ACarbone_CE_Report

March 3, 2021

1 Lab 2: Compton Effect

PHYS3112 - Experimental and Computational Physics

Anthony Carbone - z5260477

Report Due - Wed 3/3/2021

1.1 Aim

The aim of this experiment is to confirm experimental results reflect those predicted by the Compton Scattering process. This was done by observing the scattering of gamma rays from a 18.5MBq Cs-137 source off a thick steel rod.

I also implement an original novel method of removing background noise from the raw gamma ray spectrum.

Compton Scattering:

$$\Delta\lambda = \frac{h}{m_e c}(1-\cos(\theta))$$

and scattered photon energy:

$$E' = \left(\frac{1}{m_e c^2}(1-\cos(\theta)) + \frac{1}{E_0}\right)^{-1}.$$

1.2 Method

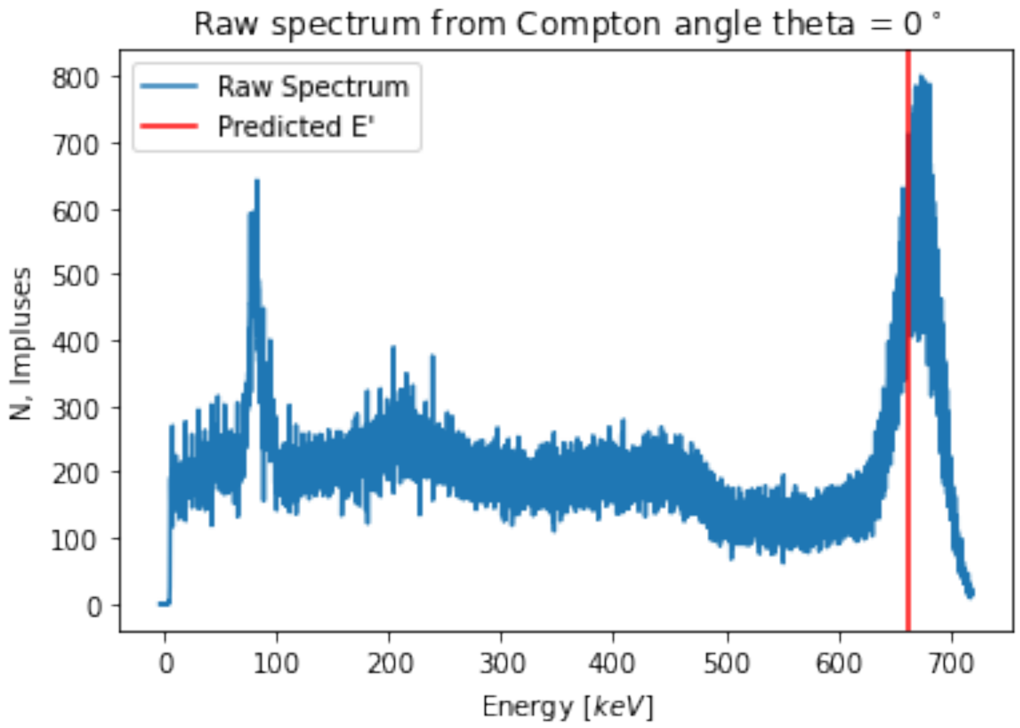
1. Turn on all equipment, ie Multi-Channel Analyser (MCA), photomultiplier, analysis software.
2. Prepare the set-up as in the Operating Instructions, including positioning the Pb blocks appropriately.
3. Place the weak 37kBq Cs-137 coin source inside the opening of the detector, and the 74kBq Na-22 source as close as possible to the detector
4. Gather ~20 minutes of calibration data, and assign the three known energy peaks; 32keV, 511keV, and 662keV.
5. Remove these sources, and set-up for the angle-varying measurements, I used the angles $0°, 30°, 60°, 90°, 120°$ each for ~15 minutes, then exported the spectrum data to my local machine. Note: the $0°$ angle had no rod in the way, this was to ensure no backscattering occured.
6. I gathered a ~15 minute background measurement after removing all radioactive sources and utilised this in my noise reduction.

