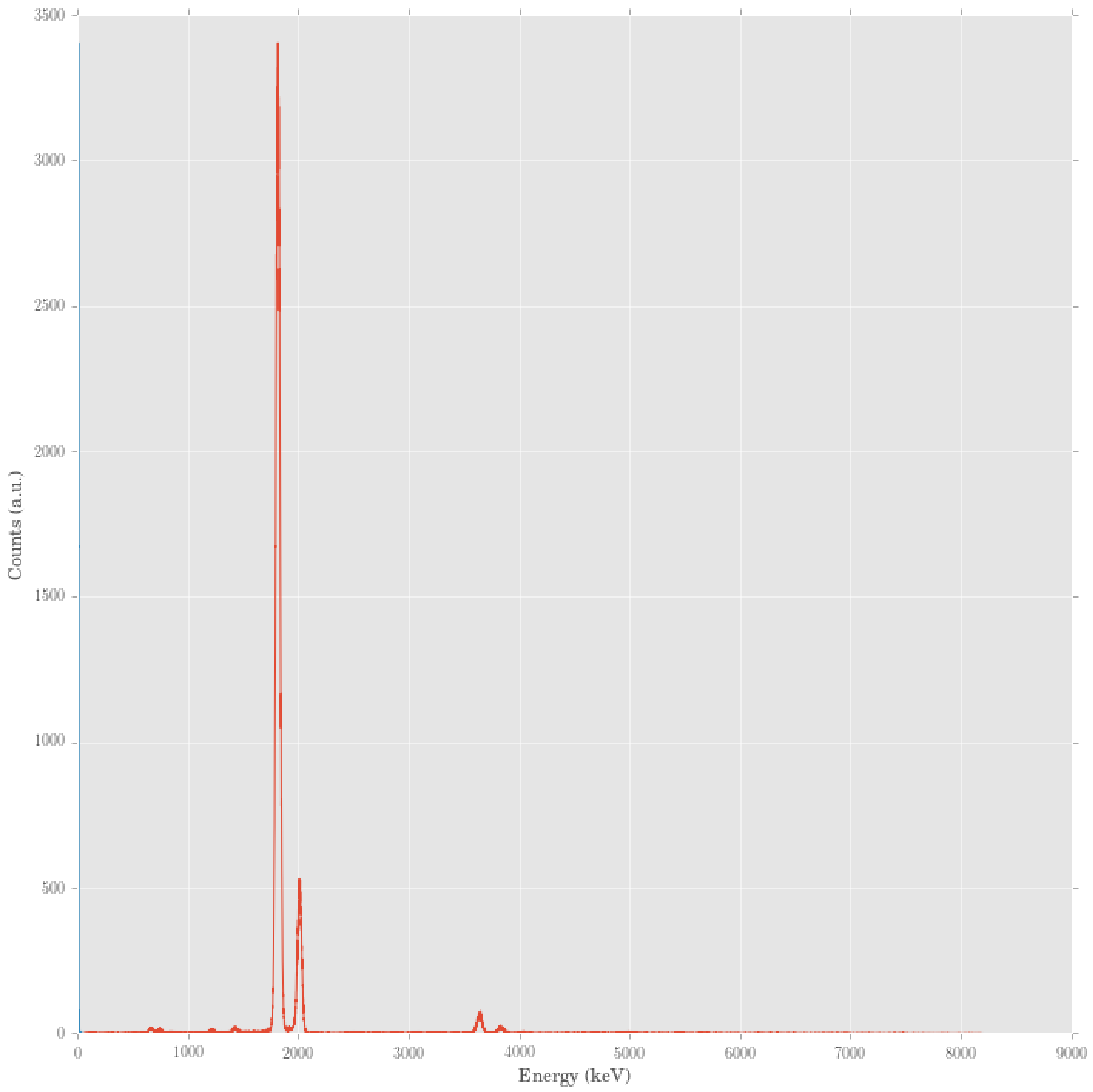
```
In [46]: a.plot_calib()
```

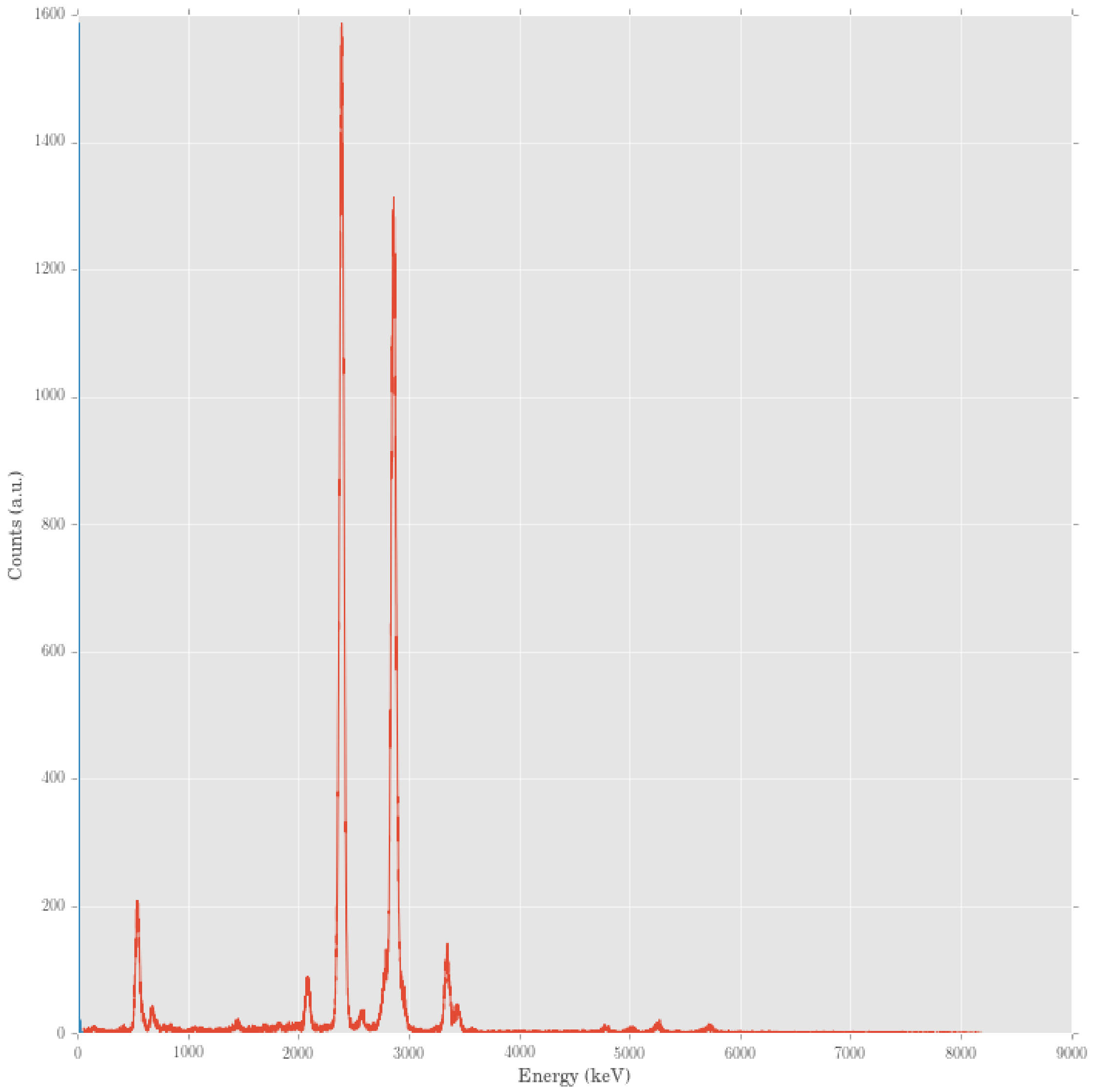


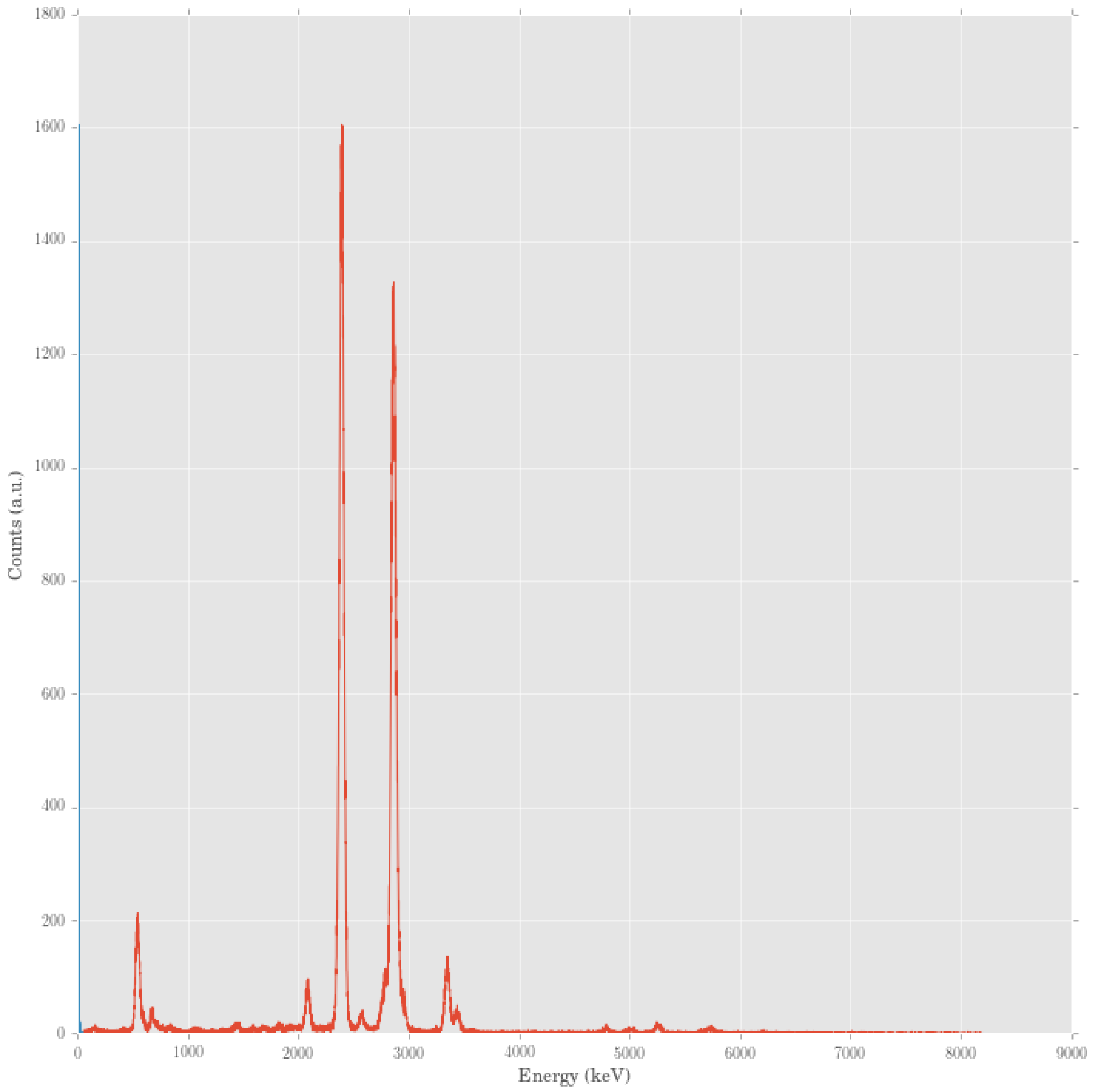
```
In [64]: test = spectrograph('SumCrossXRF/Calibration/7Mar_Cu1.txt')
         # test.cal_good_fit()
         # print(test.calm_report[1])
         # test.plot_calib()
         # test.file
         # len(test.cal_energy)
         # test.file['Energy (keV)']
         # np.array(test.file['Counts'])

In [57]: # Plot all calibration Runs
         for file in sorted(os.listdir('SumCrossXRF/Calibration/')):
             spect = spectrograph('SumCrossXRF/Calibration/{}'.format(file))
             spect.plot_spectrum(file[5:7])
```

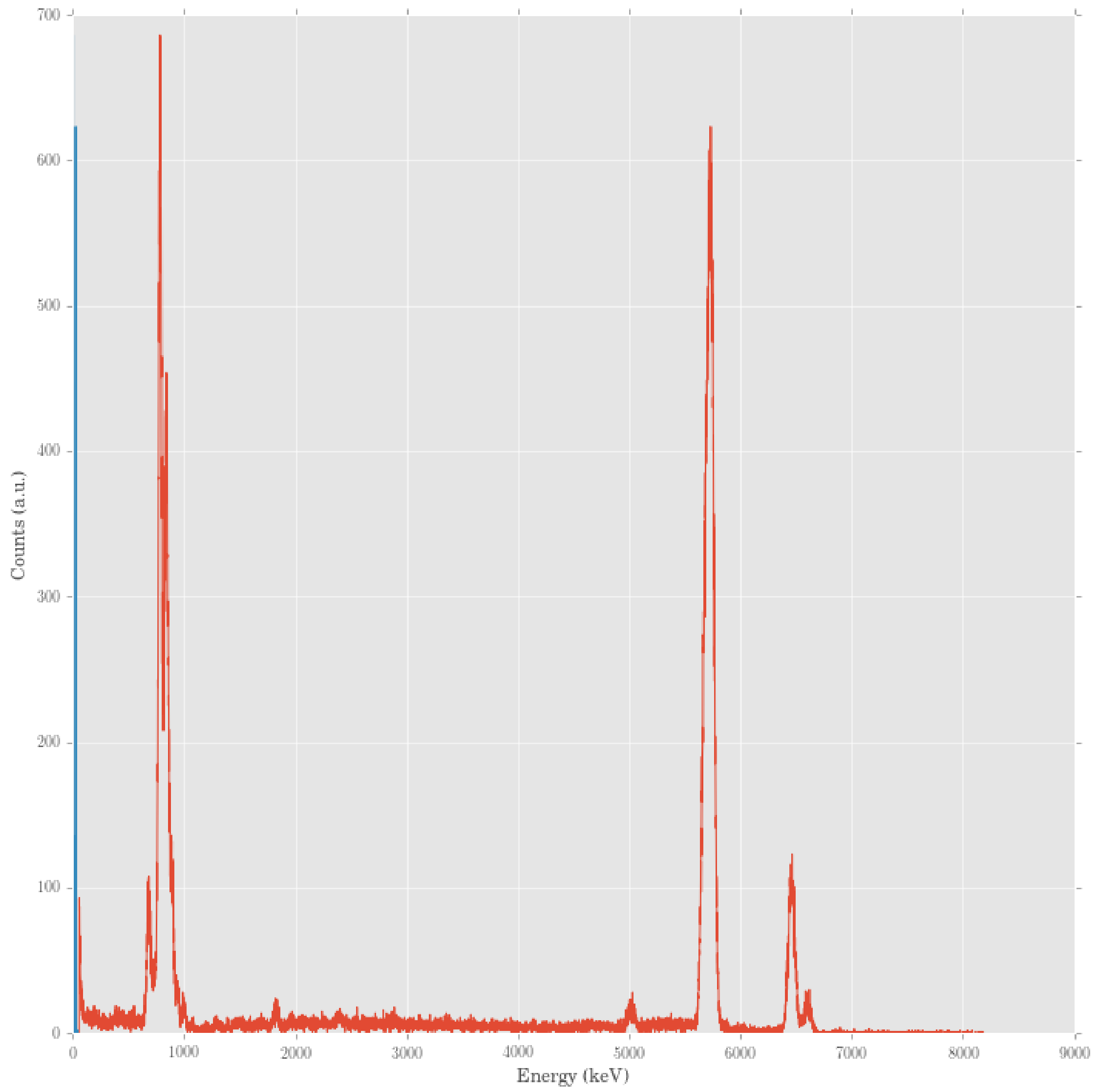


Counts (a.u.)
Energy (keV)
0
500
1000
1500
2000
2500
3000
3500
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

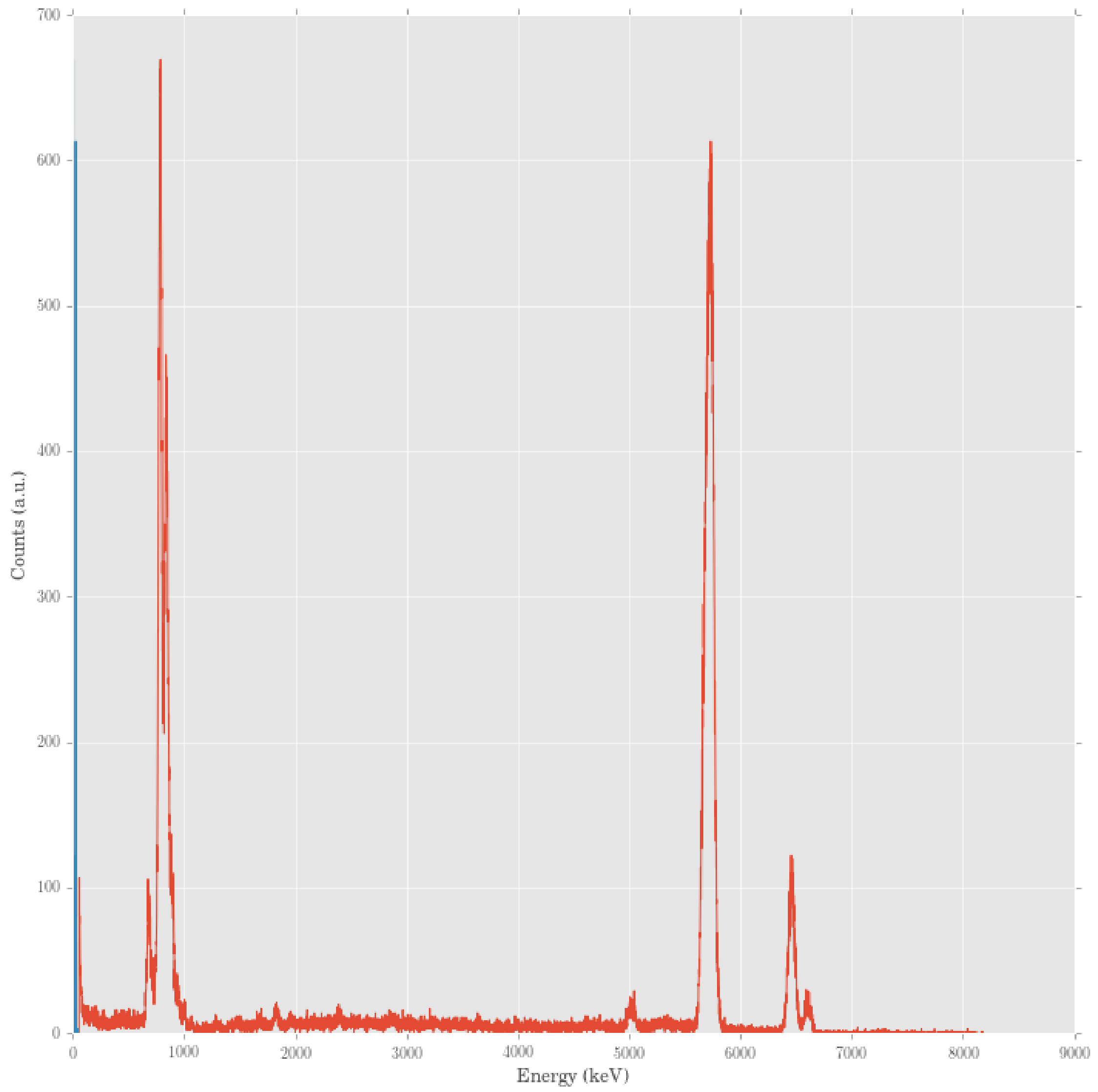

Counts (a.u.)
Energy (keV)
0
500
1000
1500
2000
2500
3000
3500
0
1000
2000
3000
4000
5000
6000
7000
8000
9000


Counts (a.u.)
Energy (keV)
0
200
400
600
800
1000
1200
1400
1600
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

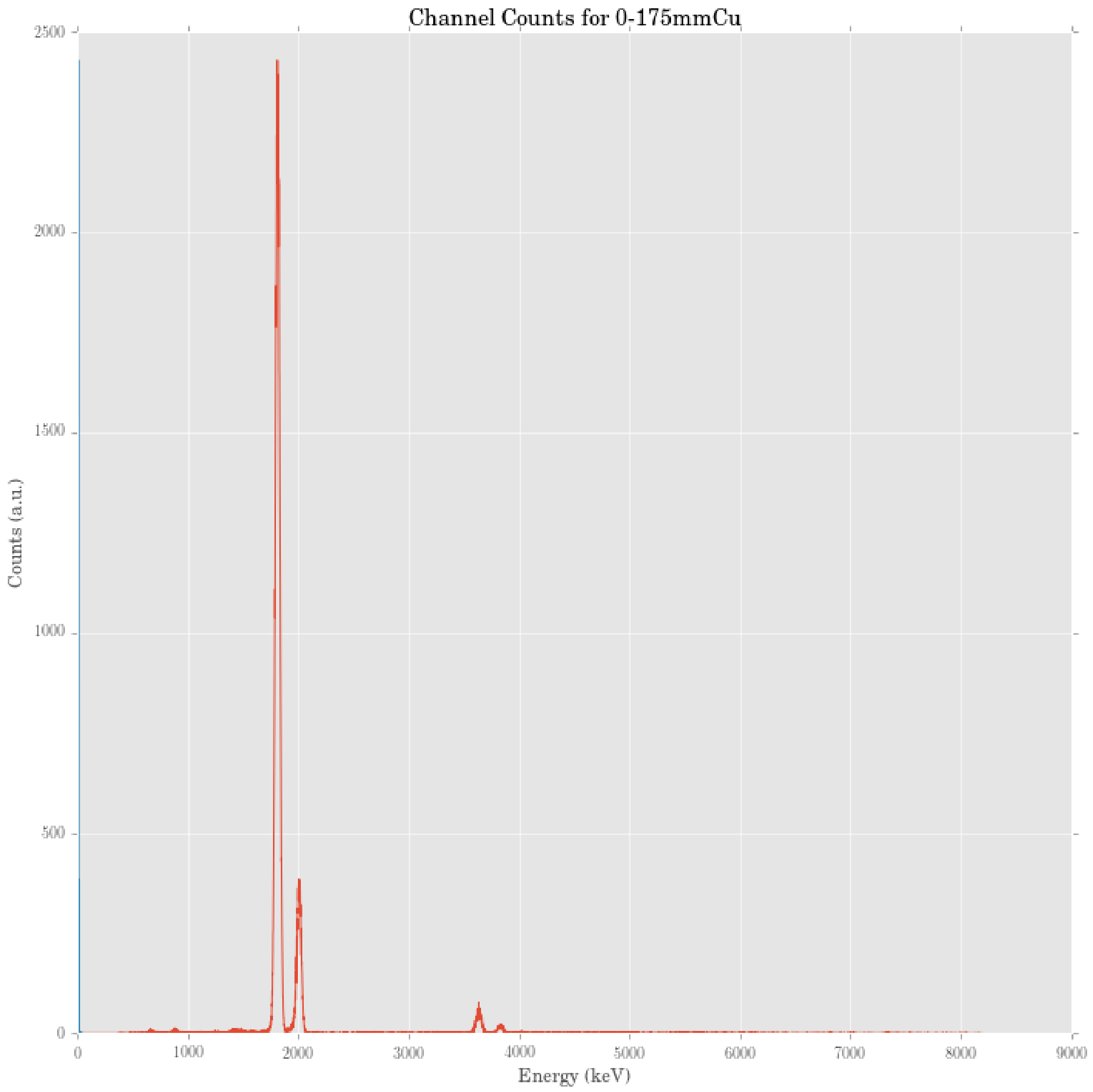

Counts (a.u.)
Energy (keV)
0
200
400
600
800
1000
1200
1400
1600
1800
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