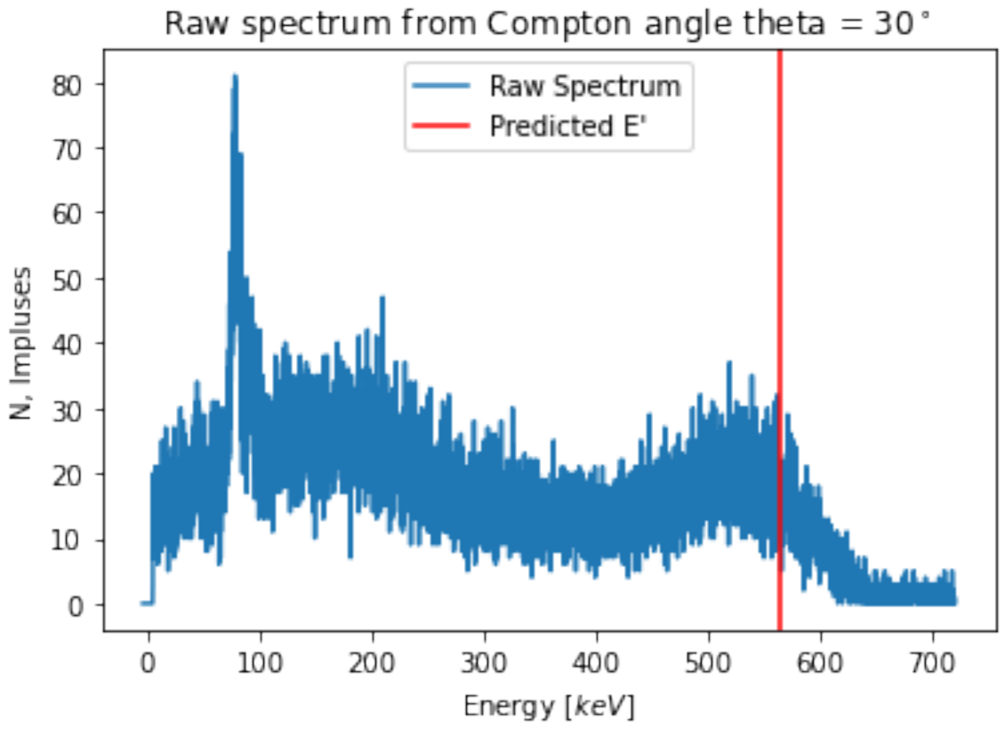
1.3 Preliminary Analysis

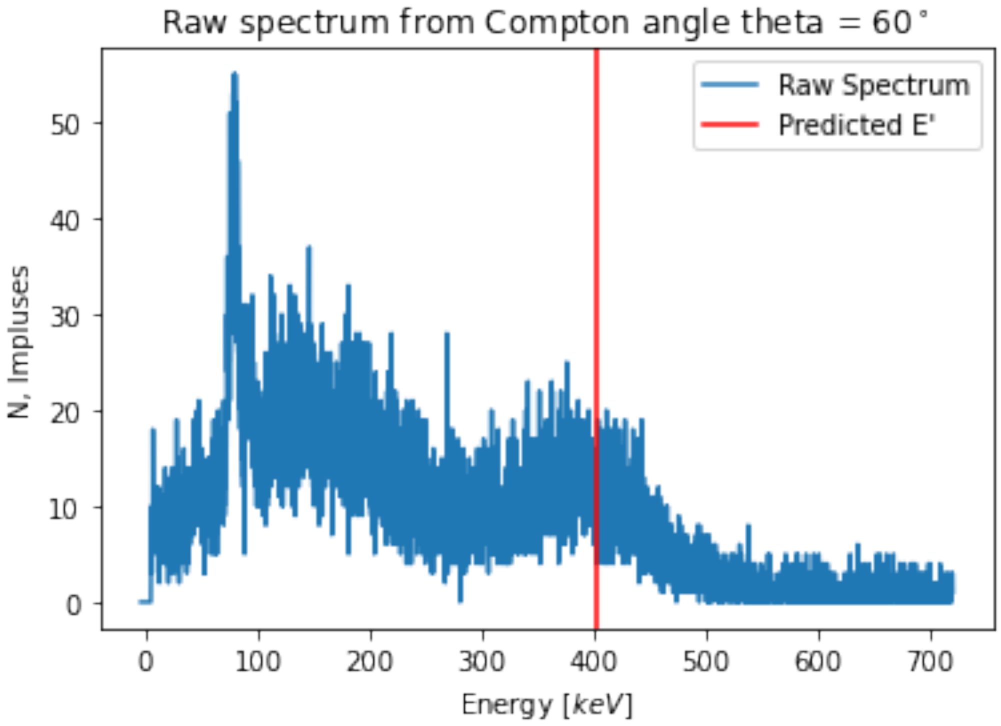
Observing the raw data plots (below), it is clear that for the most part, the Compton scattering is distinct in the first angles but progressively diminishes in strength (as expected) and assembles with the background artifact located around 80keV.

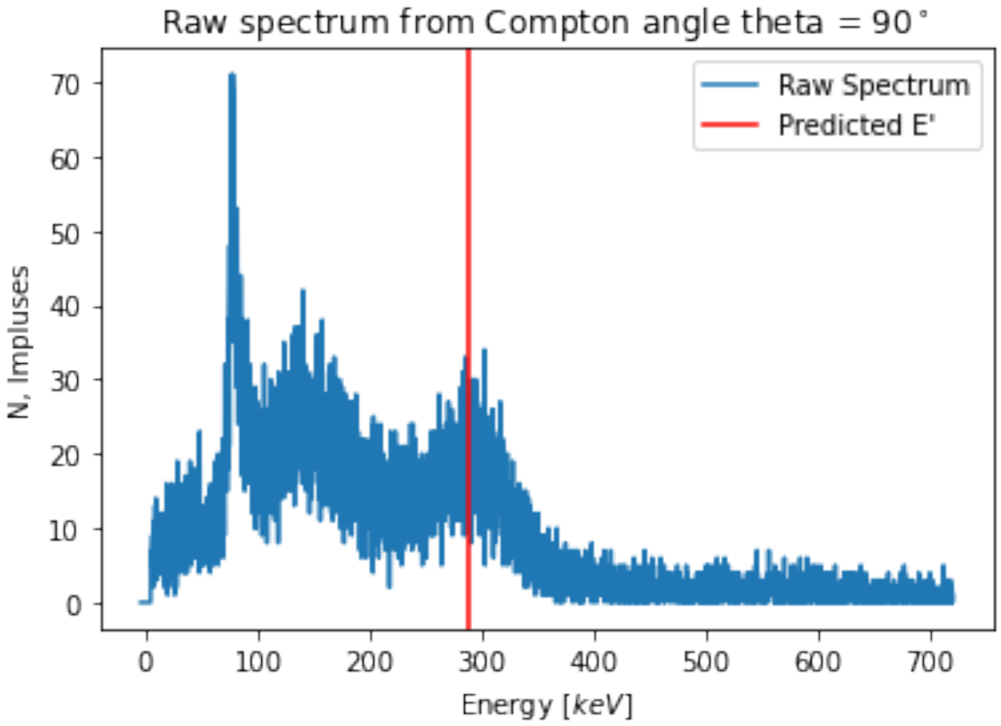
[8]:
```
#%% Load modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.constants import h, m_e, c
from scipy.optimize import curve_fit

#%% Import Data
thetas = np.array([0, 30, 60, 90, 120])
data = {}

for theta in thetas:
    data[theta] = pd.read_csv(f'{theta}deg_15mins.txt', sep='\t', skiprows=[0,␣
 ↪1, 2, 3], names=['E', 'N'])
del theta

#%% Predictions
keV = 6.2415096471204e15 # keV / J
E_0 = 662 / keV
E_pred = {}
for theta in thetas:
    E_pred[theta] = 1 / (1 / (m_e * c**2) * (1 - np.cos(np.radians(theta))) +\
                         1 / E_0) * keV
del c, m_e, keV, h, theta
```