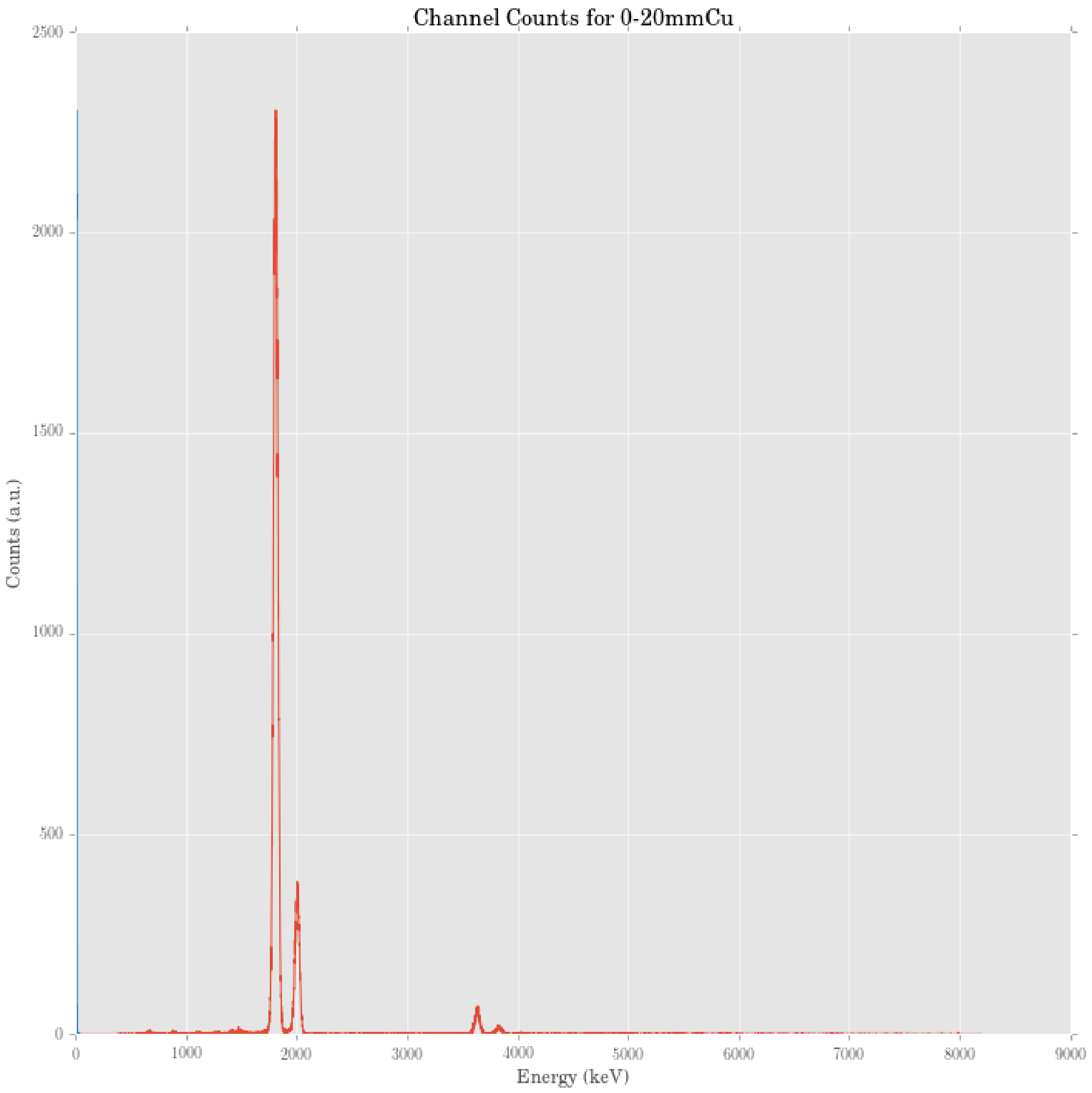


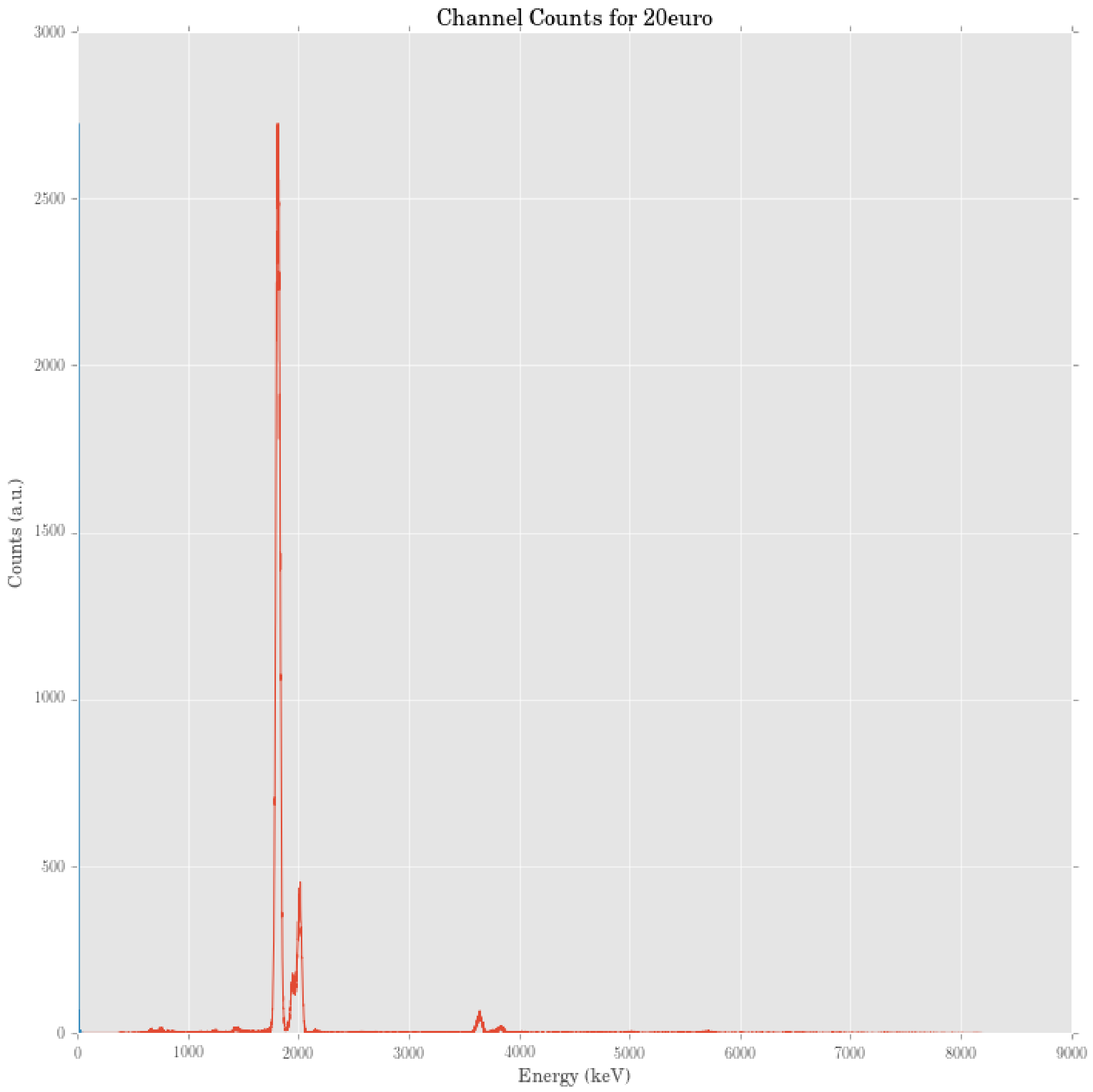
Counts (a.u.)
Energy (keV)
0
100
200
300
400
500
600
700
1000
2000
3000
4000
5000
6000
7000
8000
9000

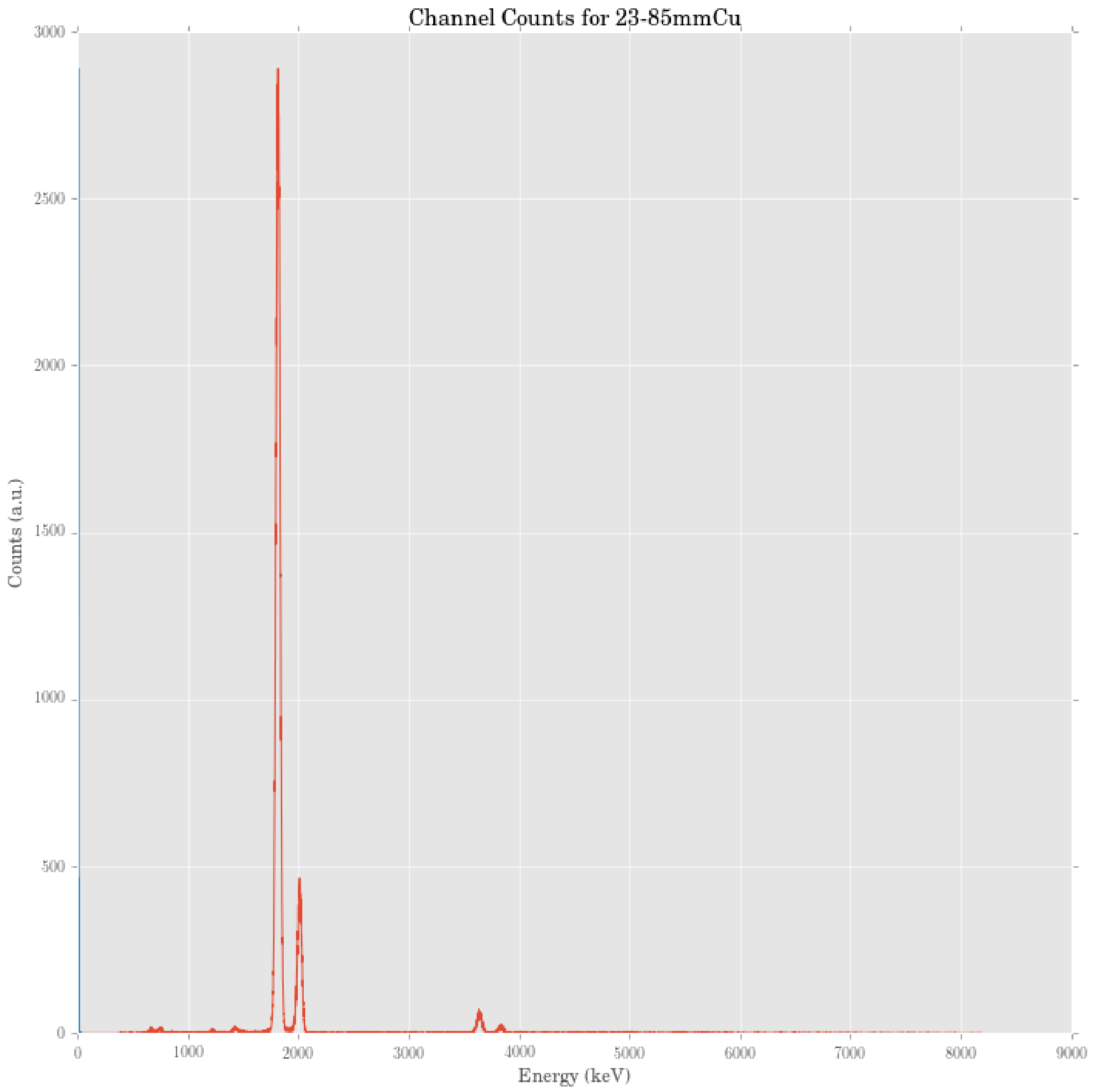


```
In [62]: # Plot all Data Runs
         for file in sorted(os.listdir('SumCrossXRF/Data/')):
         #     print(file.partition('.')[0].partition('_')[2])
             spect = spectrograph('SumCrossXRF/Data/{}'.format(file))
             spect.plot_spectrum(file.partition('.')[0].partition('_')[2])
```

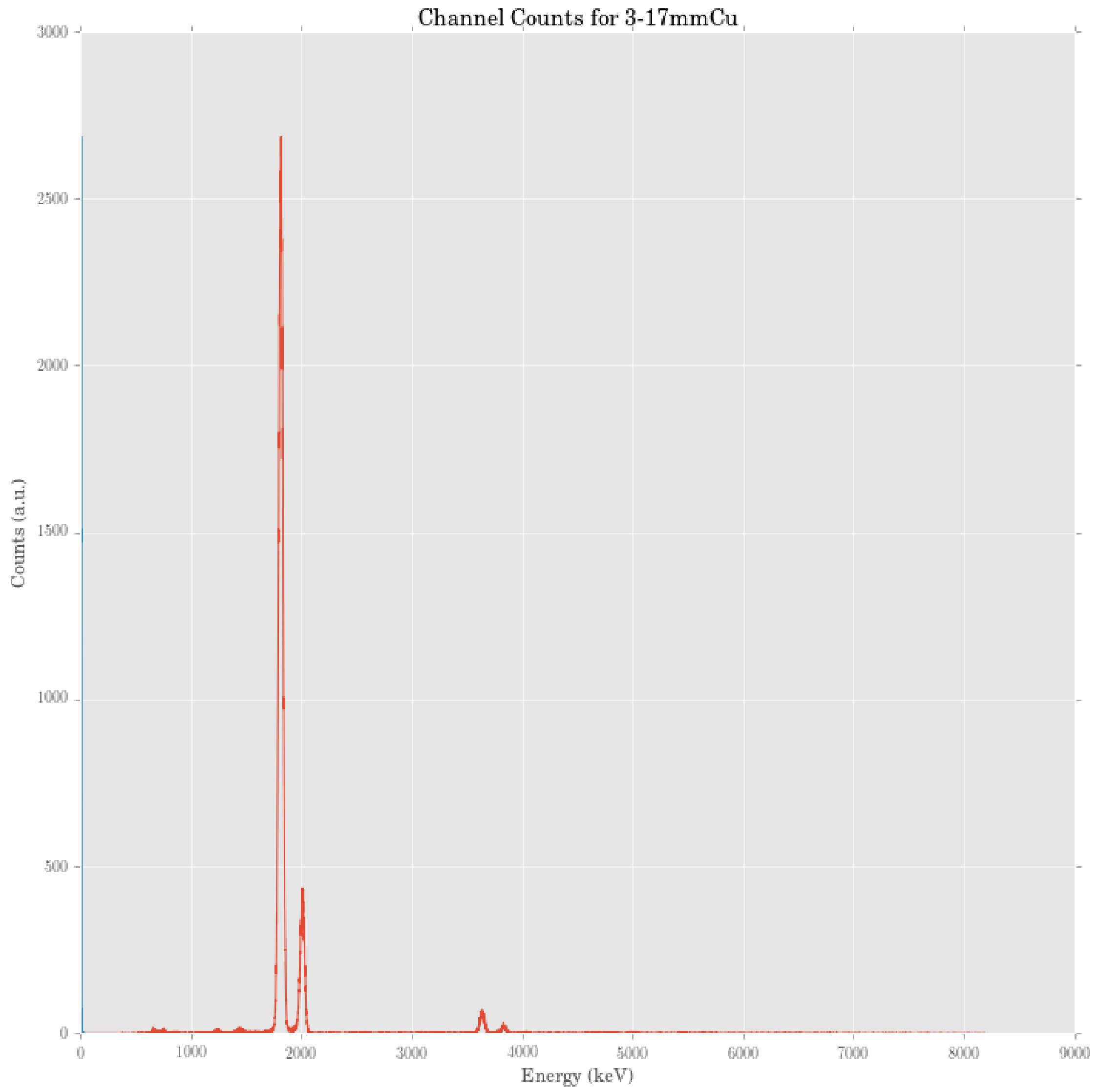

Channel Counts for 0-175mmCu
Counts (a.u.)
Energy (keV)
0
500
1000
1500
2000
2500
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

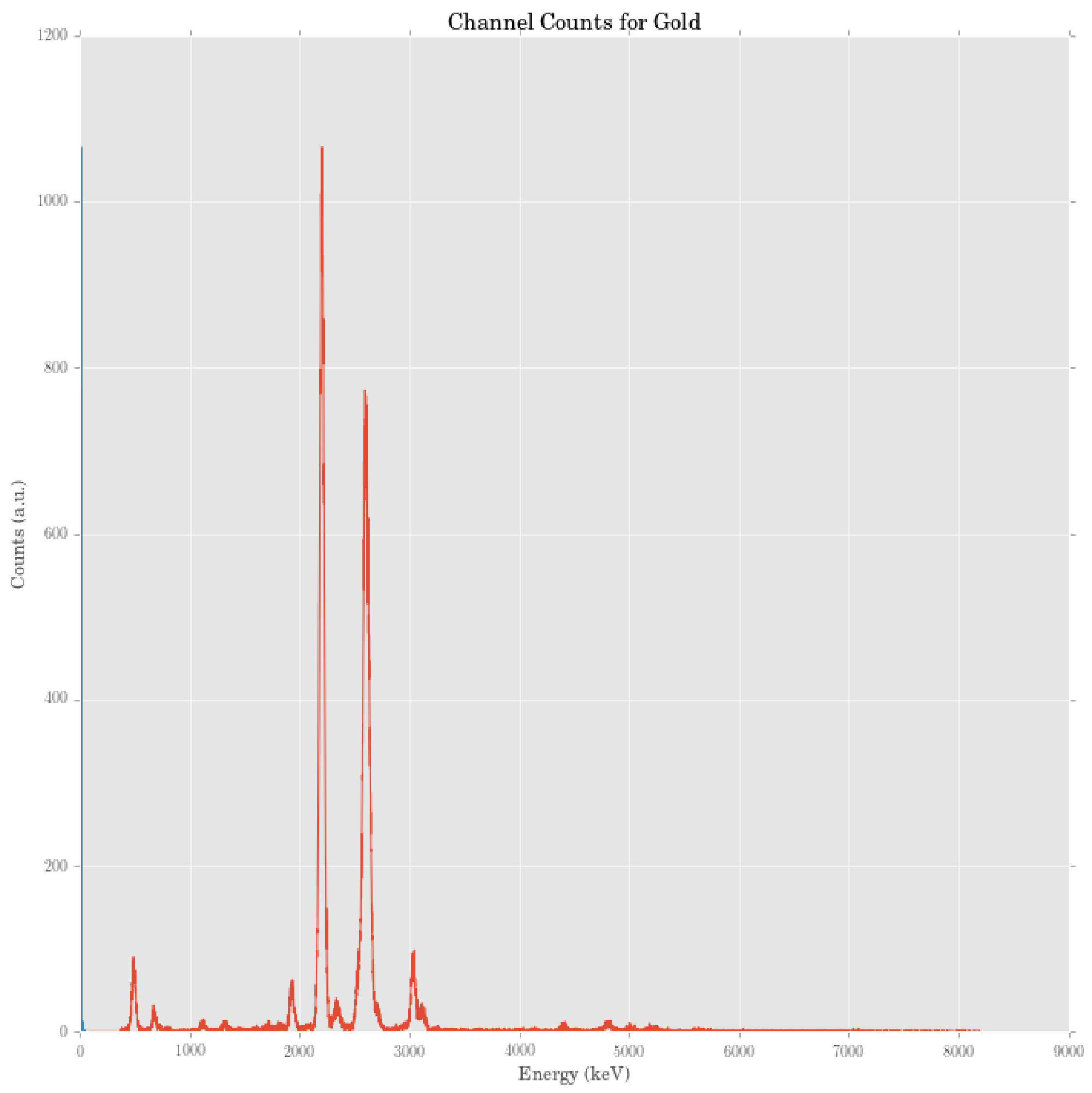

Channel Counts for 0-20mmCu
Counts (a.u.)
Energy (keV)
0
500
1000
1500
2000
2500
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

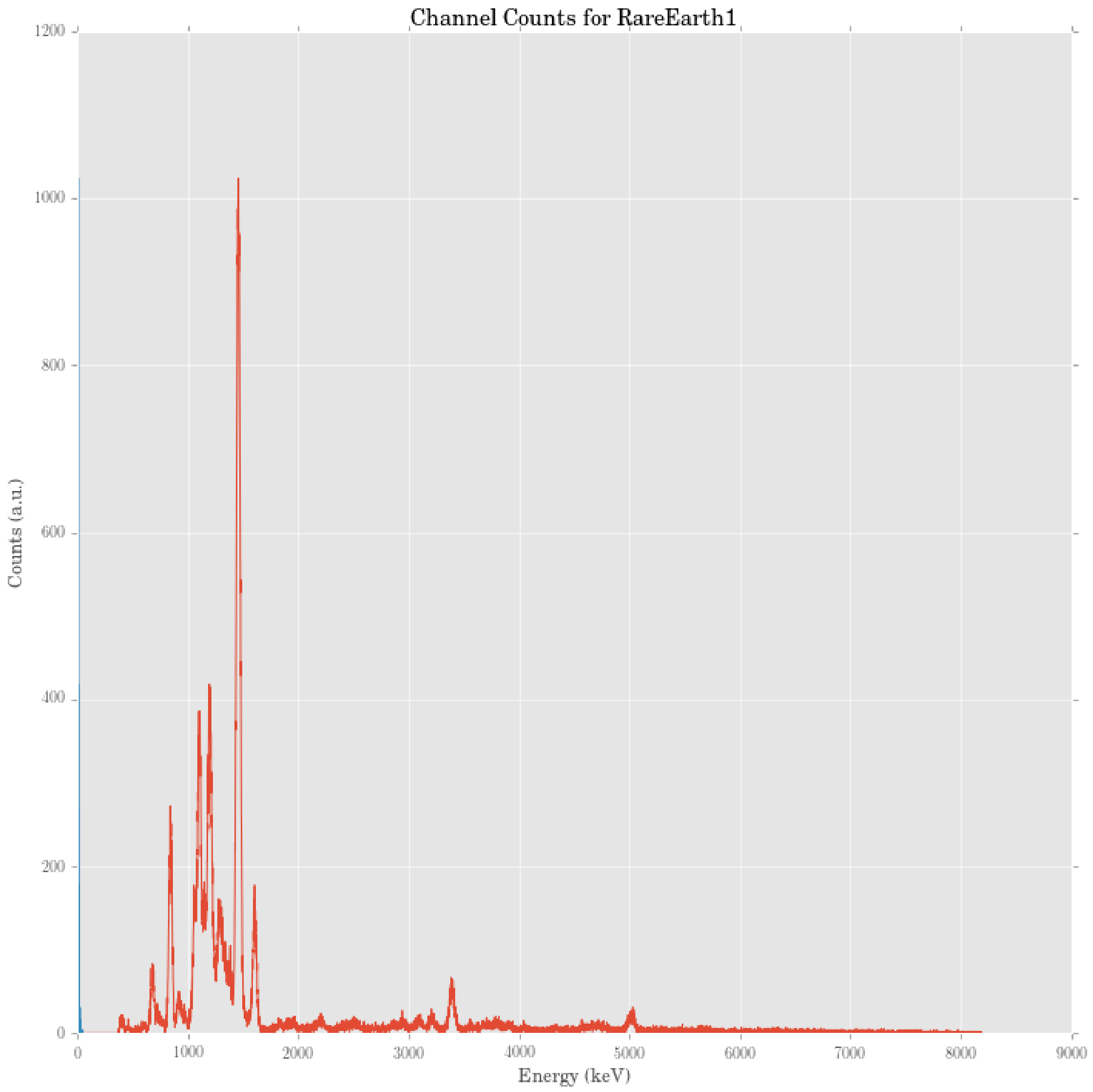

Channel Counts for 20euro
Counts (a.u.)
Energy (keV)
0
500
1000
1500
2000
2500
3000
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

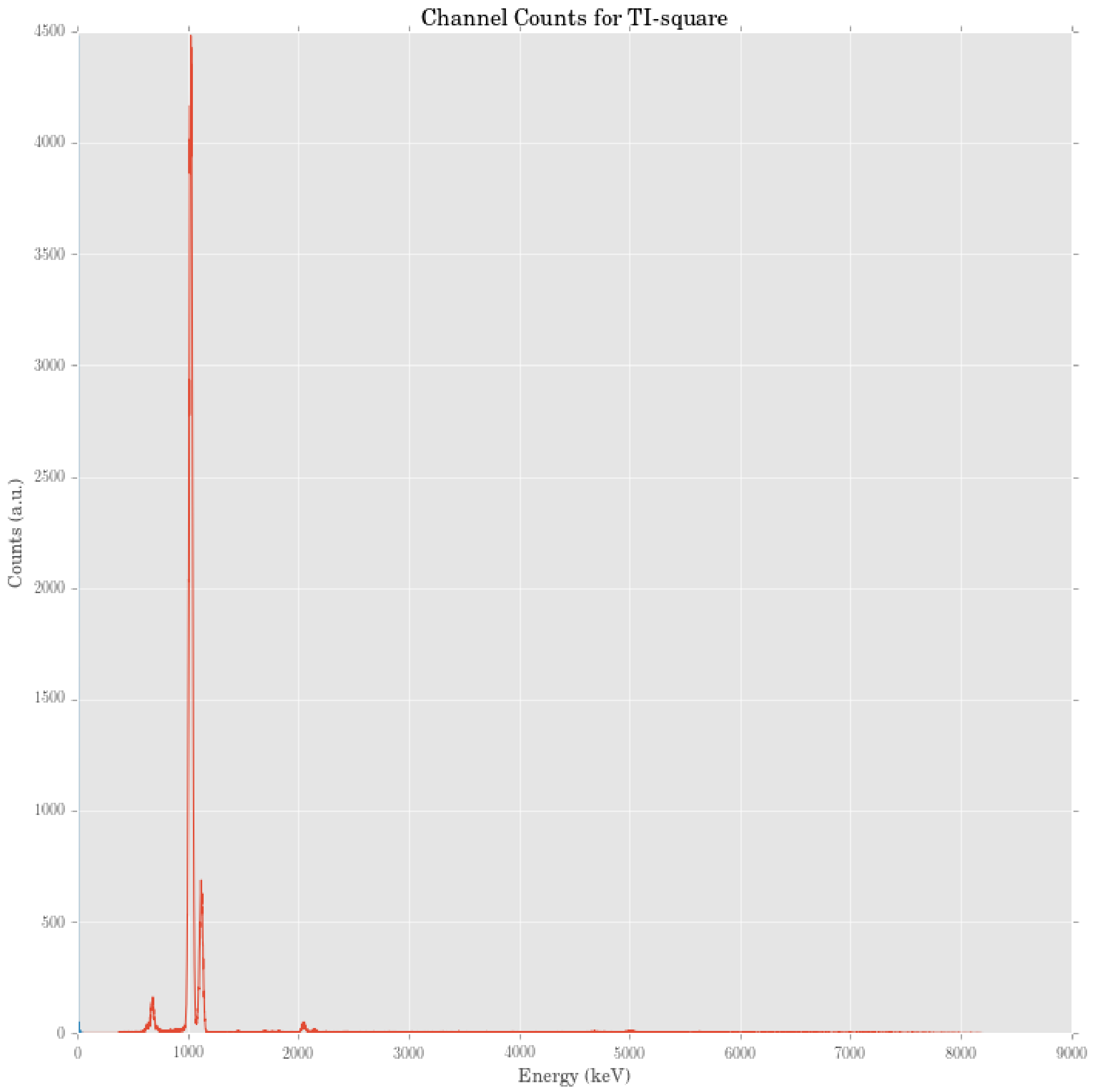


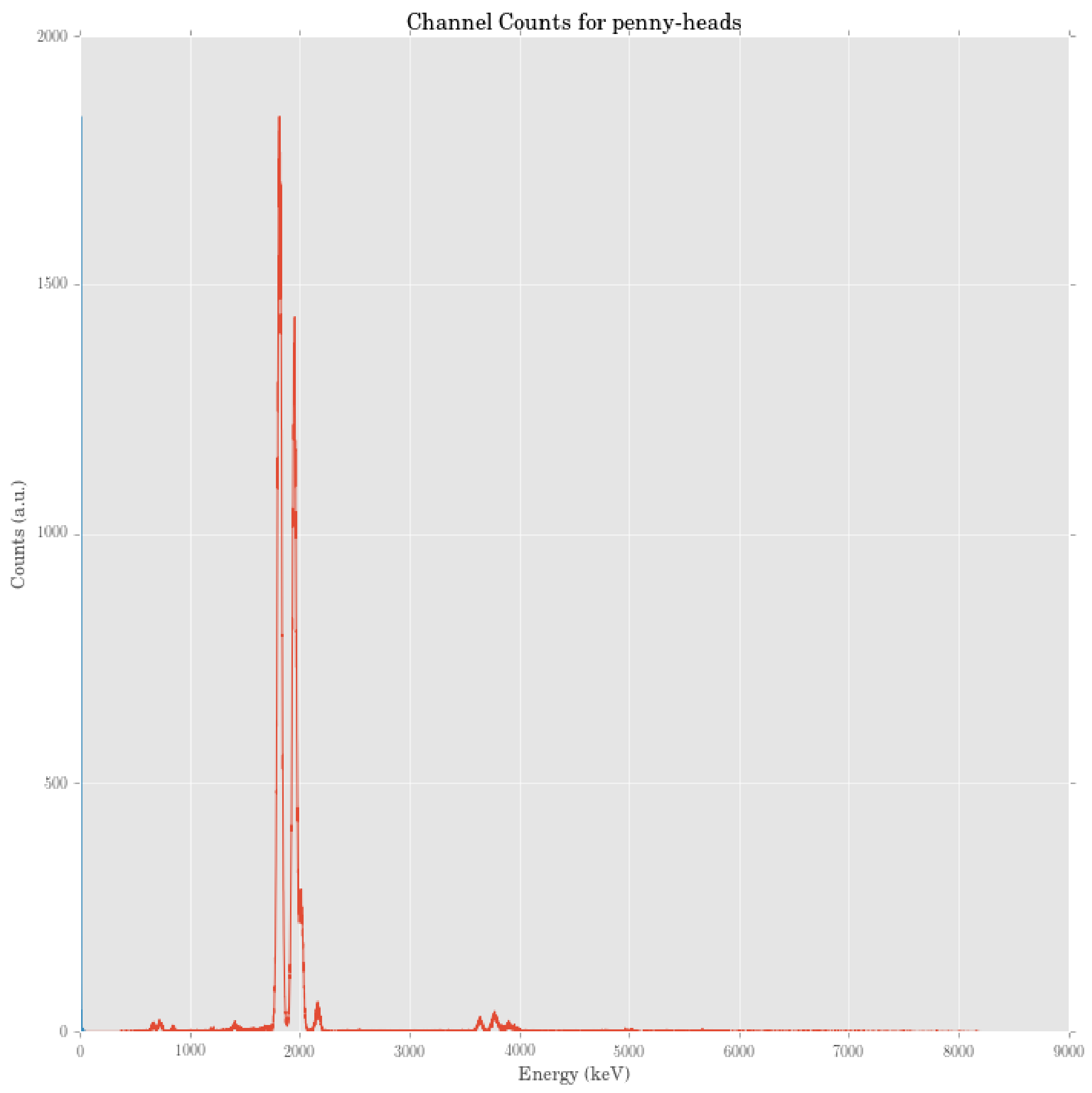
Channel Counts for 23-85mmCu
Counts (a.u.)
Energy (keV)
0
500
1000
1500
2000
2500
3000
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

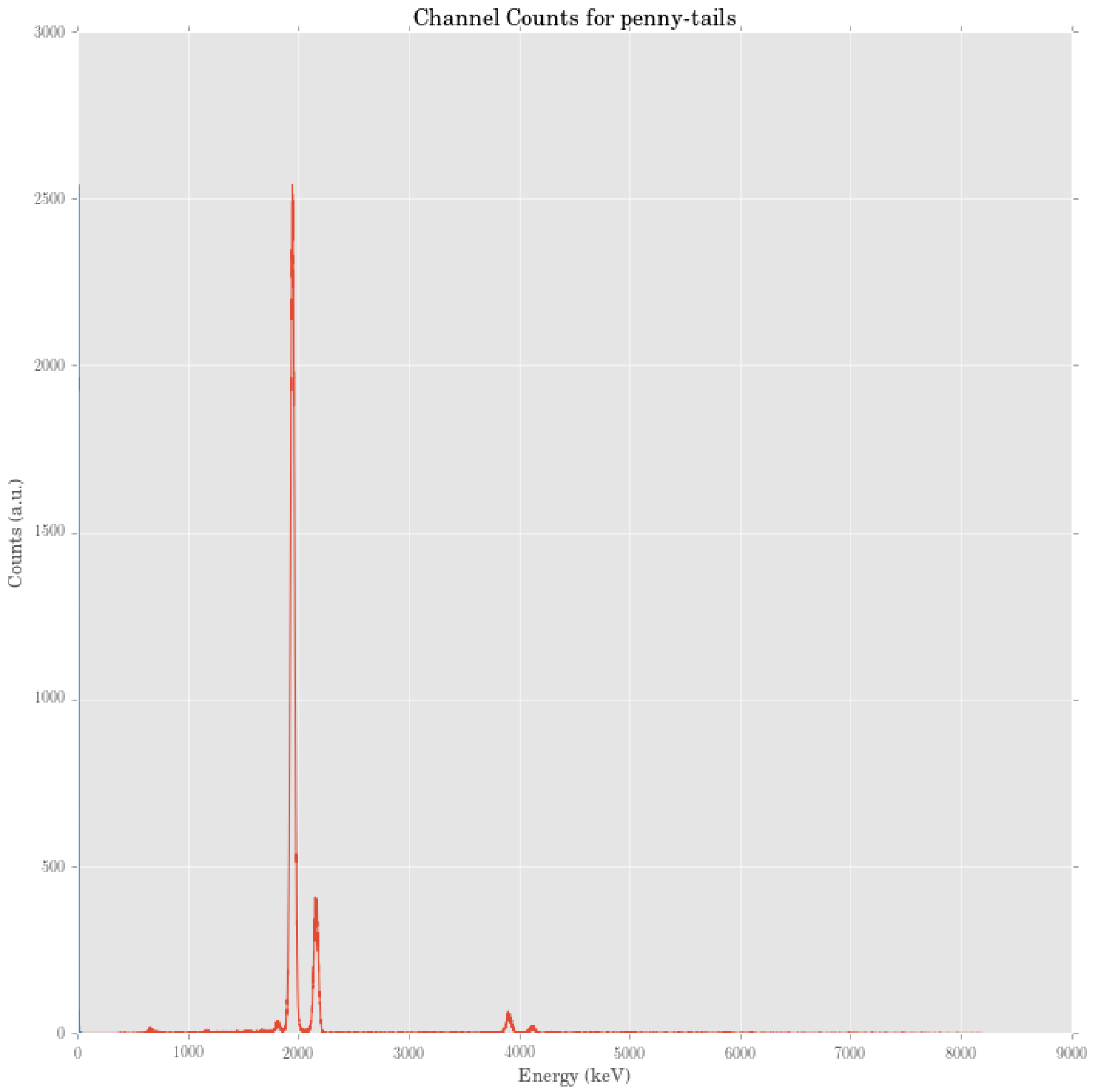

Channel Counts for 3-17mmCu
Counts (a.u.)
Energy (keV)
0
500
1000
1500
2000
2500
3000
0
1000
2000
3000
4000
5000
6000
7000
8000
9000


Channel Counts for Gold
Counts (a.u.)
Energy (keV)
0
200
400
600
800
1000
1200
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

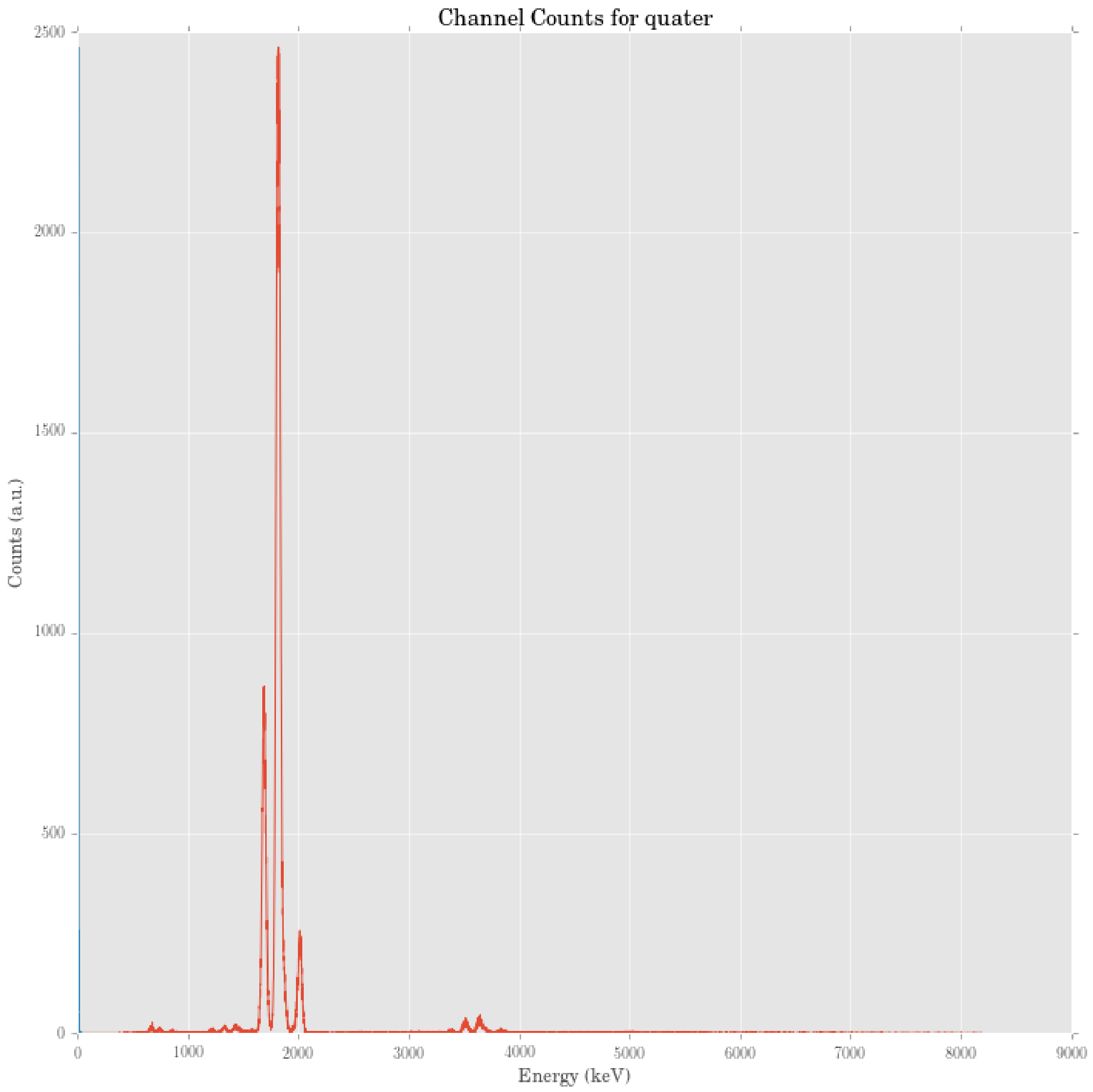

Channel Counts for RareEarth1
Counts (a.u.)
Energy (keV)
0
200
400
600
800
1000
1200
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

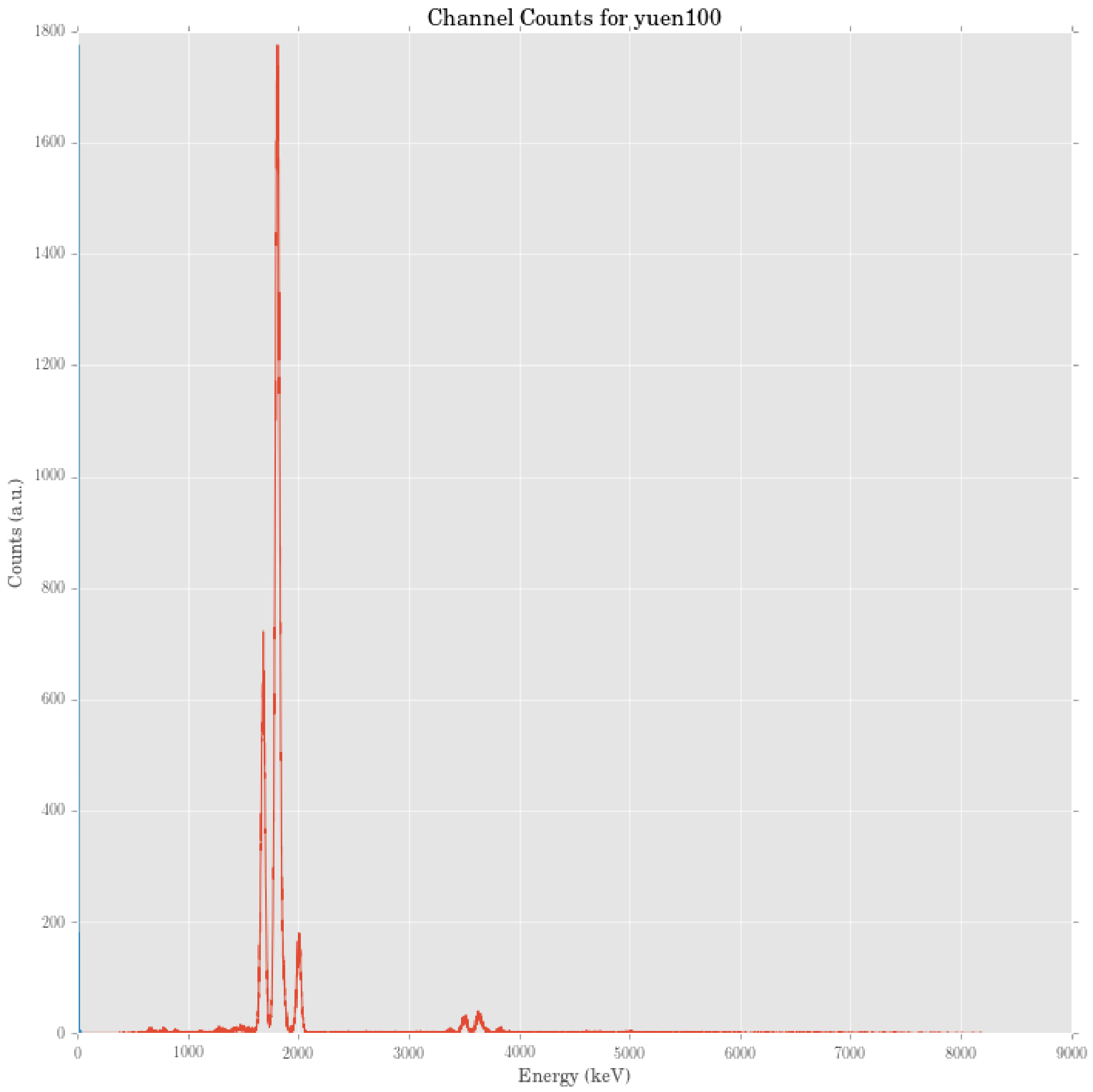

Channel Counts for TI-square
Counts (a.u.)
Energy (keV)
0
500
1000
1500
2000
2500
3000
3500
4000
4500
0
1000
2000
3000
4000
5000
6000
7000
8000
9000


Channel Counts for penny-heads
Counts (a.u.)
Energy (keV)
0
500
1000
1500
2000
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