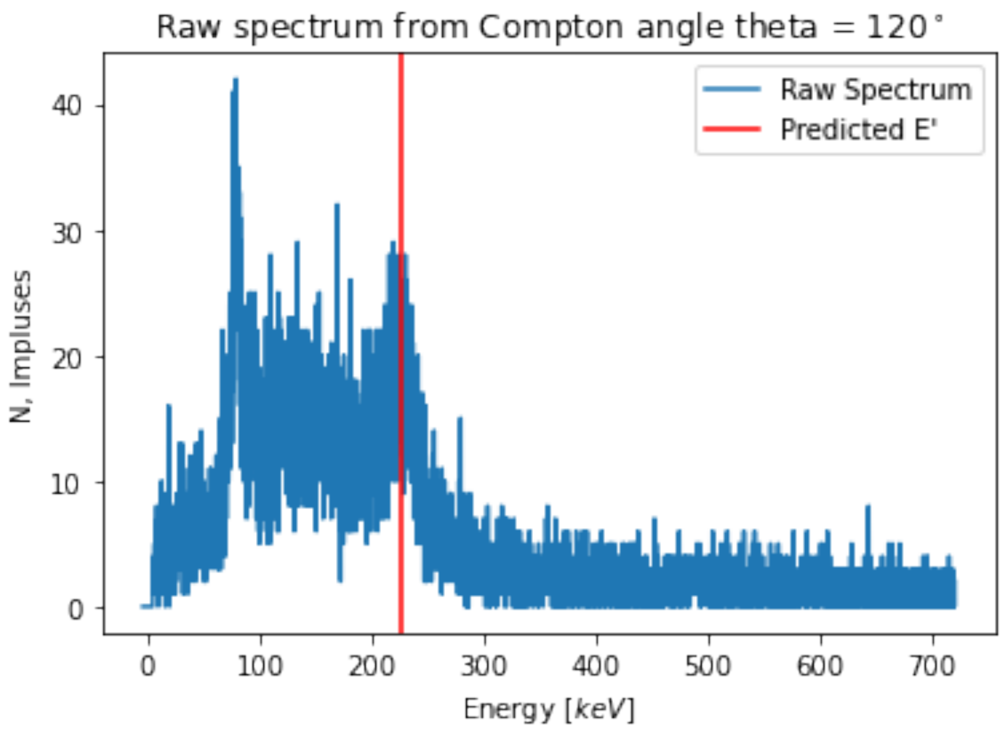
[2]:
```
for theta in thetas:
    plt.figure()
    plt.plot(data[theta]['E'], data[theta]['N'], label="Raw Spectrum")
    plt.xlabel('Energy [$k eV$]')
    plt.ylabel('N, Impluses')
    plt.title(f'Raw spectrum from Compton angle theta = {theta}$^\circ$')
    plt.axvline(x=E_pred[theta], color='r', label="Predicted E'")
    plt.legend()
del theta
```


Raw spectrum from Compton angle theta $= 0^\circ$
Raw Spectrum
Predicted E'
N, Impluses
Energy $[keV]$
0
100
200
300
400
500
600
700
800


Raw spectrum from Compton angle theta $= 30^\circ$
Raw Spectrum
Predicted E'
N, Impluses
Energy $[keV]$
0
10
20
30
40
50
60
70
80
100
200
300
400
500
600
700


Raw spectrum from Compton angle theta = 60°
Raw Spectrum
Predicted E'
N, Impluses
Energy [keV]
0
10
20
30
40
50
0
100
200
300
400
500
600
700