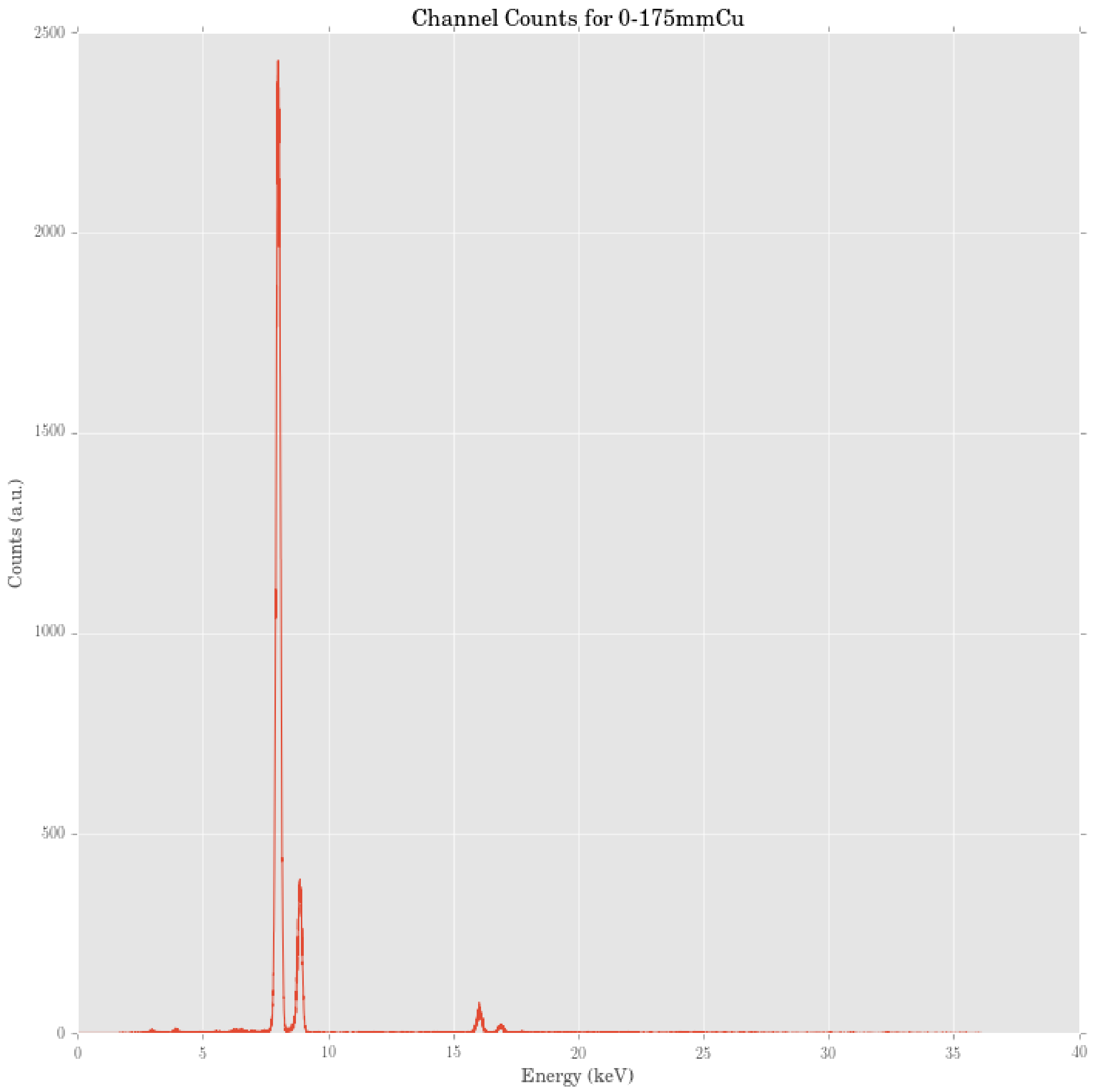


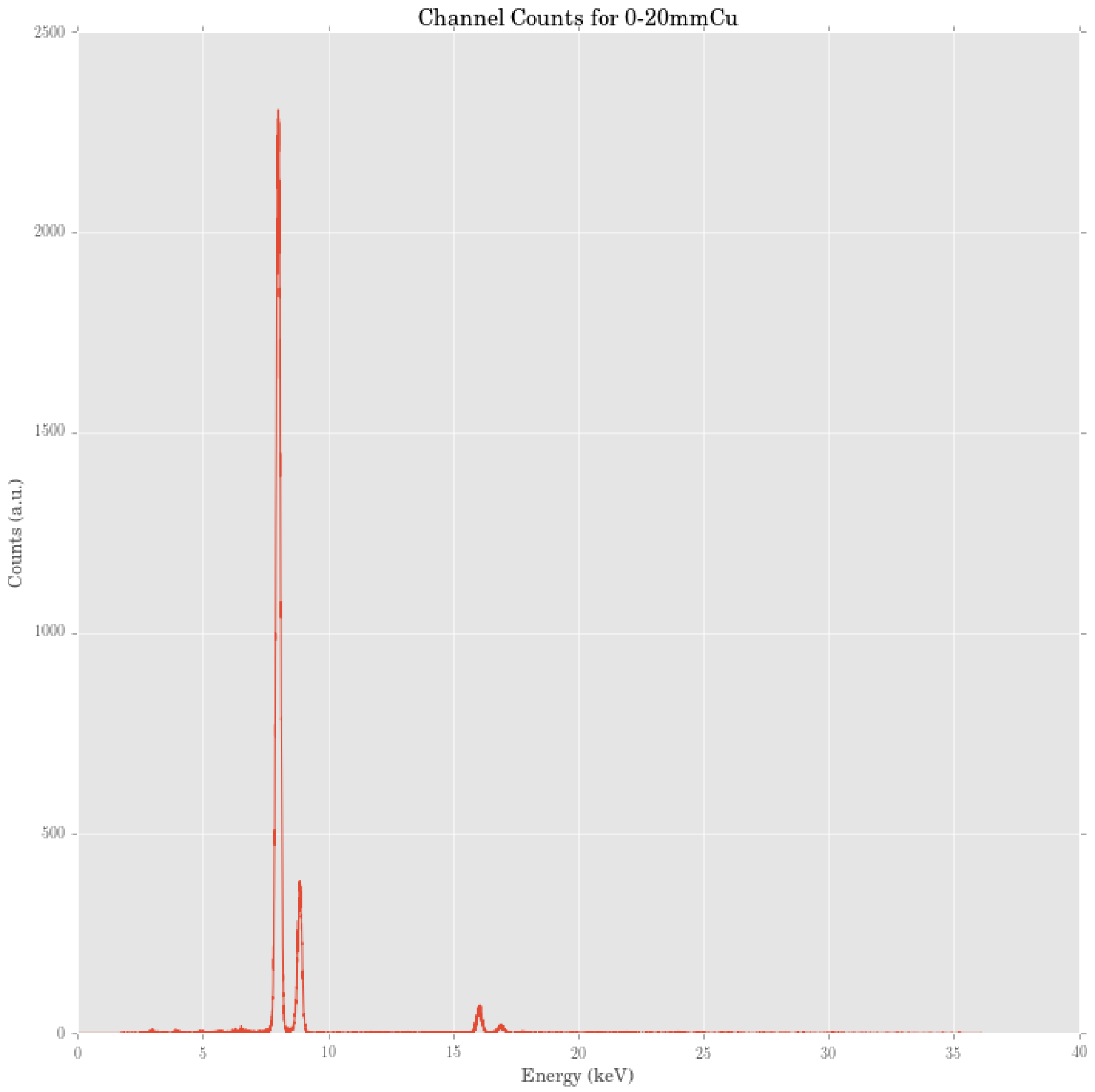
Channel Counts for penny-tails
Counts (a.u.)
Energy (keV)
0
500
1000
1500
2000
2500
3000
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

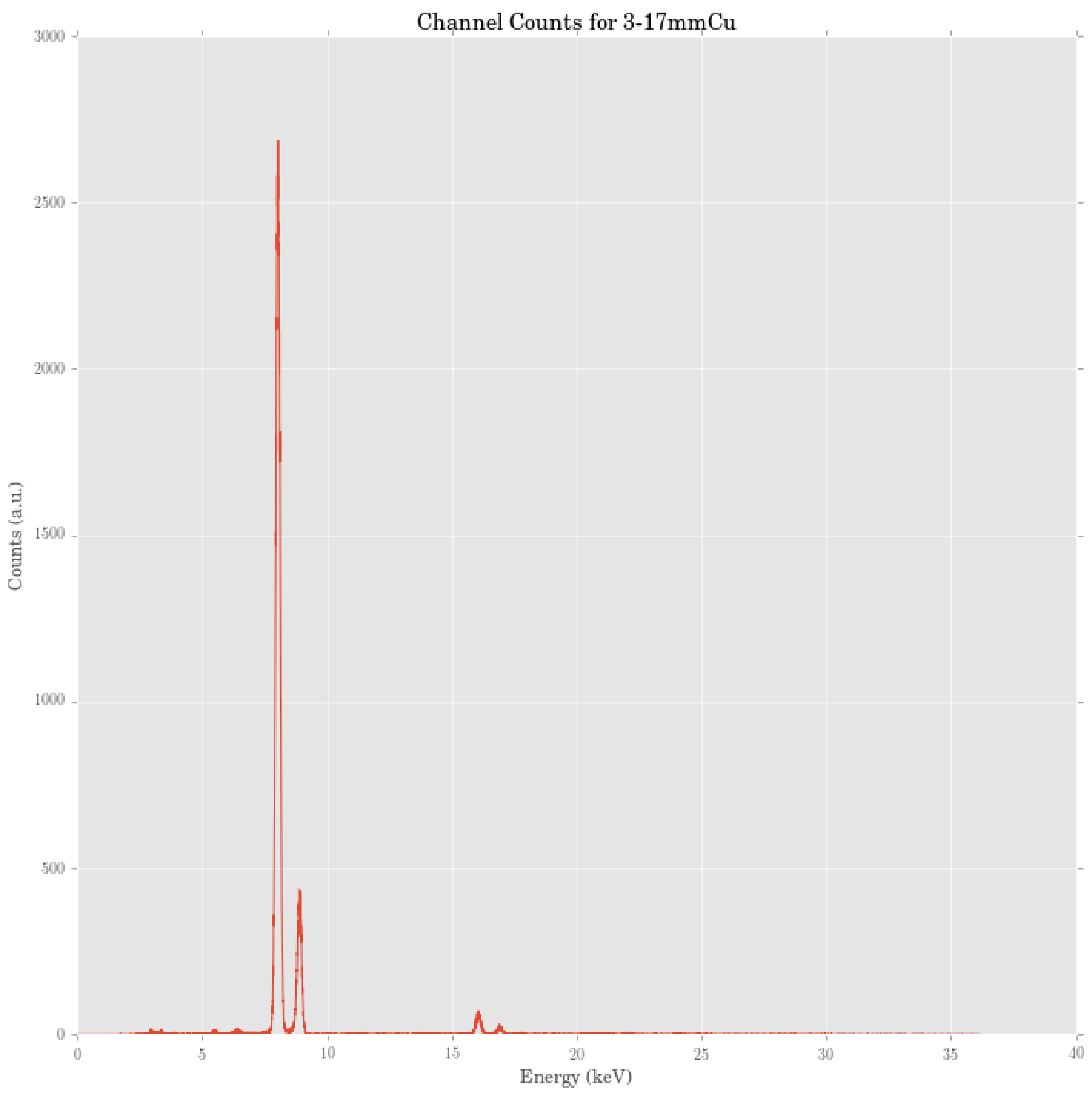

Channel Counts for quater
Counts (a.u.)
Energy (keV)
0
500
1000
1500
2000
2500
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

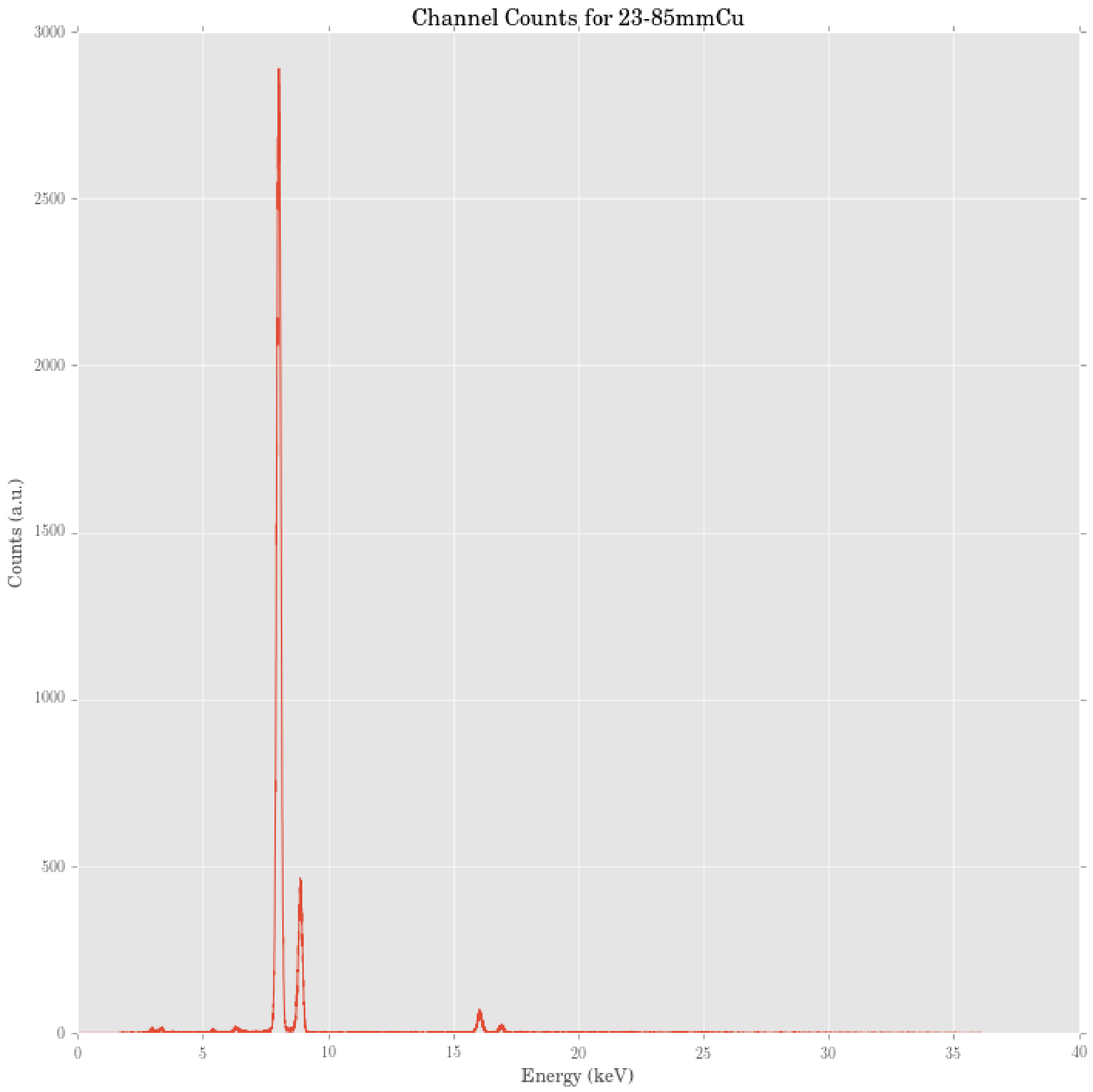


```
In [37]: # Plot all copper data runs for width comparisons
         for file in ['SumCrossXRF/Data/7Mar_0-175mmCu.txt', 'SumCrossXRF/Data/7Mar_0-20mmCu.txt
             spect = spectrograph(file)
             spect.plot_spectrum(file.partition('.')[0].partition('_')[2])
```


Channel Counts for 0-175mmCu
Counts (a.u.)
Energy (keV)
0
500
1000
1500
2000
2500
0
5
10
15
20
25
30
35
40


Channel Counts for 0-20mmCu
Counts (a.u.)
Energy (keV)
0
500
1000
1500
2000
2500
0
5
10
15
20
25
30
35
40


Channel Counts for 3-17mmCu
Counts (a.u.)
Energy (keV)
0
500
1000
1500
2000
2500
3000
0
5
10
15
20
25
30
35
40



Goal: Do a fitting: If an element shows up in a material, *all* peaks should be showing up in the material spectrum. Create the theoretical spectrum for each element by creating gaussians centered on the theoretical positions. First, we parse data from http://www.med.harvard.edu/jpnm/physics/refs/xrayemis.html to get the location of all x-ray fluorescence peaks. Then, we will use an arbitrary data run from the instrument to generate a plot of these peaks against the same bins as the instrument collected data. We assume a gaussian fit for each peak.

```
In [38]: wavelengths = pd.read_fwf('X-Ray_database', names = ['Z', 'Ka1', 'Ka2', 'Kb1', 'La1', '
                'Lg1'],index_col = 0, skiprows = 2, usecols = [0,2,3,4,5,6,7,8,9]).T
         wavelengths=wavelengths.fillna(0.)
         wavelengths=wavelengths.replace(['-','0'],0.)
         wavelengths
         for i in wavelengths:
             wavelengths[i] = wavelengths[i].astype(float)
         wavelengths.columns.astype('int')
         # len(wavelengths.columns)
         wavelengths
```

```
Out[38]: Z         3       4       5      6       7       8       9      10        11  \
         Ka1  0.0543  0.1085  0.1833  0.277  0.3924  0.5249  0.6768  0.8486  1.04098
         Ka2  0.0000  0.0000  0.0000  0.000  0.0000  0.0000  0.0000  0.8486  1.04098
         Kb1  0.0000  0.0000  0.0000  0.000  0.0000  0.0000  0.0000  0.0000  1.07110
         La1  0.0000  0.0000  0.0000  0.000  0.0000  0.0000  0.0000  0.0000  0.00000
         La2  0.0000  0.0000  0.0000  0.000  0.0000  0.0000  0.0000  0.0000  0.00000
         Lb1  0.0000  0.0000  0.0000  0.000  0.0000  0.0000  0.0000  0.0000  0.00000
         Lb2  0.0000  0.0000  0.0000  0.000  0.0000  0.0000  0.0000  0.0000  0.00000
         Lg1  0.0000  0.0000  0.0000  0.000  0.0000  0.0000  0.0000  0.0000  0.00000

         Z        12   ...        86       87        88        89        90  \
         Ka1  1.2536   ...   83.7800  86.1000   88.4700   90.8840   93.3500
         Ka2  1.2536   ...   81.0700  83.2300   85.4300   87.6700   89.9530
         Kb1  1.3022   ...   94.8700  97.4700  100.1300  102.8500  105.6090
         La1  0.0000   ...   11.7270  12.0313   12.3397   12.6520   12.9687
         La2  0.0000   ...   11.5979  11.8950   12.1962   12.5008   12.8096
         Lb1  0.0000   ...   14.3160  14.7700   15.2358   15.7130   16.2022
         Lb2  0.0000   ...    0.0000  14.4500   14.8414    0.0000   15.6237
         Lg1  0.0000   ...   16.7700  17.3030   17.8490   18.4080   18.9825

         Z          91        92       93       94       95
         Ka1   95.8680   98.4390   0.0000   0.0000   0.0000
         Ka2   92.2870   94.6650   0.0000   0.0000   0.0000
         Kb1  108.4270  111.3000   0.0000   0.0000   0.0000
         La1   13.2907   13.6147  13.9441  14.2786  14.6172
         La2   13.1222   13.4388  13.7597  14.0842  14.4119
         Lb1   16.7020   17.2200  17.7502  18.2937  18.8520
         Lb2   16.0240   16.4283  16.8400  17.2553  17.6765
         Lg1   19.5680   20.1671  20.7848  21.4173  22.0652

         [8 rows x 93 columns]
```

```
In [63]: spectral_data[spectral_data.columns[1:]];
```

```
In [48]: @interact(x0 = (0,9000,50),  = (0,1000,20),  continuous_update=False)
         def make_peak(x0, ):
             x = np.linspace(0, 9000,9000)
             y = *np.sqrt(2*np.pi)*norm.pdf(x, loc = x0, scale = )
             fig, ax = plt.subplots(figsize = (8,8))
             ax.plot(x,y)
             fig.tight_layout()
```

```
interactive(children=(IntSlider(value=4500, description='x0', max=9000, step=50), IntSlider(valu
In [41]: # Create a dataset of the spectrum discretized into the same size as the instrument col
         def make_spectrum(x, wavelengths, ):
             spectrum = np.zeros(len(x))
             for x0 in wavelengths:
                 add = *np.sqrt(2*np.pi)*norm.pdf(x, loc = x0, scale = )
                 if (x0 <1.5 or x0>25.):
                     spectrum += .01**np.sqrt(2*np.pi)*norm.pdf(x, loc = x0, scale = )
                 else:
                     spectrum += *np.sqrt(2*np.pi)*norm.pdf(x, loc = x0, scale = )
             return spectrum

         # x = np.linspace(0,40,9000)
         # # .5*np.sqrt(2*np.pi)*norm.pdf(x, loc = )
         # fig, ax = plt.subplots(figsize = (10,10))
         # ax.plot(x, make_spectrum(x, wavelengths[87], .1))

         spectral_data = {}
         for column in wavelengths:
             spectral_data[column] = make_spectrum(test.file['Energy (keV)'],wavelengths[column]

         spectral_data = pd.DataFrame(spectral_data)
         spectral_data.insert(0,'Energy (keV)',test.file['Energy (keV)'])
         spectral_data
         # len(spectral_data)

         # fig, ax = plt.subplots()
         # ax.plot(spectral_data['Energy (keV)'],spectral_data[29])
         # spectral_data[5]
```

```
Out[41]:      Energy (keV)         3         4         5         6         7  \
         0        0.045407  0.073104  0.071338  0.067008  0.063828  0.063167
         1        0.049811  0.071823  0.070251  0.065936  0.062590  0.061861
         2        0.054214  0.070433  0.069063  0.064780  0.061269  0.060466
         3        0.058617  0.068941  0.067781  0.063547  0.059872  0.058989
         4        0.063020  0.067355  0.066410  0.062244  0.058406  0.057437
         5        0.067424  0.065682  0.064959  0.060878  0.056880  0.055819
         6        0.071827  0.063932  0.063434  0.059456  0.055303  0.054143
         7        0.076230  0.062112  0.061842  0.057987  0.053682  0.052417
         8        0.080634  0.060232  0.060192  0.056476  0.052027  0.050650
         9        0.085037  0.058300  0.058489  0.054932  0.050345  0.048850
         10       0.089440  0.056325  0.056743  0.053361  0.048646  0.047025
         11       0.093844  0.054316  0.054961  0.051770  0.046936  0.045184
         12       0.098247  0.052281  0.053149  0.050166  0.045225  0.043334
         13       0.102650  0.050229  0.051315  0.048556  0.043520  0.041482
         14       0.107054  0.048168  0.049466  0.046945  0.041827  0.039638
         15       0.111457  0.046106  0.047609  0.045339  0.040154  0.037807
         16       0.115860  0.044051  0.045750  0.043743  0.038507  0.035996
         17       0.120264  0.042010  0.043896  0.042163  0.036893  0.034211
```

```
18         0.124667  0.039989  0.042052  0.040603  0.035316  0.032460
19         0.129070  0.037995  0.040224  0.039066  0.033782  0.030745
20         0.133473  0.036034  0.038417  0.037557  0.032294  0.029074
21         0.137877  0.034110  0.036636  0.036078  0.030858  0.027450
22         0.142280  0.032231  0.034885  0.034633  0.029475  0.025878
23         0.146683  0.030398  0.033169  0.033223  0.028150  0.024360
24         0.151087  0.028617  0.031490  0.031851  0.026883  0.022901
25         0.155490  0.026891  0.029852  0.030518  0.025677  0.021502
26         0.159893  0.025222  0.028259  0.029226  0.024533  0.020166
27         0.164297  0.023614  0.026711  0.027974  0.023452  0.018894
28         0.168700  0.022067  0.025212  0.026764  0.022433  0.017689
29         0.173103  0.020584  0.023763  0.025595  0.021476  0.016550
...             ...       ...       ...       ...       ...       ...
8161      35.980840  0.000000  0.000000  0.000000  0.000000  0.000000
8162      35.985243  0.000000  0.000000  0.000000  0.000000  0.000000
8163      35.989646  0.000000  0.000000  0.000000  0.000000  0.000000
8164      35.994050  0.000000  0.000000  0.000000  0.000000  0.000000
8165      35.998453  0.000000  0.000000  0.000000  0.000000  0.000000
8166      36.002856  0.000000  0.000000  0.000000  0.000000  0.000000
8167      36.007260  0.000000  0.000000  0.000000  0.000000  0.000000
8168      36.011663  0.000000  0.000000  0.000000  0.000000  0.000000
8169      36.016066  0.000000  0.000000  0.000000  0.000000  0.000000
8170      36.020469  0.000000  0.000000  0.000000  0.000000  0.000000
8171      36.024873  0.000000  0.000000  0.000000  0.000000  0.000000
8172      36.029276  0.000000  0.000000  0.000000  0.000000  0.000000
8173      36.033679  0.000000  0.000000  0.000000  0.000000  0.000000
8174      36.038083  0.000000  0.000000  0.000000  0.000000  0.000000
8175      36.042486  0.000000  0.000000  0.000000  0.000000  0.000000
8176      36.046889  0.000000  0.000000  0.000000  0.000000  0.000000
8177      36.051293  0.000000  0.000000  0.000000  0.000000  0.000000
8178      36.055696  0.000000  0.000000  0.000000  0.000000  0.000000
8179      36.060099  0.000000  0.000000  0.000000  0.000000  0.000000
8180      36.064503  0.000000  0.000000  0.000000  0.000000  0.000000
8181      36.068906  0.000000  0.000000  0.000000  0.000000  0.000000
8182      36.073309  0.000000  0.000000  0.000000  0.000000  0.000000
8183      36.077713  0.000000  0.000000  0.000000  0.000000  0.000000
8184      36.082116  0.000000  0.000000  0.000000  0.000000  0.000000
8185      36.086519  0.000000  0.000000  0.000000  0.000000  0.000000
8186      36.090922  0.000000  0.000000  0.000000  0.000000  0.000000
8187      36.095326  0.000000  0.000000  0.000000  0.000000  0.000000
8188      36.099729  0.000000  0.000000  0.000000  0.000000  0.000000
8189      36.104132  0.000000  0.000000  0.000000  0.000000  0.000000
8190      36.108536  0.000000  0.000000  0.000000  0.000000  0.000000

             8         9        10        11    ...          86  87  88  \
0     0.063143  0.063143  0.054123  0.045102    ...    0.009020   0   0
1     0.061833  0.061833  0.053000  0.044167    ...    0.008833   0   0
2     0.060433  0.060433  0.051800  0.043167    ...    0.008633   0   0
```

```
3      0.058951  0.058950  0.050529  0.042107     ...      0.008421   0   0
4      0.057393  0.057393  0.049194  0.040995     ...      0.008199   0   0
5      0.055768  0.055768  0.047801  0.039834     ...      0.007967   0   0
6      0.054084  0.054084  0.046358  0.038631     ...      0.007726   0   0
7      0.052350  0.052349  0.044871  0.037392     ...      0.007478   0   0
8      0.050573  0.050572  0.043348  0.036123     ...      0.007225   0   0
9      0.048762  0.048761  0.041795  0.034829     ...      0.006966   0   0
10     0.046924  0.046923  0.040220  0.033517     ...      0.006703   0   0
11     0.045069  0.045068  0.038629  0.032191     ...      0.006438   0   0
12     0.043202  0.043201  0.037030  0.030858     ...      0.006172   0   0
13     0.041334  0.041332  0.035428  0.029523     ...      0.005905   0   0
14     0.039469  0.039467  0.033829  0.028191     ...      0.005638   0   0
15     0.037615  0.037614  0.032240  0.026867     ...      0.005373   0   0
16     0.035780  0.035777  0.030666  0.025555     ...      0.005111   0   0
17     0.033968  0.033965  0.029113  0.024261     ...      0.004852   0   0
18     0.032185  0.032182  0.027584  0.022987     ...      0.004597   0   0
19     0.030437  0.030433  0.026086  0.021738     ...      0.004348   0   0
20     0.028729  0.028724  0.024621  0.020517     ...      0.004103   0   0
21     0.027064  0.027058  0.023193  0.019327     ...      0.003865   0   0
22     0.025446  0.025440  0.021806  0.018171     ...      0.003634   0   0
23     0.023880  0.023872  0.020461  0.017051     ...      0.003410   0   0
24     0.022366  0.022357  0.019163  0.015969     ...      0.003194   0   0
25     0.020909  0.020898  0.017912  0.014927     ...      0.002985   0   0
26     0.019509  0.019496  0.016711  0.013926     ...      0.002785   0   0
27     0.018168  0.018153  0.015560  0.012966     ...      0.002593   0   0
28     0.016887  0.016870  0.014460  0.012050     ...      0.002410   0   0
29     0.015667  0.015647  0.013411  0.011176     ...      0.002235   0   0
...         ...       ...       ...       ...     ...           ...  ..  ..
8161   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8162   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8163   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8164   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8165   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8166   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8167   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8168   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8169   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8170   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8171   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8172   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8173   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8174   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8175   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8176   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8177   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8178   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8179   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
8180   0.000000  0.000000  0.000000  0.000000     ...      0.000000   0   0
```

```
8181  0.000000  0.000000  0.000000  0.000000    ...     0.000000   0   0
8182  0.000000  0.000000  0.000000  0.000000    ...     0.000000   0   0
8183  0.000000  0.000000  0.000000  0.000000    ...     0.000000   0   0
8184  0.000000  0.000000  0.000000  0.000000    ...     0.000000   0   0
8185  0.000000  0.000000  0.000000  0.000000    ...     0.000000   0   0
8186  0.000000  0.000000  0.000000  0.000000    ...     0.000000   0   0
8187  0.000000  0.000000  0.000000  0.000000    ...     0.000000   0   0
8188  0.000000  0.000000  0.000000  0.000000    ...     0.000000   0   0
8189  0.000000  0.000000  0.000000  0.000000    ...     0.000000   0   0
8190  0.000000  0.000000  0.000000  0.000000    ...     0.000000   0   0

            89  90  91  92        93        94        95
0     0.009020   0   0   0  0.027061  0.027061  0.027061
1     0.008833   0   0   0  0.026500  0.026500  0.026500
2     0.008633   0   0   0  0.025900  0.025900  0.025900
3     0.008421   0   0   0  0.025264  0.025264  0.025264
4     0.008199   0   0   0  0.024597  0.024597  0.024597
5     0.007967   0   0   0  0.023900  0.023900  0.023900
6     0.007726   0   0   0  0.023179  0.023179  0.023179
7     0.007478   0   0   0  0.022435  0.022435  0.022435
8     0.007225   0   0   0  0.021674  0.021674  0.021674
9     0.006966   0   0   0  0.020898  0.020898  0.020898
10    0.006703   0   0   0  0.020110  0.020110  0.020110
11    0.006438   0   0   0  0.019315  0.019315  0.019315
12    0.006172   0   0   0  0.018515  0.018515  0.018515
13    0.005905   0   0   0  0.017714  0.017714  0.017714
14    0.005638   0   0   0  0.016915  0.016915  0.016915
15    0.005373   0   0   0  0.016120  0.016120  0.016120
16    0.005111   0   0   0  0.015333  0.015333  0.015333
17    0.004852   0   0   0  0.014556  0.014556  0.014556
18    0.004597   0   0   0  0.013792  0.013792  0.013792
19    0.004348   0   0   0  0.013043  0.013043  0.013043
20    0.004103   0   0   0  0.012310  0.012310  0.012310
21    0.003865   0   0   0  0.011596  0.011596  0.011596
22    0.003634   0   0   0  0.010903  0.010903  0.010903
23    0.003410   0   0   0  0.010231  0.010231  0.010231
24    0.003194   0   0   0  0.009582  0.009582  0.009582
25    0.002985   0   0   0  0.008956  0.008956  0.008956
26    0.002785   0   0   0  0.008355  0.008355  0.008355
27    0.002593   0   0   0  0.007780  0.007780  0.007780
28    0.002410   0   0   0  0.007230  0.007230  0.007230
29    0.002235   0   0   0  0.006706  0.006706  0.006706
...        ...  ..  ..  ..       ...       ...       ...
8161  0.000000   0   0   0  0.000000  0.000000  0.000000
8162  0.000000   0   0   0  0.000000  0.000000  0.000000
8163  0.000000   0   0   0  0.000000  0.000000  0.000000
8164  0.000000   0   0   0  0.000000  0.000000  0.000000
8165  0.000000   0   0   0  0.000000  0.000000  0.000000
```

```
8166  0.000000  0  0  0  0.000000  0.000000  0.000000
8167  0.000000  0  0  0  0.000000  0.000000  0.000000
8168  0.000000  0  0  0  0.000000  0.000000  0.000000
8169  0.000000  0  0  0  0.000000  0.000000  0.000000
8170  0.000000  0  0  0  0.000000  0.000000  0.000000
8171  0.000000  0  0  0  0.000000  0.000000  0.000000
8172  0.000000  0  0  0  0.000000  0.000000  0.000000
8173  0.000000  0  0  0  0.000000  0.000000  0.000000
8174  0.000000  0  0  0  0.000000  0.000000  0.000000
8175  0.000000  0  0  0  0.000000  0.000000  0.000000
8176  0.000000  0  0  0  0.000000  0.000000  0.000000
8177  0.000000  0  0  0  0.000000  0.000000  0.000000
8178  0.000000  0  0  0  0.000000  0.000000  0.000000
8179  0.000000  0  0  0  0.000000  0.000000  0.000000
8180  0.000000  0  0  0  0.000000  0.000000  0.000000
8181  0.000000  0  0  0  0.000000  0.000000  0.000000
8182  0.000000  0  0  0  0.000000  0.000000  0.000000
8183  0.000000  0  0  0  0.000000  0.000000  0.000000
8184  0.000000  0  0  0  0.000000  0.000000  0.000000
8185  0.000000  0  0  0  0.000000  0.000000  0.000000
8186  0.000000  0  0  0  0.000000  0.000000  0.000000
8187  0.000000  0  0  0  0.000000  0.000000  0.000000
8188  0.000000  0  0  0  0.000000  0.000000  0.000000
8189  0.000000  0  0  0  0.000000  0.000000  0.000000
8190  0.000000  0  0  0  0.000000  0.000000  0.000000

[8191 rows x 94 columns]
```

We wish to find an expression

$$A_3 * \text{Spectrum}(Z = 3) + A_4 * \text{Spectrum}(Z = 4) + ... + A_{95} * \text{Spectrum}(Z = 95) = \text{Spectrum}(Data) \tag{1}$$

such that the A_i represents how much of the $Z = i$ is present in the sample. We have an abundance of data, as each point represents a value that, as a scalar, must satisfy the above equation. Thus we convert this equation into its matrix form $\mathbf{A}\vec{x} = \vec{b}$. Each column of the above matrix (dataframe) represents the column of the coefficient matrix, $\vec{x}$ is the column vector with elements A_i, and $\vec{b}$ is the column vector describing the data obtained.

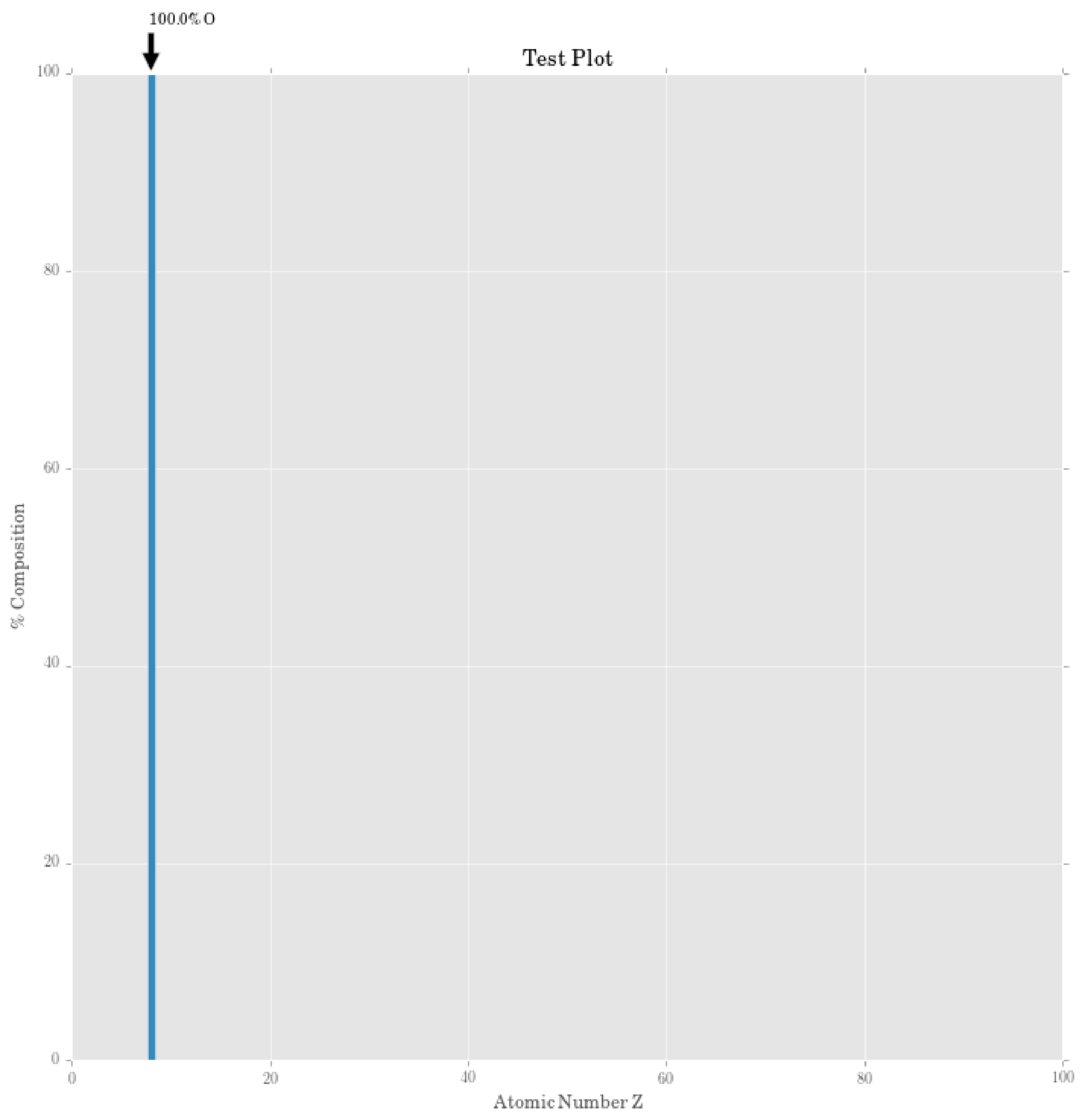
Because we have an overdetermined system, we do a linear least squares fitting to find the best 'fit'. However, we must add the constraint that $A_i >= 0$, since we can't have spectral subtraction. For this, we use FORTRAN's non-negative least squares solver, implemented in scipy.optimize.nnls

```
In [42]: Z_dict = pd.read_fwf('X-Ray_database', names = ['Z', 'Element'],index_col = 0, skiprows
         # Z_dict['Element'].astype('str')
         # print(Z_dict.iloc[16])
         Z_dict.iloc[8]['Element']
         Z_dict
```

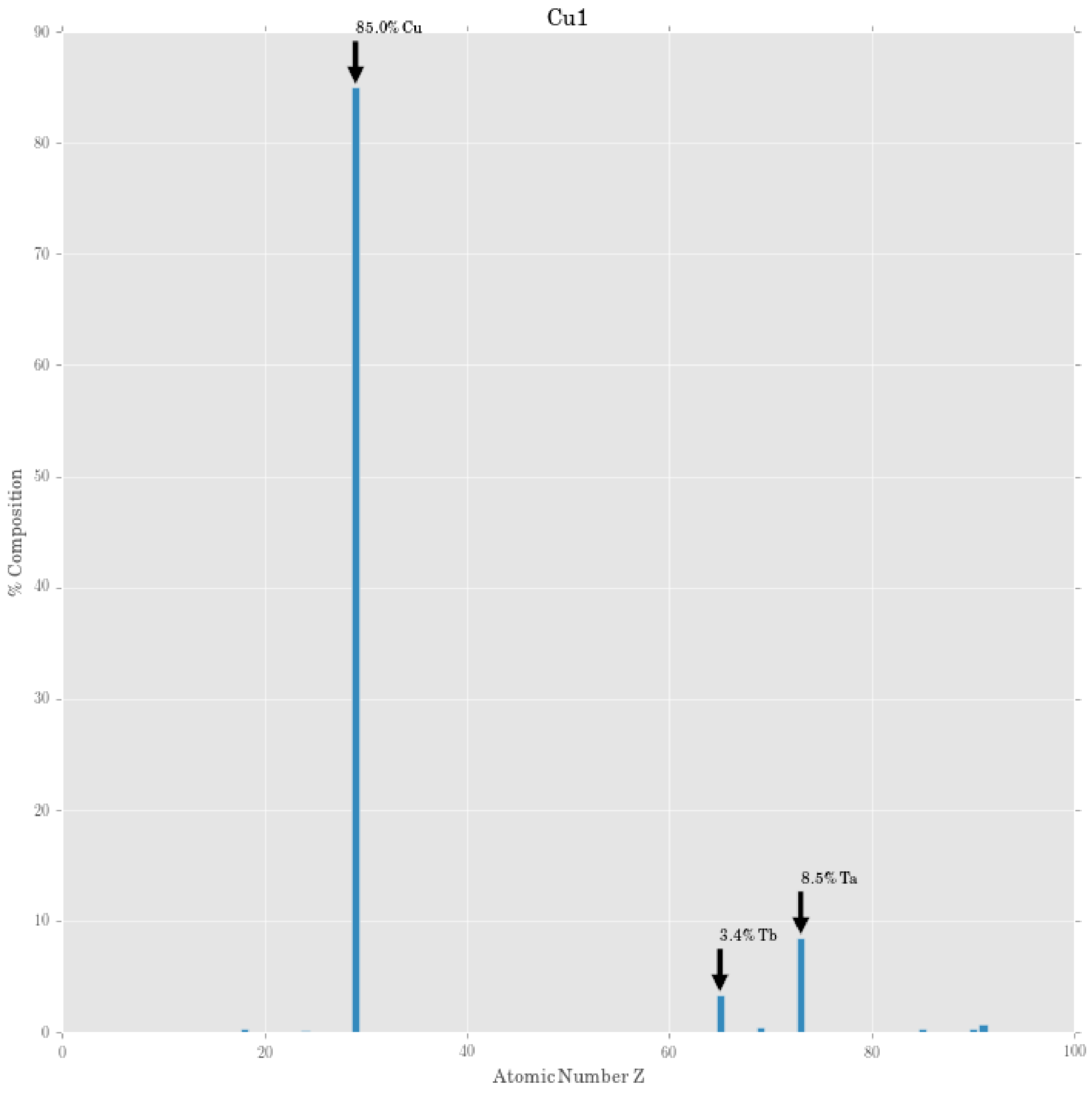
```
Out[42]:     Element
         Z
         3        Li
         4        Be
         5         B
         6         C
         7         N
         8         O
         9         F
         10       Ne
         11       Na
         12       Mg
         13       Al
         14       Si
         15        P
         16        S
         17       Cl
         18       Ar
         19        K
         20       Ca
         21       Sc
         22       Ti
         23        V
         24       Cr
         25       Mn
         26       Fe
         27       Co
         28       Ni
         29       Cu
         30       Zn
         31       Ga
         32       Ge
         ..      ...
         66       Dy
         67       Ho
         68       Er
         69       Tm
         70       Yb
         71       Lu
         72       Hf
         73       Ta
         74        W
         75       Re
         76       Os
         77       Ir
         78       Pt
         79       Au
         80       Hg
```