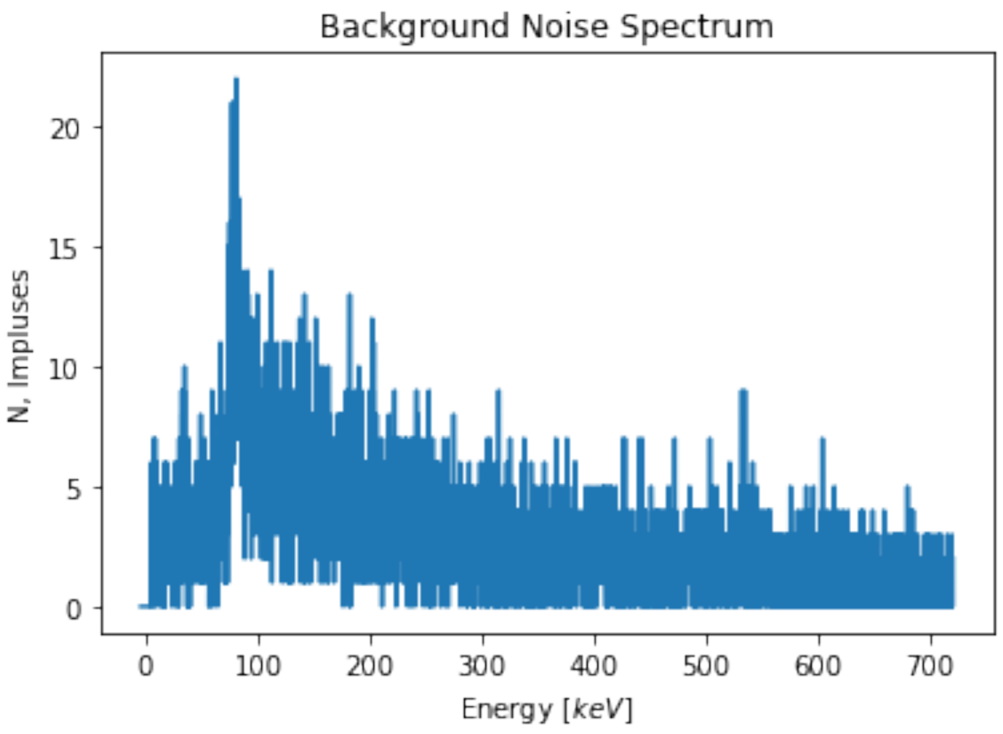

Raw spectrum from Compton angle theta = 90°
Raw Spectrum
Predicted E'
N, Impluses
Energy [keV]
0
10
20
30
40
50
60
70
0
100
200
300
400
500
600
700



1.4 De-noising

Recognising the issue, I investigated novel ways to incorporate the ~15 minute background reading (below) to reduce the background noise in the data in preparation for analysis.

[4]:
```
#%% Import Noise
noise = pd.read_csv('nosource_15mins.txt', sep='\t', skiprows=[0, 1, 2, 3],␣
 ↪names=['E', 'N'])

#%% Plot Noise
plt.plot(noise['E'], noise['N'])
plt.title('Background Noise Spectrum')
plt.xlabel('Energy [$k eV$]')
plt.ylabel('N, Impluses')
plt.show()
```



This background radiation may be from a number of unknown sources, and in this case reside in the low x-ray region.

I tried several methods for eliminating this background radiation; 1. simply subtracting the background from the raw spectrum, 2. scaling the background to match total counts of the raw spectrum, then subtracting it, 3. same as 2. but scaling the background to match the ~80keV peak counts, and 4. same as 3. but performing a 'weighted subtraction' of the background which weights the background heavier for higher count energy levels (so the background counts between 80keV and 220keV) according to

$$w(noise) = weight * e^{\frac{\ln(\frac{noise\_max}{weight})}{max} \cdot noise\_value}$$

which aims to fully eliminate the max peak (as $w(noise\_max) = noise\_max$), and the parameter *weight* may alter the intensity of te denoising attack, like that of a signal compression.

The resultant 'denoised' spectrum is shown below for each of these methods for $\theta = 0°, 60°, 120°$.

Also note that in each method, I set any resulting negative values to zero.

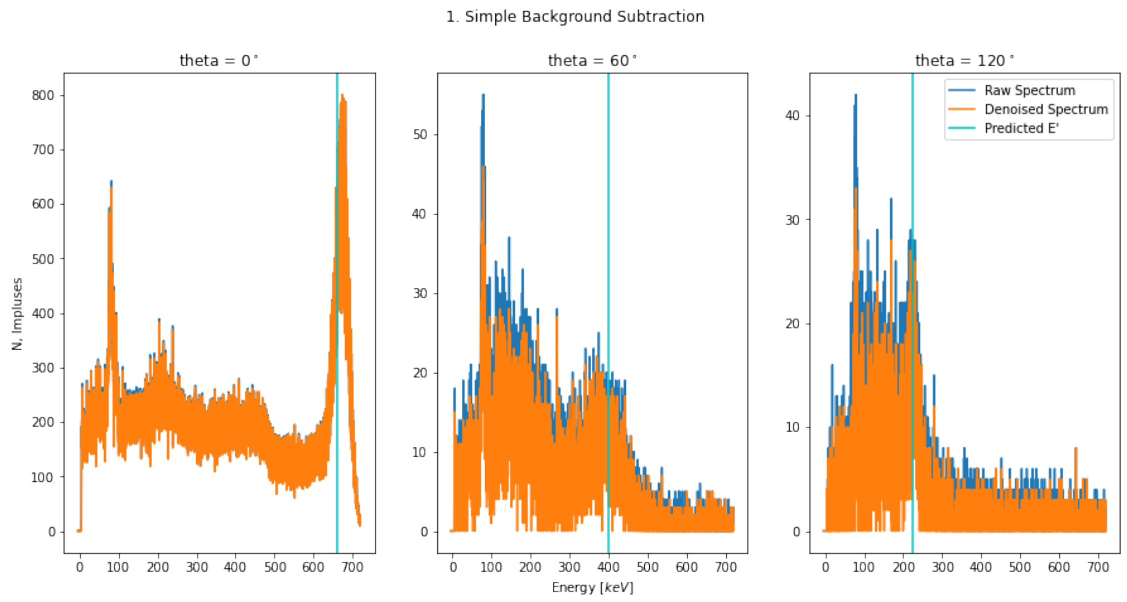
[5]:
```
#%% Function Defs
def denoise(data_series, noise_series, weight):
    denoised_data_series = data_series - noise_weighted(noise_series, weight,␣
 ↪data_series)
    return denoised_data_series

def noise_weighted(noise_series, weight, data_series):
    '''Weights noise series according to my exponential function that weighs␣
↪the larger
    noises exponentially more than the smaller ones.'''

    max_ratio = data_series.max() / noise_series.max() # Max ratio

    max_corrected_noise_series = max_ratio * noise_series # Correct for this␣
↪ratio
    # They now share the same max

    max_value = data_series.max() # What is this shared max?


    max_corrected_noise_series.apply(exponential_weighting, args=(weight,␣
↪max_value))

    return max_corrected_noise_series

def exponential_weighting(noise_value, weight, max_value):
    return weight * np.exp(np.log(max_value / weight) / max_value * noise_value)
```

```
[6]: #%% Denoise Methods 1-4
denoised_Nseries_1 = {}
for theta in [0, 60, 120]:
    denoised_Nseries_1[theta] = data[theta]['N'] - noise['N']
    denoised_Nseries_1[theta][denoised_Nseries_1[theta] < 0] = 0

denoised_Nseries_2 = {}
for theta in [0, 60, 120]:
    count_ratio = data[theta]['N'].sum() / noise['N'].sum()
    denoised_Nseries_2[theta] = data[theta]['N'] - count_ratio * noise['N']
    denoised_Nseries_2[theta][denoised_Nseries_2[theta] < 0] = 0

denoised_Nseries_3 = {}
for theta in [0, 60, 120]:
    max_ratio = data[theta]['N'][:600].max() / noise['N'].max()
    denoised_Nseries_3[theta] = data[theta]['N'] - max_ratio * noise['N']
    denoised_Nseries_3[theta][denoised_Nseries_3[theta] < 0] = 0

denoised_Nseries_4 = {}
weight = 1000000 # How vigourously the noise 'attack' is
for theta in [0, 60, 120]:
    denoised_Nseries_4[theta] = denoise(data[theta]['N'], noise['N'], weight)
    denoised_Nseries_4[theta][denoised_Nseries_4[theta] < 0] = 0
del theta, count_ratio, max_ratio, weight
```