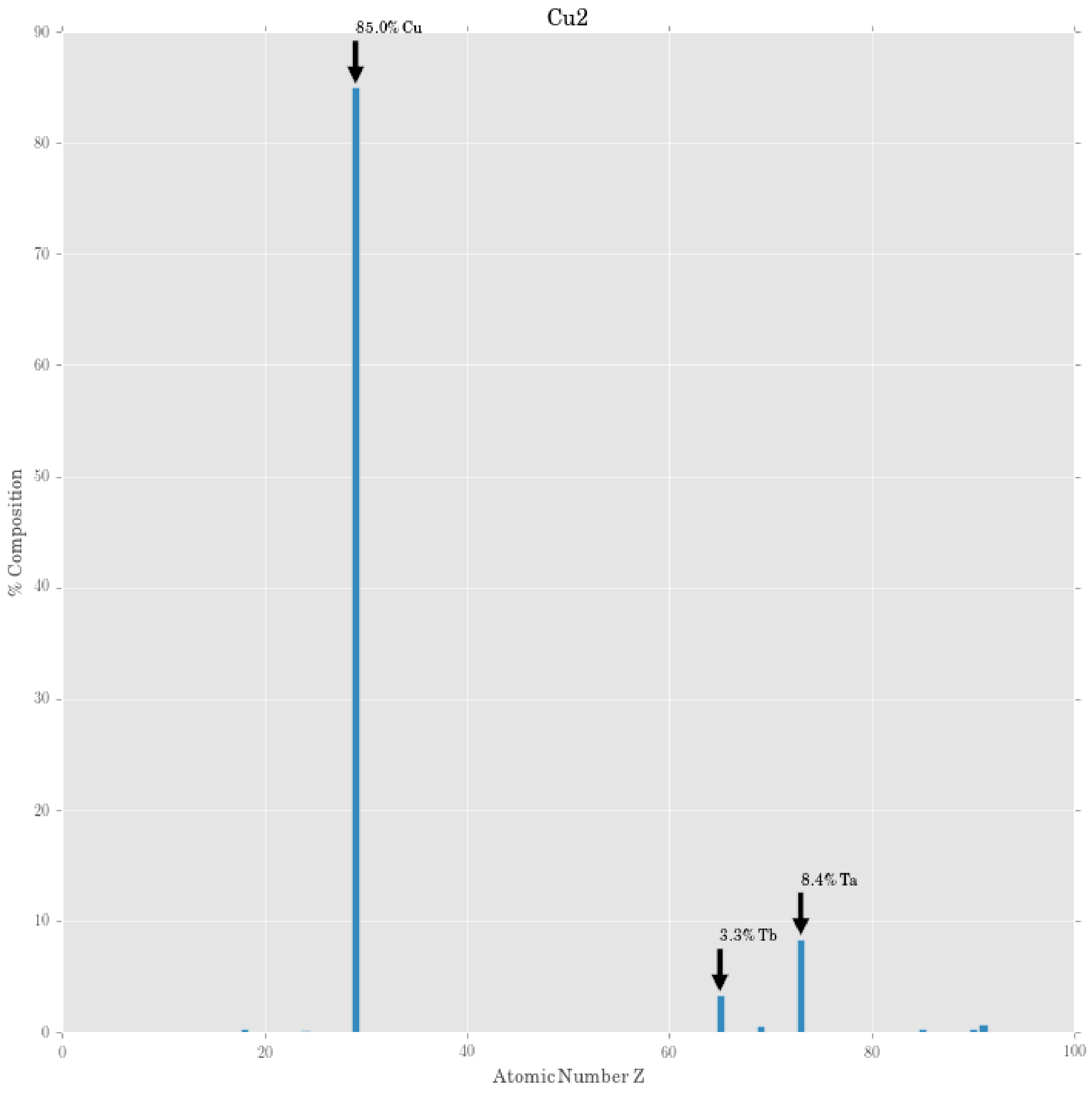
```
81      Tl
82      Pb
83      Bi
84      Po
85      At
86      Rn
87      Fr
88      Ra
89      Ac
90      Th
91      Pa
92       U
93      Np
94      Pu
95      Am

[93 rows x 1 columns]
```

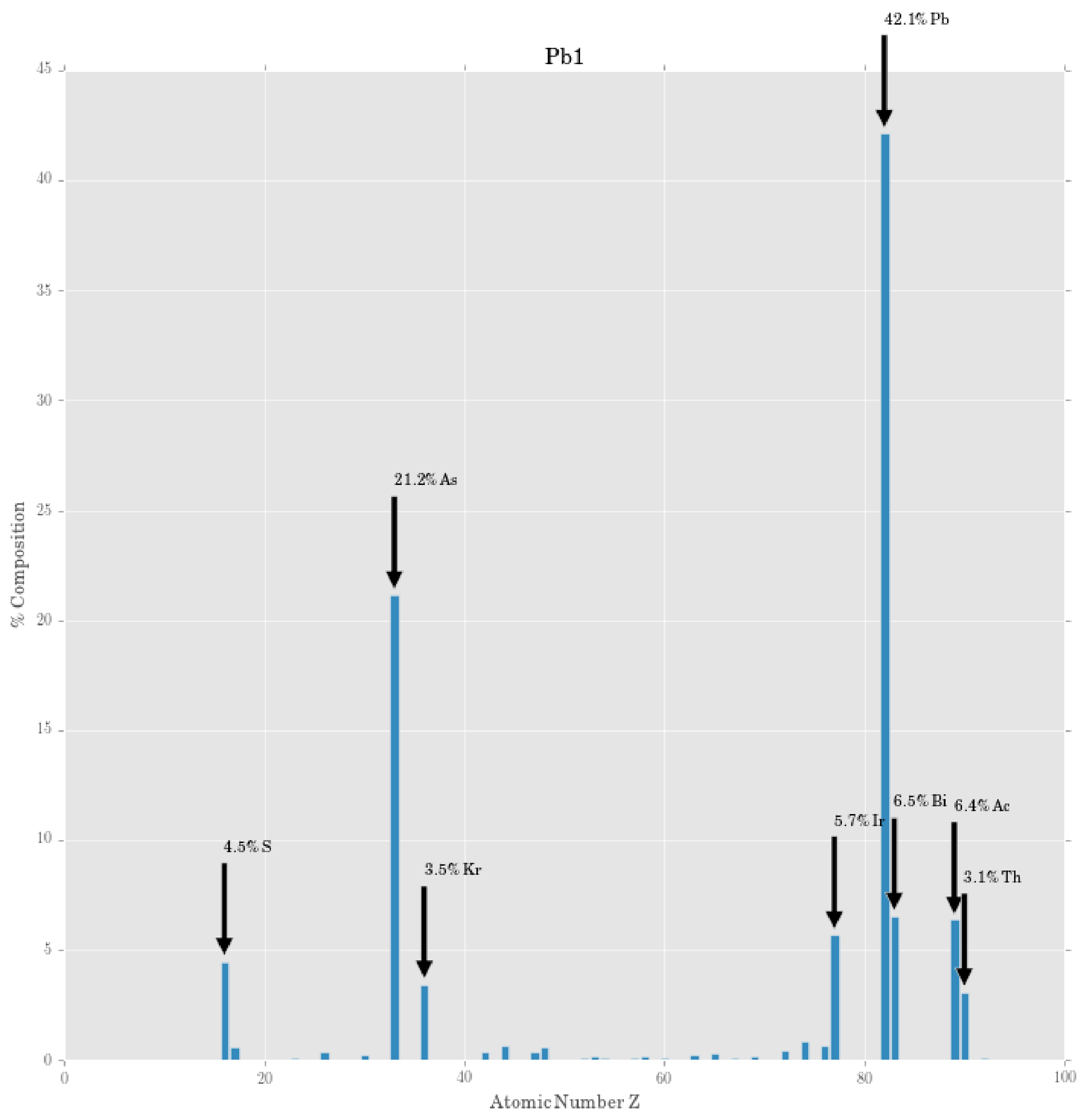
```
In [43]: A = spectral_data[spectral_data.columns[1:]].as_matrix()

         def composition(data):
             comps, resid = optimize.nnls(A,data)
             comps /= np.sum(comps)
             return comps, resid

         def plot_composition(data, title, threshold):
             comps = composition(data)[0]
             important_idx = np.where(comps>threshold)[0]
             fig, ax = plt.subplots(figsize=(10,10))
             ax.bar(np.linspace(3,95, 93), comps*100, align = 'center')
             ax.set_xlabel('Atomic Number Z')
             ax.set_ylabel('\% Composition')
             for i in important_idx:
                 ax.annotate('{:.1f}\% {}'.format(comps[i]*100, Z_dict.iloc[i]['Element']),\
                             xy = (i+3,comps[i]*100), xytext = (i+3,comps[i]*100+5),\
                             arrowprops=dict(facecolor='black', shrink=0.05) )
             ax.set_title(title)
             fig.tight_layout()


         # Lets test this on the known spectrum for various elements as given by the website ref

         plot_composition(spectral_data[8], 'Test Plot', .02)
         # plot_composition(spectrograph('SumCrossXRF/Calibration/7Mar_Cu1.txt').get_counts(),'C
         # plot_composition(spectrograph('SumCrossXRF/Data/7Mar_Gold.txt').get_counts(),'Gold Fo
```

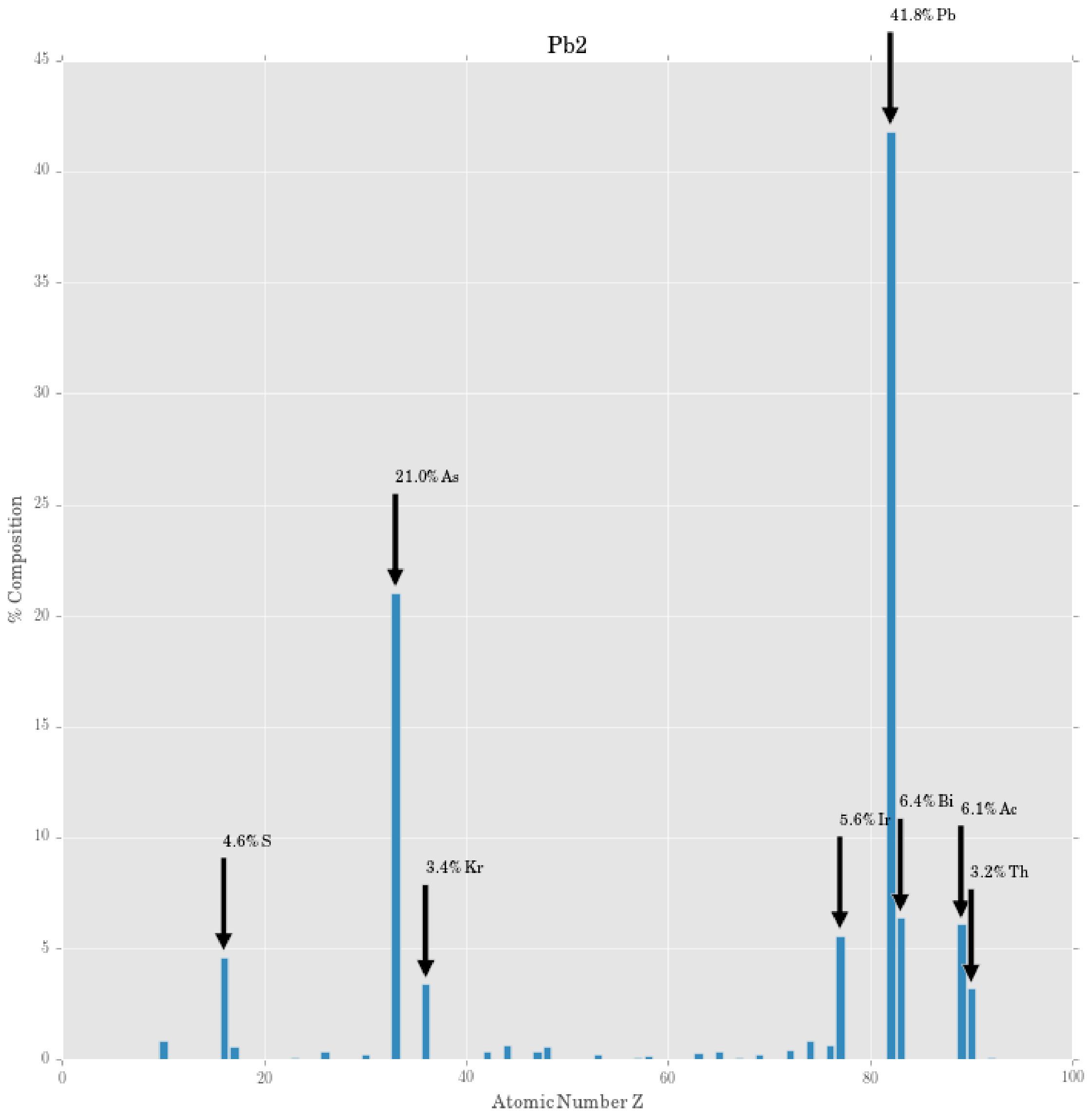


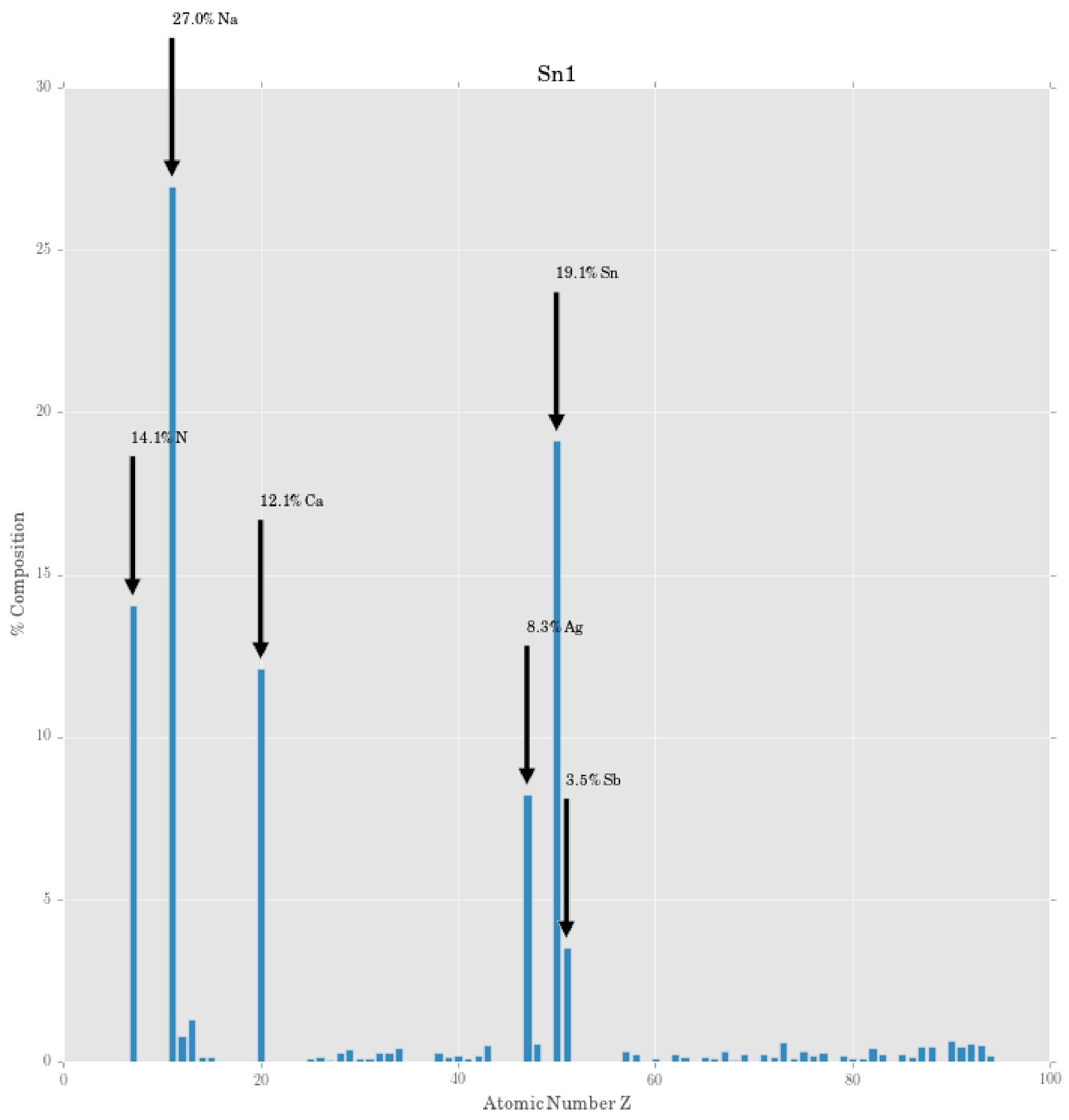
```
In [59]: # Now lets look at the composition of all the data runs
         for file in sorted(os.listdir('SumCrossXRF/Calibration/')):
         #     print(file.partition('.')[0].partition('_')[2])
             spect = spectrograph('SumCrossXRF/Calibration/{}'.format(file))
             plot_composition(spect.get_counts(),file.partition('.')[0].partition('_')[2],.02)
```


Cu1
85.0% Cu
8.5% Ta
3.4% Tb
% Composition
Atomic Number Z
0
10
20
30
40
50
60
70
80
90
100

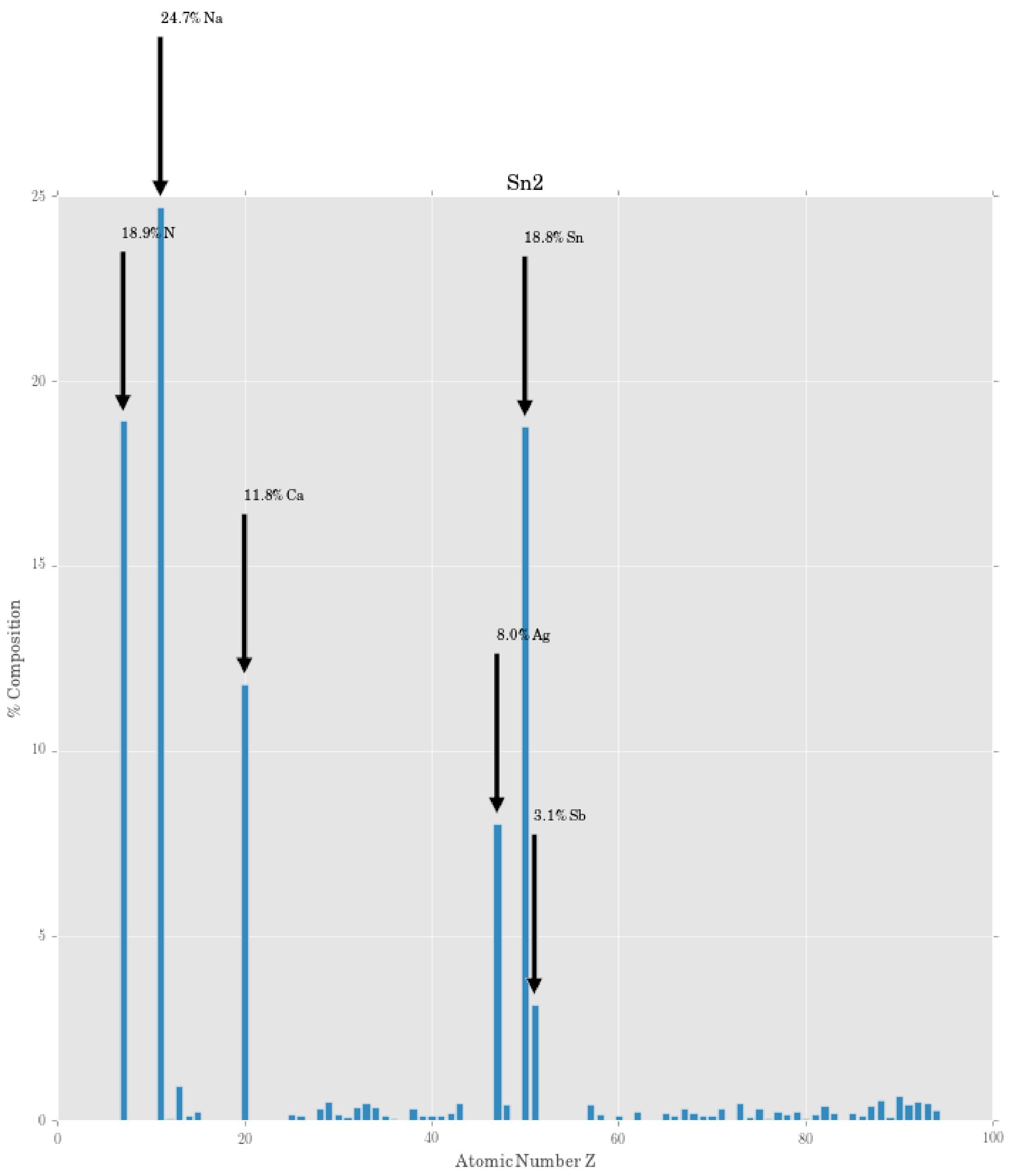

Cu2
85.0% Cu
3.3% Tb
8.4% Ta
% Composition
Atomic Number Z
0
10
20
30
40
50
60
70
80
90
100


Pb1
% Composition
Atomic Number Z
42.1% Pb
21.2% As
4.5% S
3.5% Kr
5.7% Ir
6.5% Bi
6.4% Ac
3.1% Th
0
5
10
15
20
25
30
35
40
45
0
20
40
60
80
100

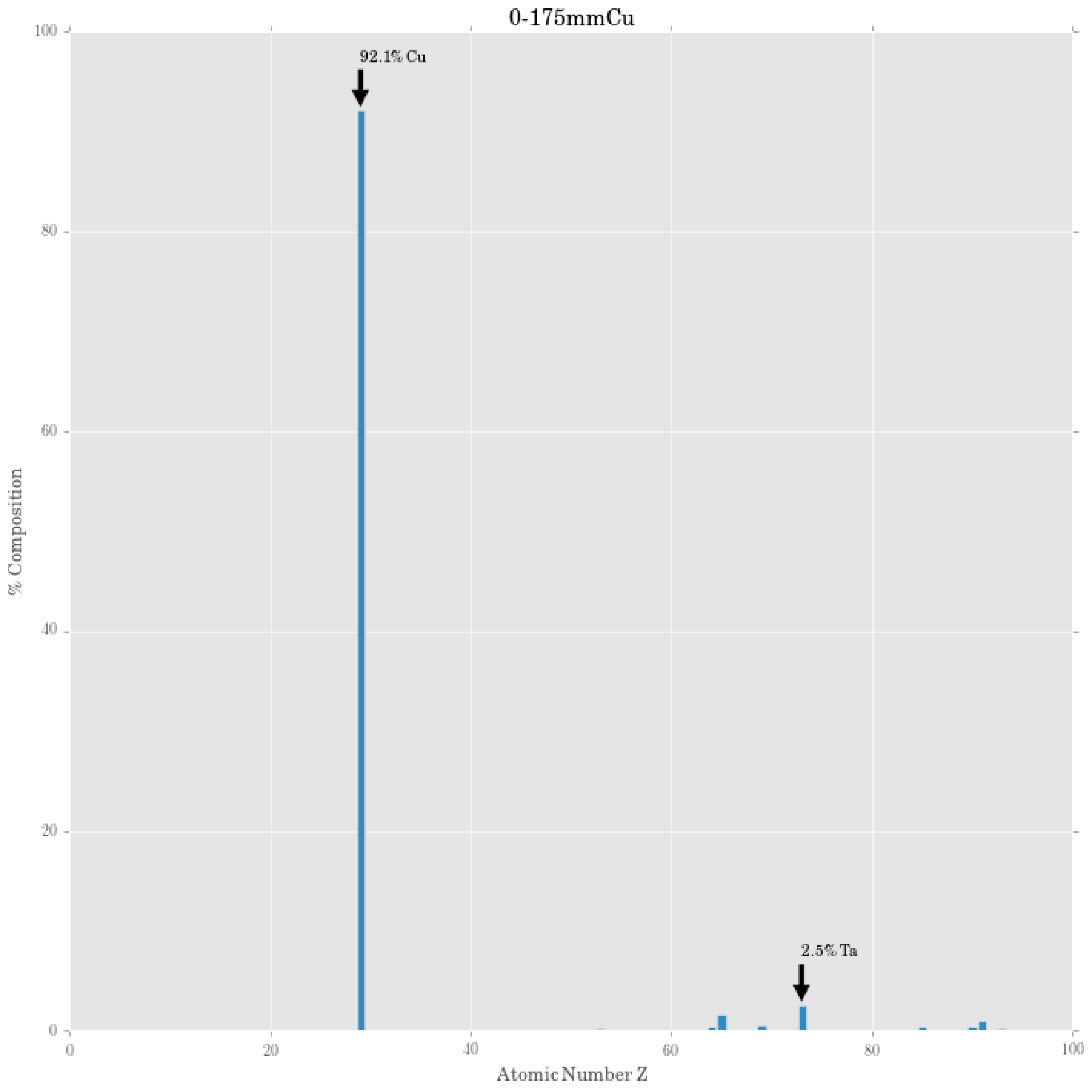

Pb2
41.8% Pb
21.0% As
4.6% S
3.4% Kr
5.6% Ir
6.4% Bi
6.1% Ac
3.2% Th
% Composition
Atomic Number Z
0
5
10
15
20
25
30
35
40
45
0
20
40
60
80
100

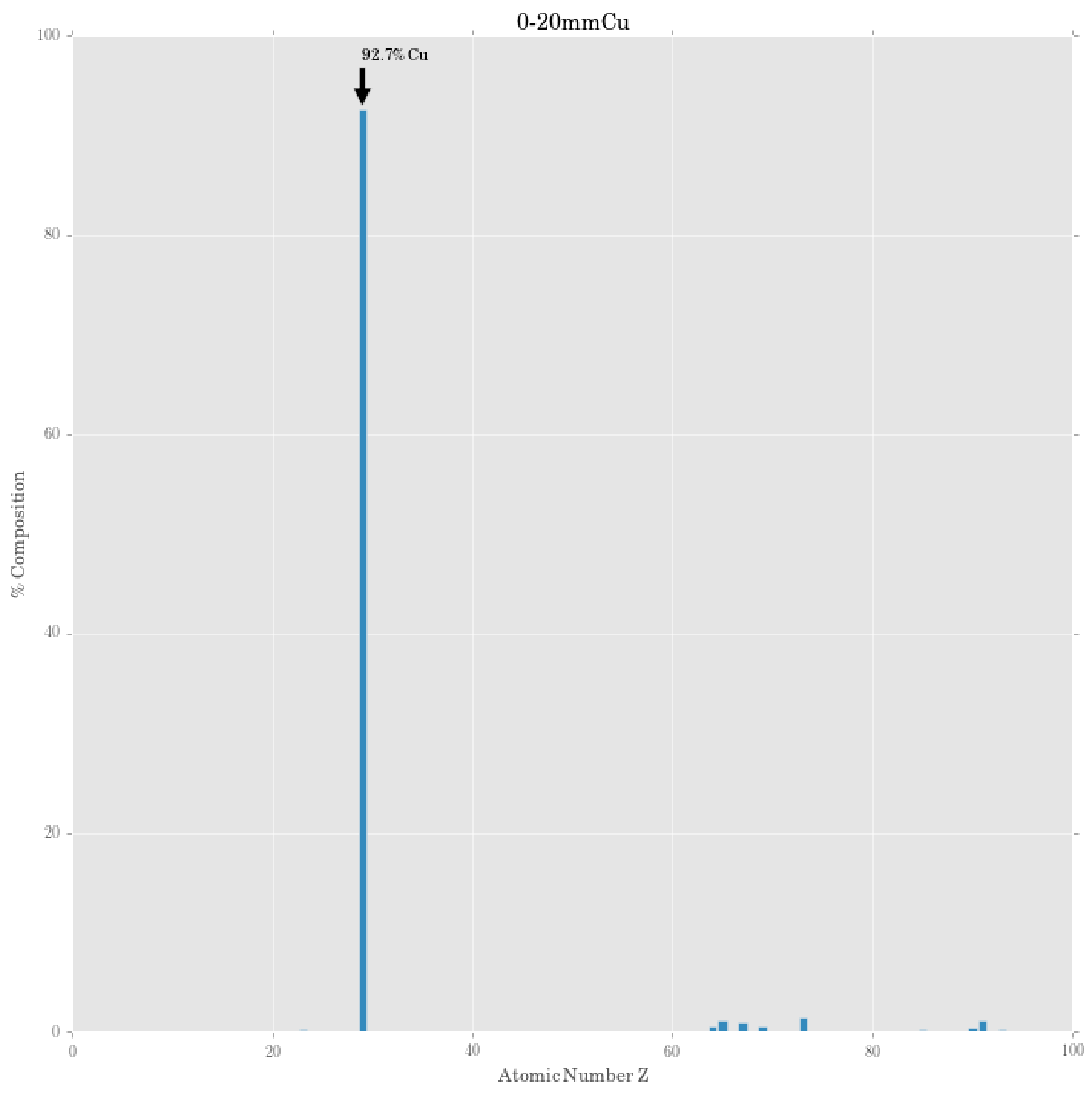


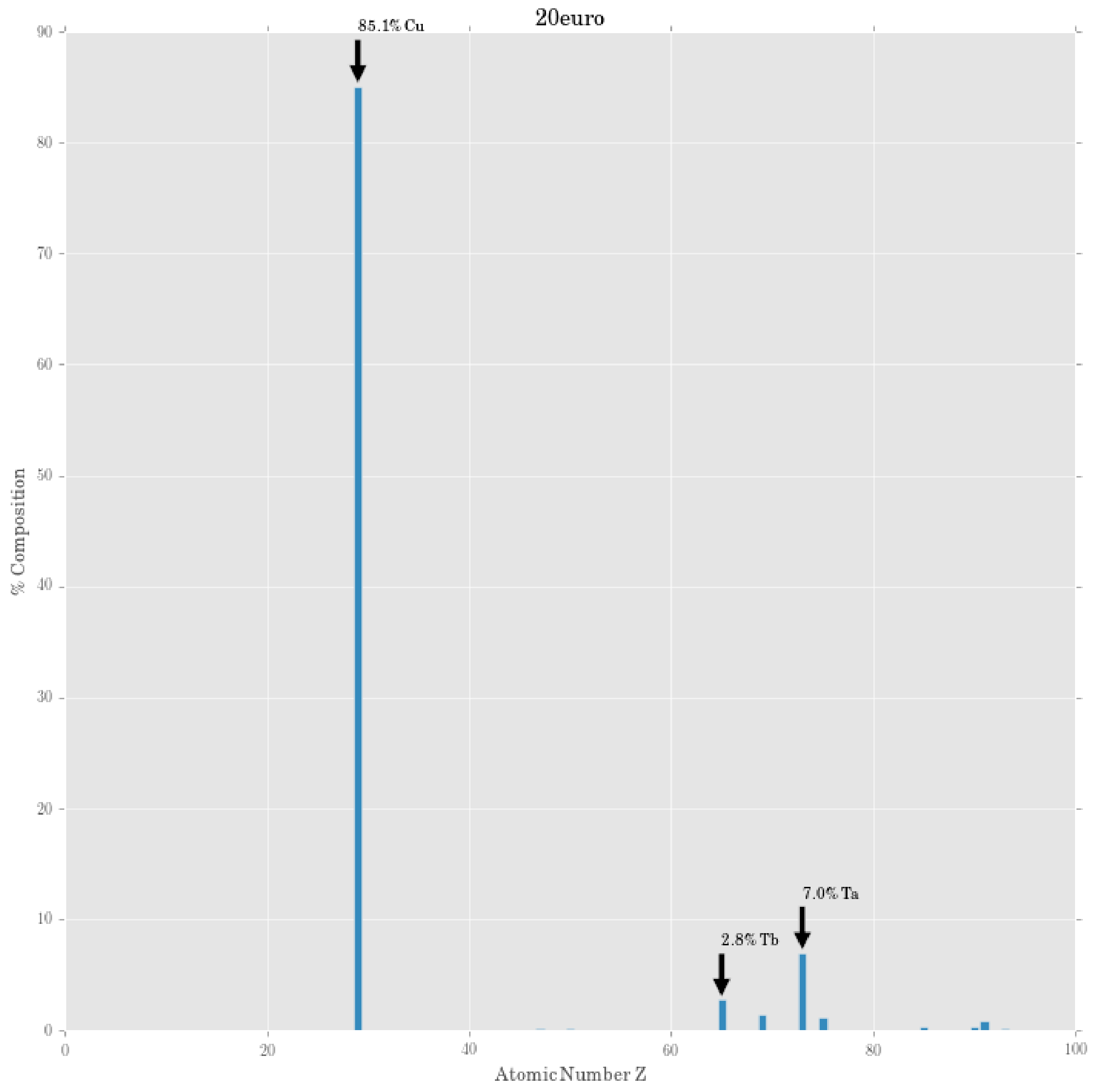
Sn1
27.0% Na
19.1% Sn
14.1% N
12.1% Ca
8.3% Ag
3.5% Sb
% Composition
Atomic Number Z
0
5
10
15
20
25
30
0
20
40
60
80
100



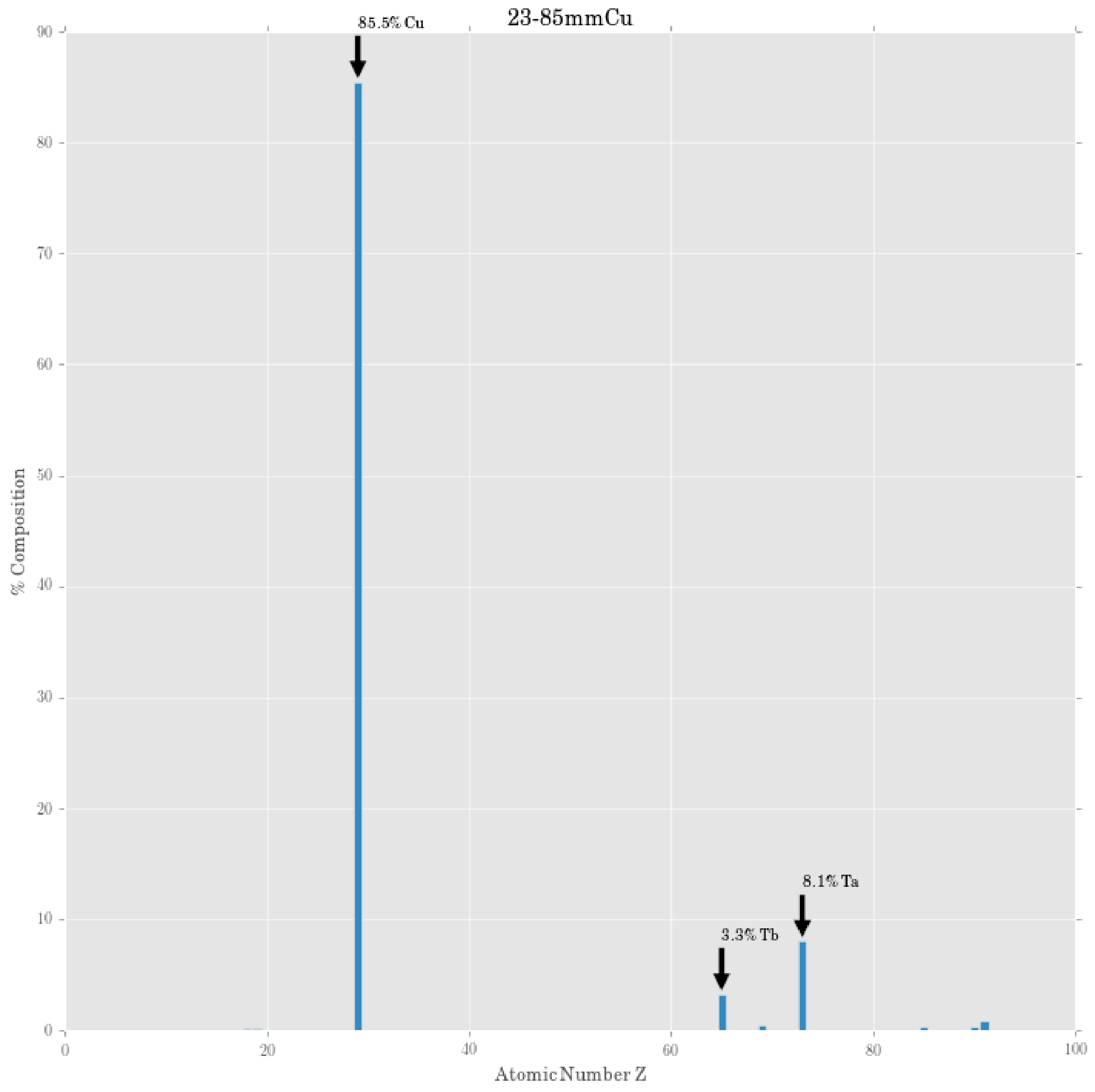
```
In [44]: # Now lets look at the composition of all the data runs
         for file in sorted(os.listdir('SumCrossXRF/Data/')):
         #     print(file.partition('.')[0].partition('_')[2])
             spect = spectrograph('SumCrossXRF/Data/{}'.format(file))
             plot_composition(spect.get_counts(),file.partition('.')[0].partition('_')[2],.02)
```

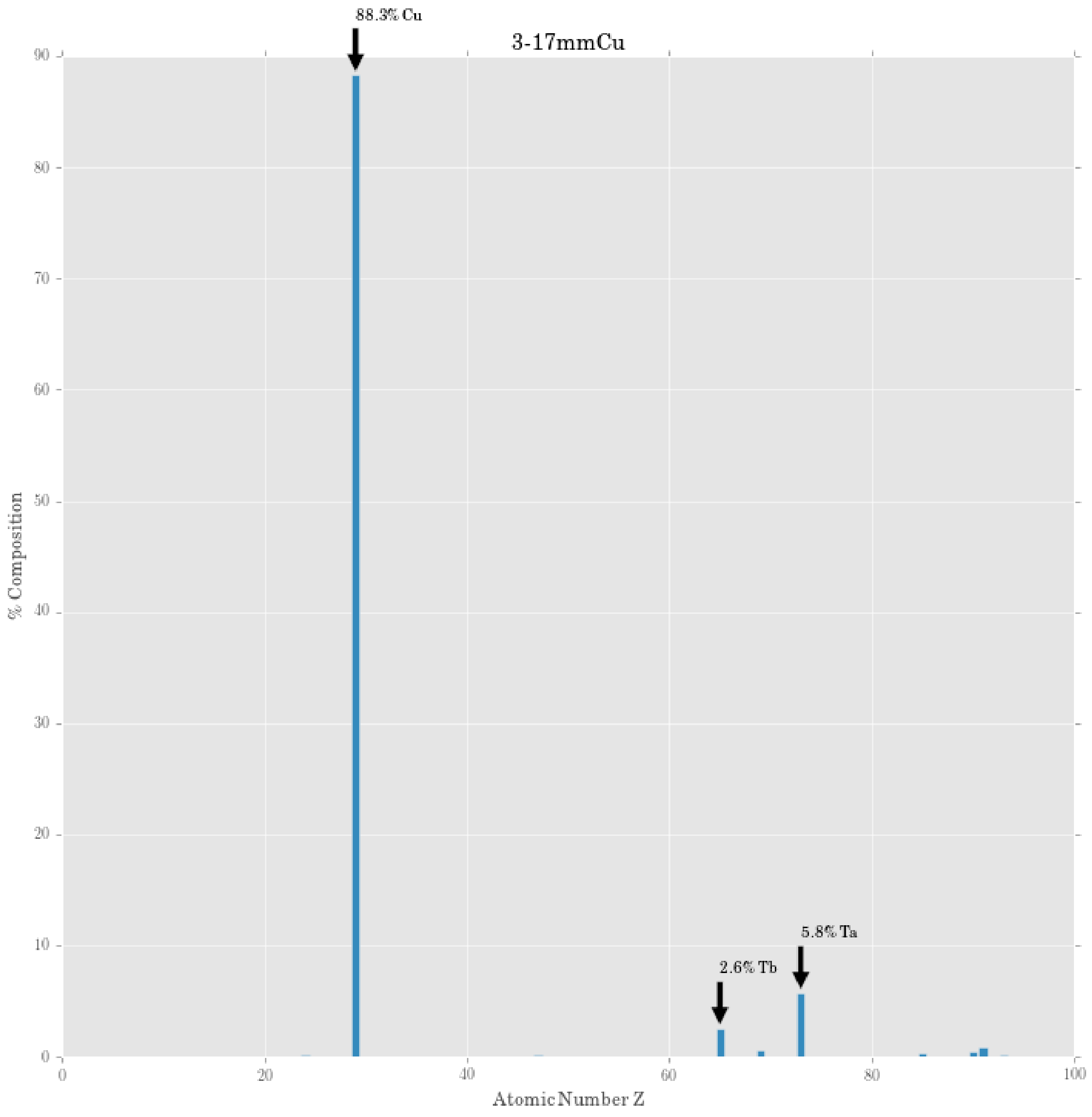

0-175mmCu
92.1% Cu
2.5% Ta
% Composition
Atomic Number Z
0
20
40
60
80
100
0
20
40
60
80
100

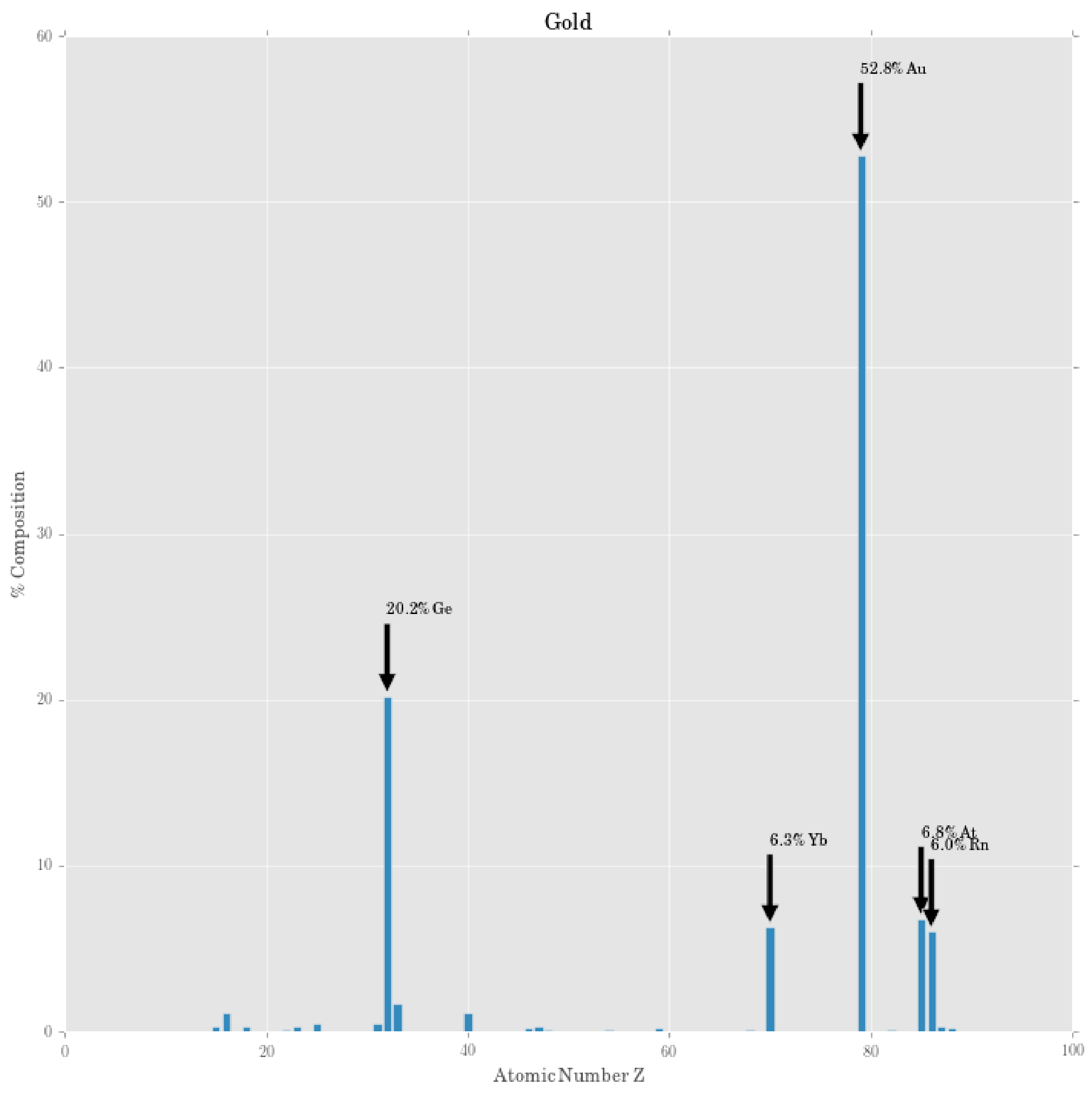

0-20mmCu
92.7% Cu
% Composition
Atomic Number Z
0
20
40
60
80
100



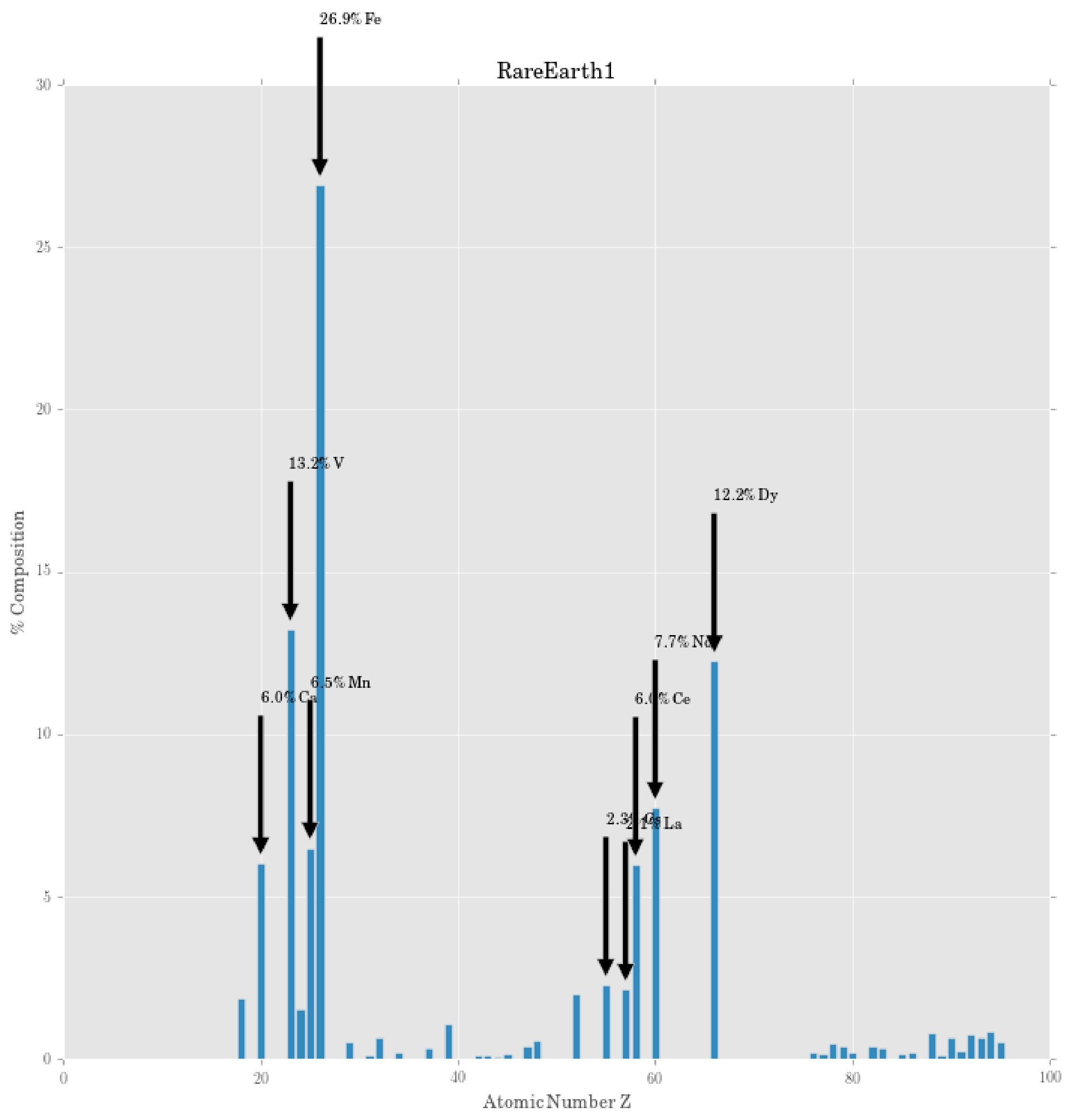
20euro
85.1% Cu
7.0% Ta
2.8% Tb
% Composition
Atomic Number Z
0
10
20
30
40
50
60
70
80
90
0
20
40
60
80
100