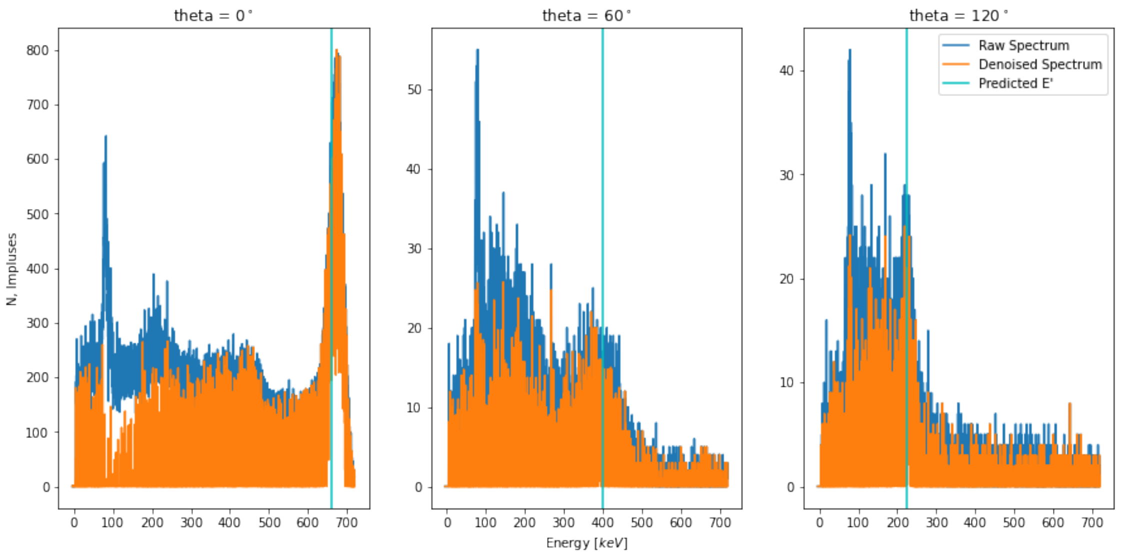
```python
#%% Plot Noise Elimination Methods 1-4
# 1. Simple Background Subtraction
fig, (ax1, ax2, ax3) = plt.subplots(ncols=3, figsize=(15,7))
fig.suptitle('1. Simple Background Subtraction')

ax1.set_title('theta = 0$^\circ$')
ax1.plot(data[0]['E'], data[0]['N'])
ax1.plot(data[0]['E'], denoised_Nseries_1[0])
ax1.set_ylabel('N, Impluses')

ax2.set_title('theta = 60$^\circ$')
ax2.plot(data[60]['E'], data[60]['N'])
ax2.plot(data[60]['E'], denoised_Nseries_1[60])
ax2.set_xlabel('Energy [$k eV$]')

ax3.set_title('theta = 120$^\circ$')
ax3.plot(data[120]['E'], data[120]['N'], label='Raw Spectrum')
ax3.plot(data[120]['E'], denoised_Nseries_1[120], label='Denoised Spectrum')

ax1.axvline(x=E_pred[0], color='c')
ax2.axvline(x=E_pred[60], color='c')
ax3.axvline(x=E_pred[120], color='c', label="Predicted E'")

ax3.legend()

plt.show()

# 2. Count-scaled Background Subtraction
fig, (ax1, ax2, ax3) = plt.subplots(ncols=3, figsize=(15,7))
fig.suptitle('2. Count-scaled Background Subtraction')

ax1.set_title('theta = 0$^\circ$')
ax1.plot(data[0]['E'], data[0]['N'])
ax1.plot(data[0]['E'], denoised_Nseries_2[0])
ax1.set_ylabel('N, Impluses')

ax2.set_title('theta = 60$^\circ$')
ax2.plot(data[60]['E'], data[60]['N'])
ax2.plot(data[60]['E'], denoised_Nseries_2[60])
ax2.set_xlabel('Energy [$k eV$]')

ax3.set_title('theta = 120$^\circ$')
ax3.plot(data[120]['E'], data[120]['N'], label='Raw Spectrum')
ax3.plot(data[120]['E'], denoised_Nseries_2[120], label='Denoised Spectrum')

ax1.axvline(x=E_pred[0], color='c')
```

```python
ax2.axvline(x=E_pred[60], color='c')
ax3.axvline(x=E_pred[120], color='c', label="Predicted E'")

ax3.legend()
plt.show()

# 3. Max-scaled Background Subtraction
fig, (ax1, ax2, ax3) = plt.subplots(ncols=3, figsize=(15,7))
fig.suptitle('3. Max-scaled Background Subtraction')

ax1.set_title('theta = 0$^\circ$')
ax1.plot(data[0]['E'], data[0]['N'])
ax1.plot(data[0]['E'], denoised_Nseries_3[0])
ax1.set_ylabel('N, Impluses')

ax2.set_title('theta = 60$^\circ$')
ax2.plot(data[60]['E'], data[60]['N'])
ax2.plot(data[60]['E'], denoised_Nseries_3[60])
ax2.set_xlabel('Energy [$k eV$]')

ax3.set_title('theta = 120$^\circ$')
ax3.plot(data[120]['E'], data[120]['N'], label='Raw Spectrum')
ax3.plot(data[120]['E'], denoised_Nseries_3[120], label='Denoised Spectrum')

ax1.axvline(x=E_pred[0], color='c')
ax2.axvline(x=E_pred[60], color='c')
ax3.axvline(x=E_pred[120], color='c', label="Predicted E'")

ax3.legend()
plt.show()

# 4. Weighted Background Subtraction
fig, (ax1, ax2, ax3) = plt.subplots(ncols=3, figsize=(15,7))
fig.suptitle('4. Weighted Background Subtraction')

ax1.set_title('theta = 0$^\circ$')
ax1.plot(data[0]['E'], data[0]['N'])
ax1.plot(noise['E'], denoised_Nseries_4[0])
ax1.set_ylabel('N, Impluses')

ax2.set_title('theta = 60$^\circ$')
ax2.plot(data[60]['E'], data[60]['N'])
ax2.plot(noise['E'], denoised_Nseries_4[60])
ax2.set_xlabel('Energy [$k eV$]')

ax3.set_title('theta = 120$^\circ$')
ax3.plot(data[120]['E'], data[120]['N'], label='Raw Spectrum')
```