23-85mmCu
85.5% Cu
3.3% Tb
8.1% Ta
% Composition
Atomic Number Z
0
10
20
30
40
50
60
70
80
90
100

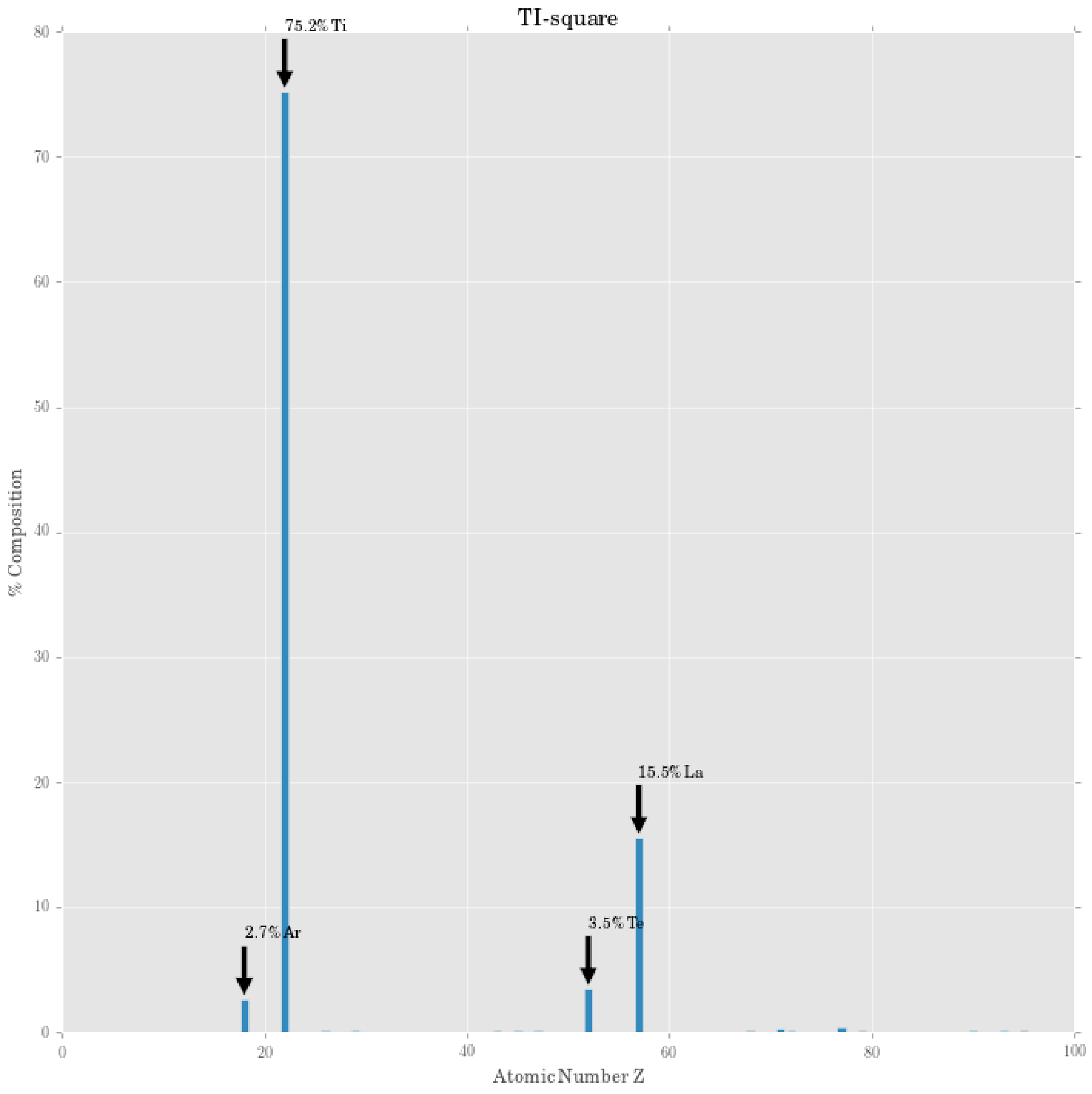

3-17mmCu
88.3% Cu
2.6% Tb
5.8% Ta
% Composition
Atomic Number Z
0
10
20
30
40
50
60
70
80
90
0
20
40
60
80
100

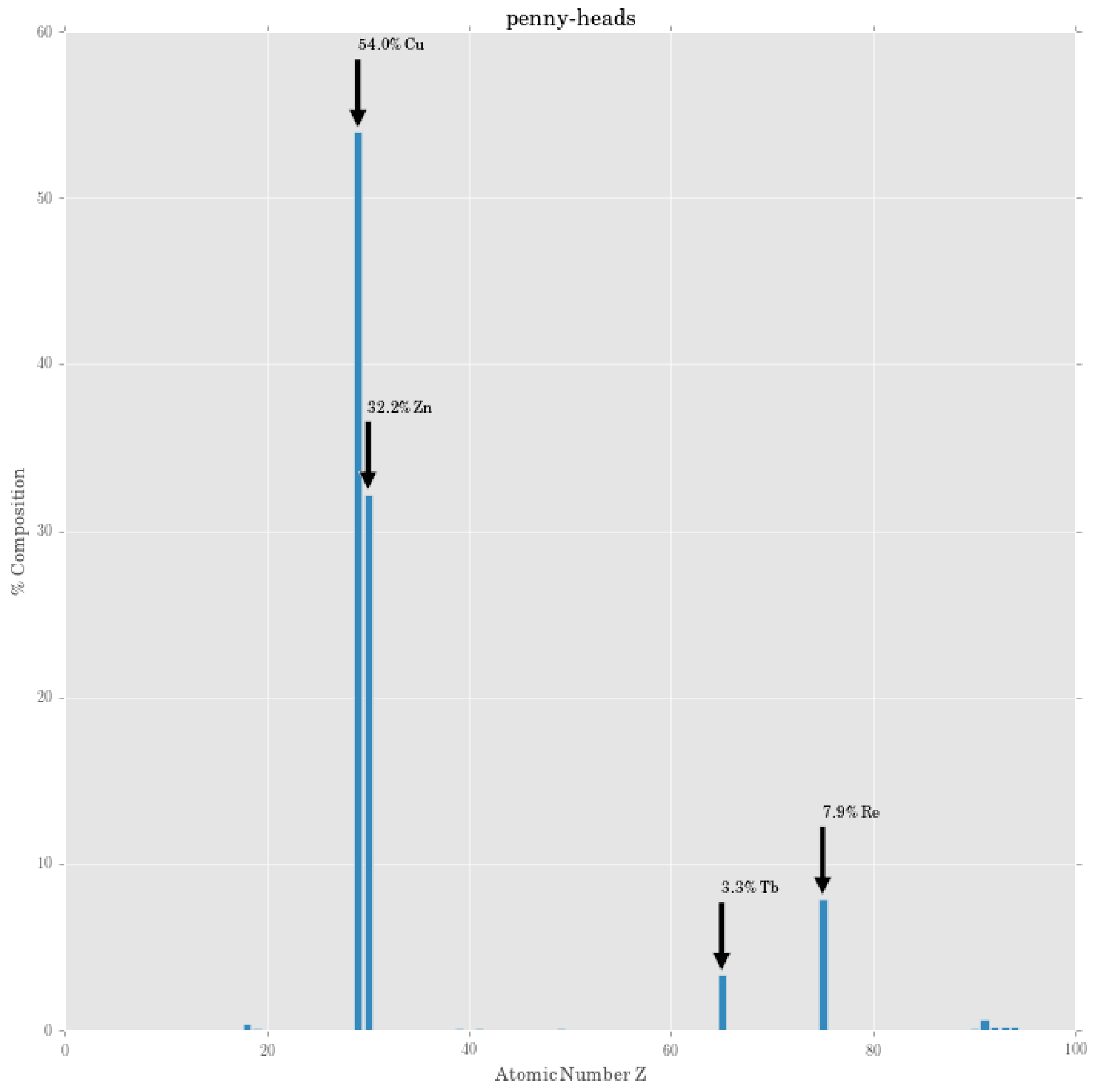


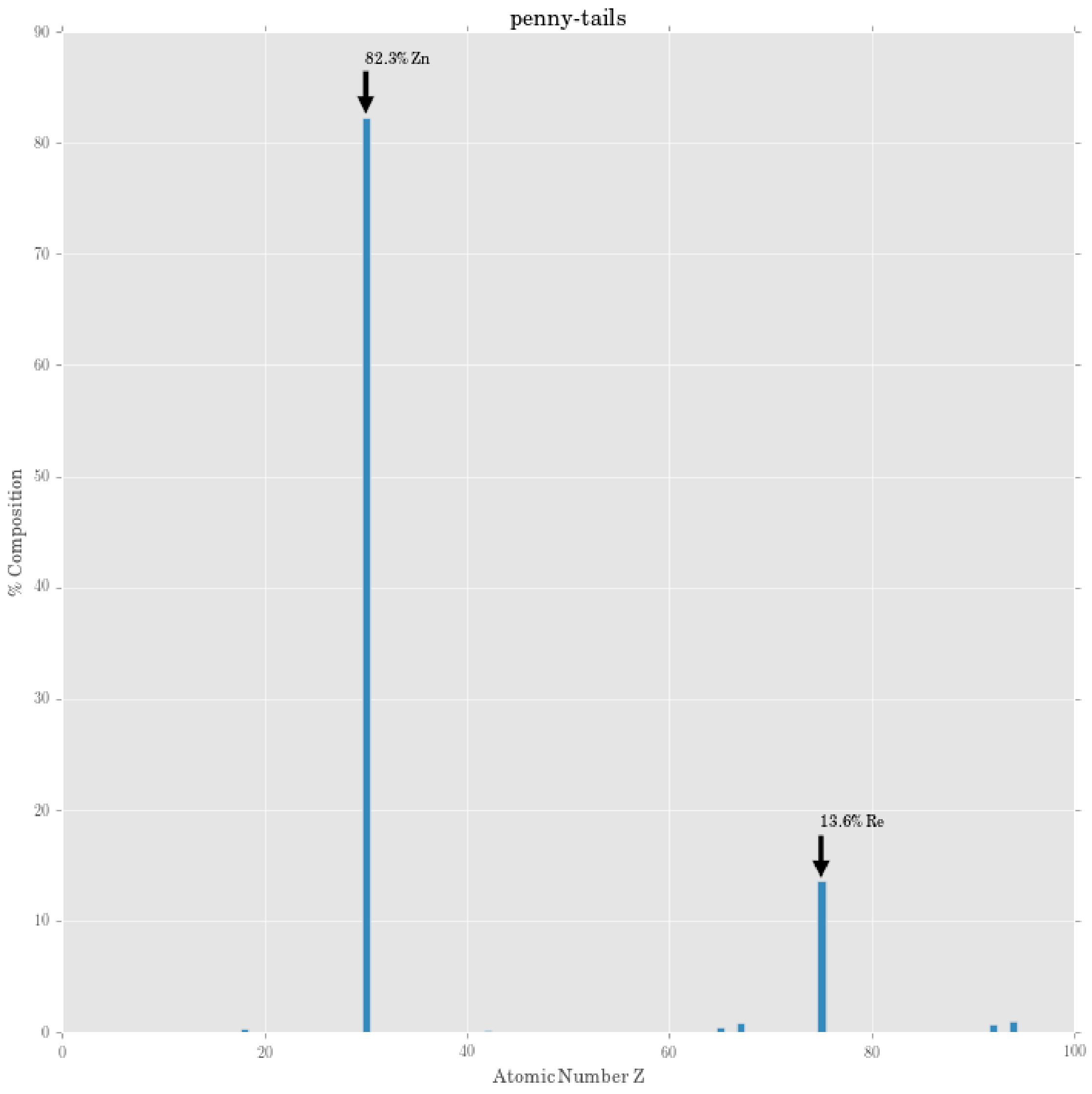
Gold
52.8% Au
20.2% Ge
6.3% Yb
6.8% At
6.0% Rn
% Composition
Atomic Number Z
0
10
20
30
40
50
60
0
20
40
60
80
100

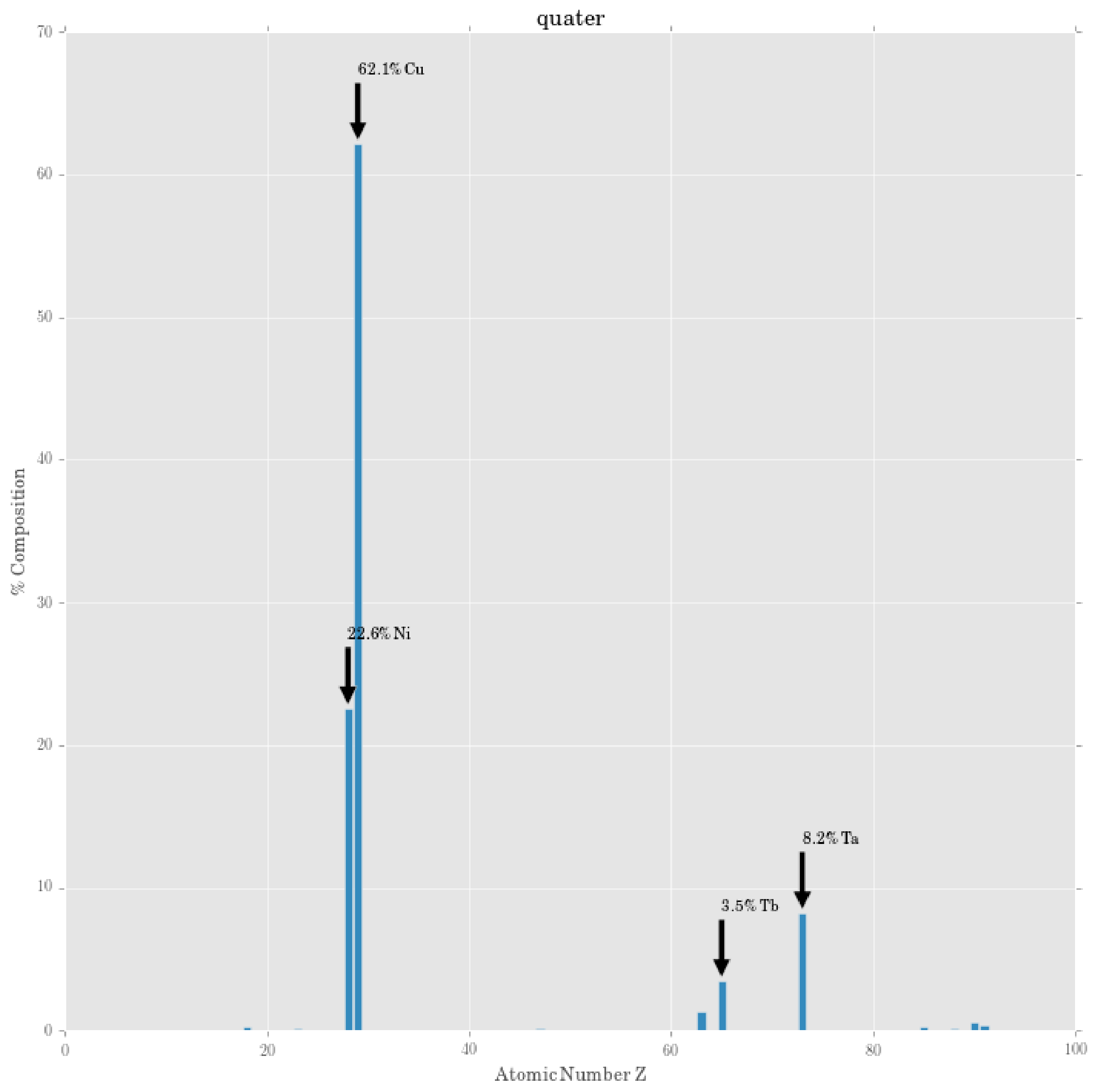


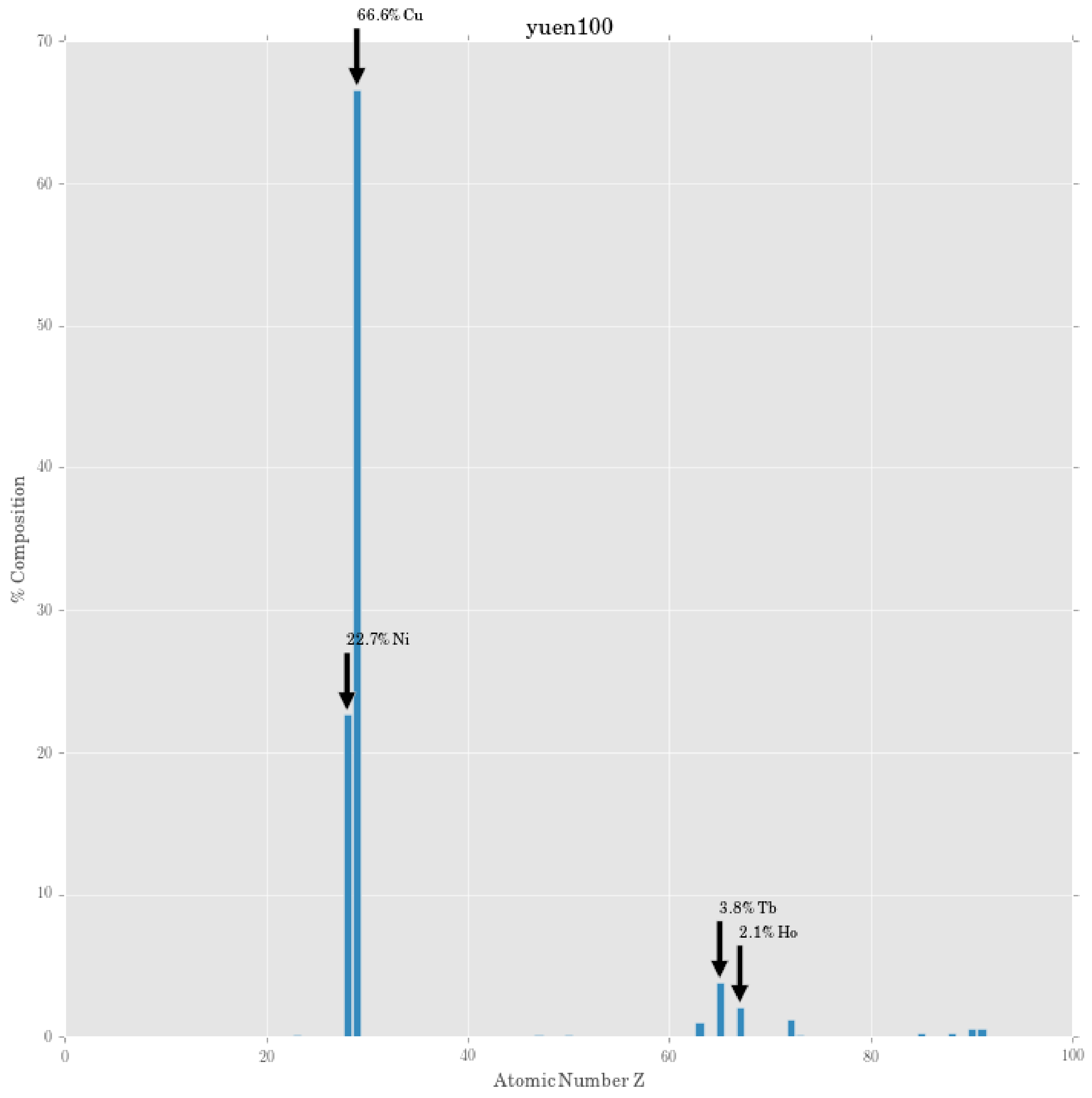
26.9% Fe
RareEarth1
13.2% V
12.2% Dy
7.7% Nd
6.5% Mn
6.0% Ca
6.0% Ce
2.3% Cs
2.1% La
% Composition
Atomic Number Z
0
5
10
15
20
25
30
0
20
40
60
80
100


TI-square
75.2% Ti
15.5% La
3.5% Te
2.7% Ar
% Composition
Atomic Number Z
0
10
20
30
40
50
60
70
80
0
20
40
60
80
100


penny-heads
54.0% Cu
32.2% Zn
3.3% Tb
7.9% Re
% Composition
Atomic Number Z
0
10
20
30
40
50
60
80
100


penny-tails
82.3% Zn
13.6% Re
% Composition
Atomic Number Z
0
10
20
30
40
50
60
70
80
90
0
20
40
60
80
100


quater
62.1% Cu
22.6% Ni
3.5% Tb
8.2% Ta
% Composition
Atomic Number Z
0
10
20
30
40
50
60
70
0
20
40
60
80
100



Clearly, there are some populations in the composition that make me question the fit. For example, in the tin calibration samples, the dominant signal comes from sodium, which is more likely a false artefact from the data, as the sample seemed to be a reasonably well refined material.