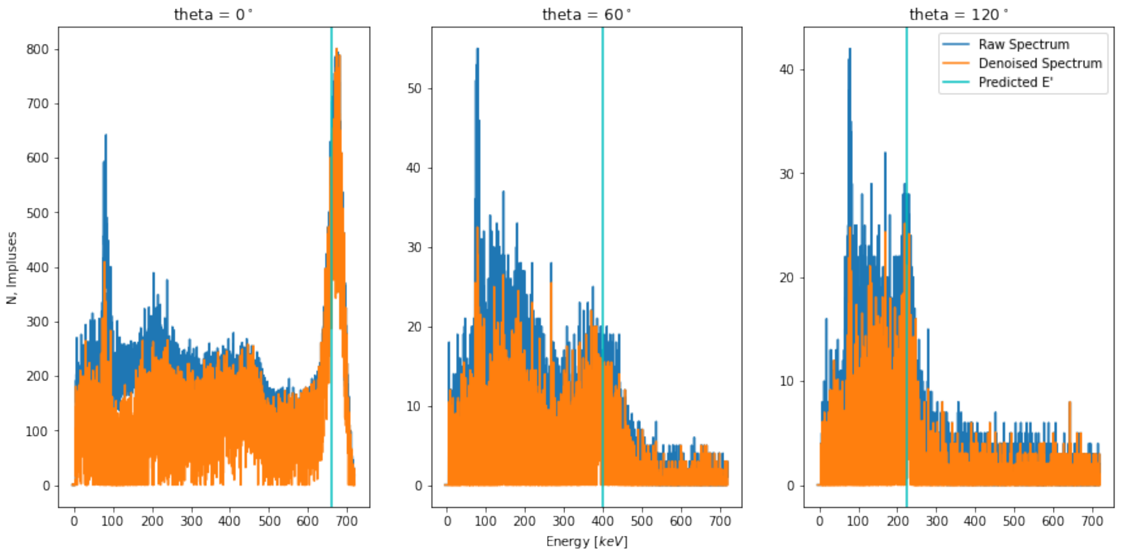
```
ax3.plot(noise['E'], denoised_Nseries_4[120], label='Denoised Spectrum')

ax1.axvline(x=E_pred[0], color='c')
ax2.axvline(x=E_pred[60], color='c')
ax3.axvline(x=E_pred[120], color='c', label="Predicted E'")

ax3.legend()

plt.show()

del fig, ax1, ax2, ax3
del denoised_Nseries_1, denoised_Nseries_2, denoised_Nseries_3,␣
 ↪denoised_Nseries_4
```

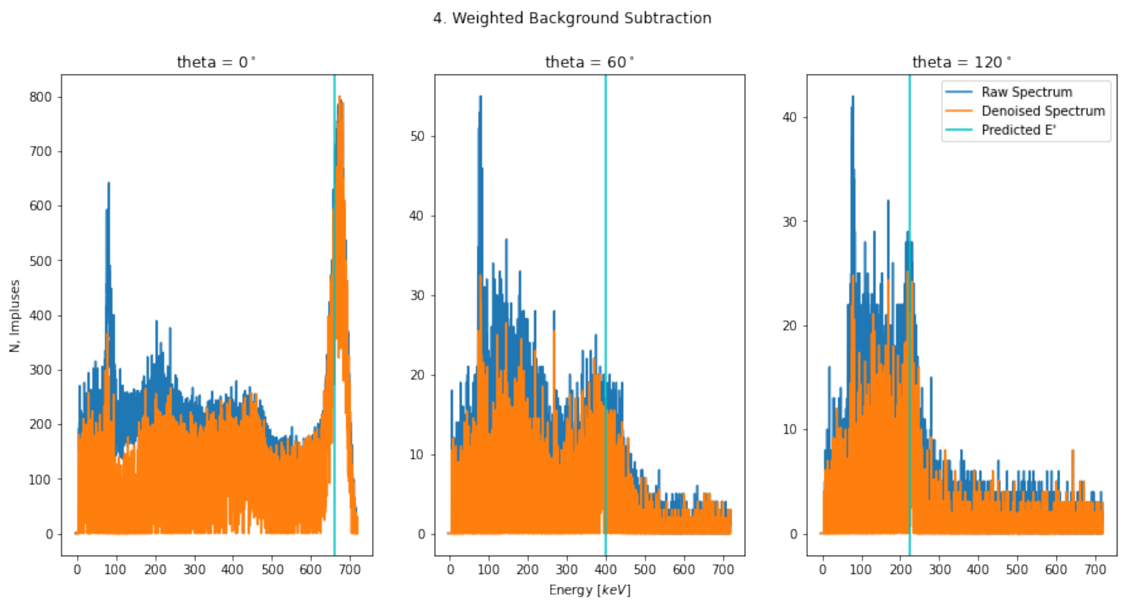


2. Count-scaled Background Subtraction



3. Max-scaled Background Subtraction





From the above plots, it can be concluded that; 1. The simple subtraction method barely reduces the noise, this is because the background counts are so few compared to the data counts themselves. Another method must be used. 2. The count-scaled method seems to eliminate the x-ray background effectively, and impacts the expected peak little. This method may be used. 3. The max-scaled method achieves similar results to method 2 and seemingly impacts the expected peak even less. This method may be preferred to method 2. 4. While the weighted-subtraction method impacts the expected peak least (as intended), it fails to effectively elimnate the x-ray background peak. Though novel, this more complex denoising process fails to out perform the simpler method 3, even from investigation into different *weight* parameters, but other functions than that suggested above may be considered to be useful.

Therefore, I shall use method 3, the max-scaled method for denoising the data for analysis.

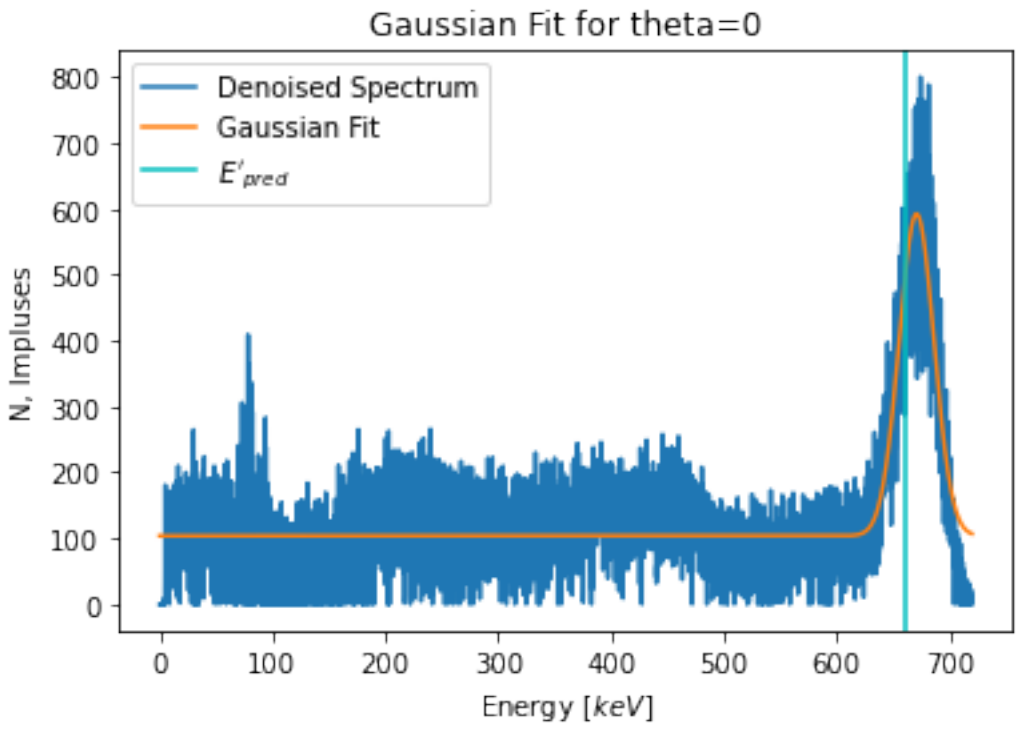
```
[10]: denoised_data = {}
      for theta in thetas:
          denoised_data[theta] = data[theta].copy() # Make copy of DF
          max_ratio = data[theta]['N'][:600].max() / noise['N'].max() # Scaling␣
       ↪factor
          denoised_data[theta]['N'] = data[theta]['N'] - max_ratio * noise['N'] #␣
       ↪Subtract scaled noise
          denoised_data[theta][denoised_data[theta] < 0] = 0 # Set -ve values to 0
      del theta, max_ratio
```

1.5 Analysis

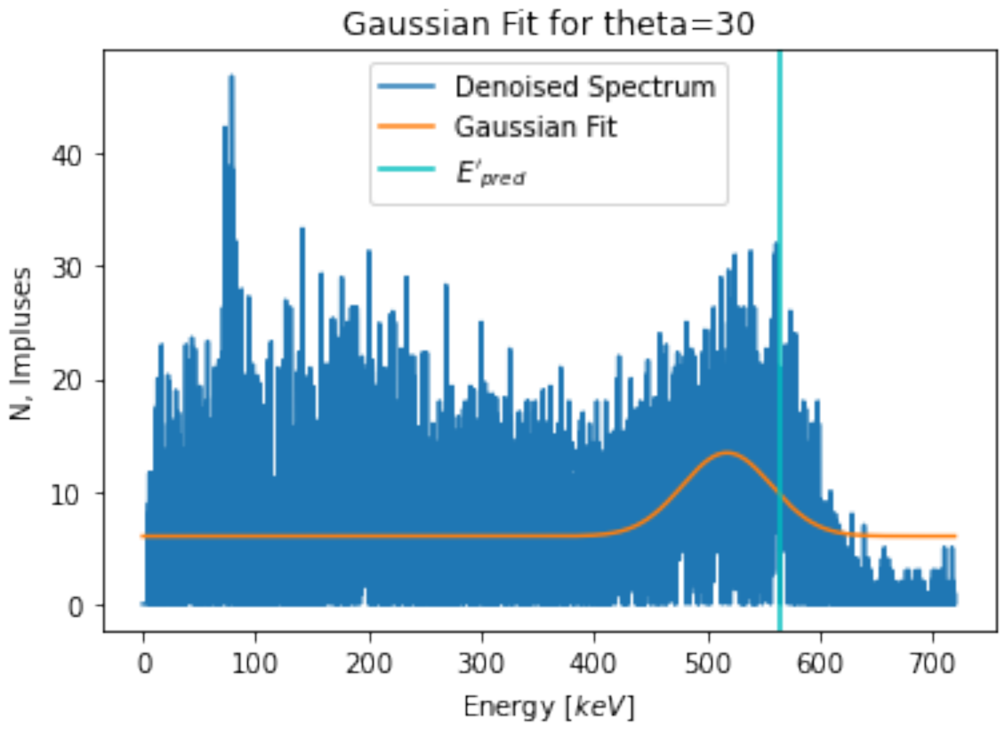
The broadened Compton peak is assumed to be Gaussian, so I fitted the de-noised data with a Gaussian function. See the fits and resultant parameters below.

[25]:
```
# Function to fit
def gauss(x, a, x_0, sigma, h):
    return h + a * np.exp(-(x - x_0) ** 2 / (2 * sigma ** 2))

# Convenient function to fit
def gauss_fit(theta):
    mean = E_pred[theta]
    sigma = 50
    x = denoised_data[theta]['E']
    y = denoised_data[theta]['N']
    popt, pcov = curve_fit(gauss, x, y, p0=[np.max(y), mean, sigma, 0])
    return popt

for theta in thetas:
    a, x_0, sigma, h = gauss_fit(theta)
    print(f'For theta={theta}: E_pred={E_pred[theta]:.2f}, E={x_0:.2f},␣
↪sigma={sigma:.2f}, a={a:.2f}, and h={h:.2f}')

    plt.figure()
    plt.title(f'Gaussian Fit for theta={theta}')
    plt.plot(denoised_data[theta]['E'], denoised_data[theta]['N'],␣
↪label='Denoised Spectrum')
    plt.plot(denoised_data[theta]['E'], gauss(denoised_data[theta]['E'], a,␣
↪x_0, sigma, h), label=f'Gaussian Fit')
    plt.xlabel('Energy [$k eV$]')
    plt.ylabel('N, Impluses')
    plt.axvline(x=E_pred[theta], label="$E'_{pred}$", color='c')
    plt.legend()
    plt.show()

del theta, a, x_0, sigma
```

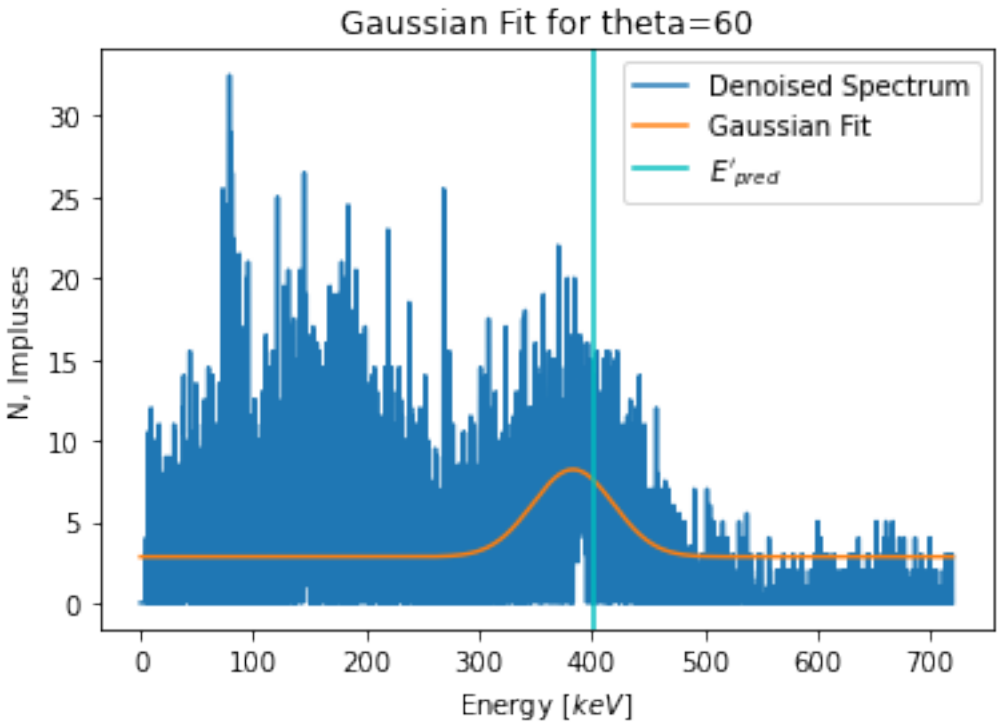
```
For theta=0: E_pred=662.00, E=670.85, sigma=15.45, a=488.31, and h=103.79
```



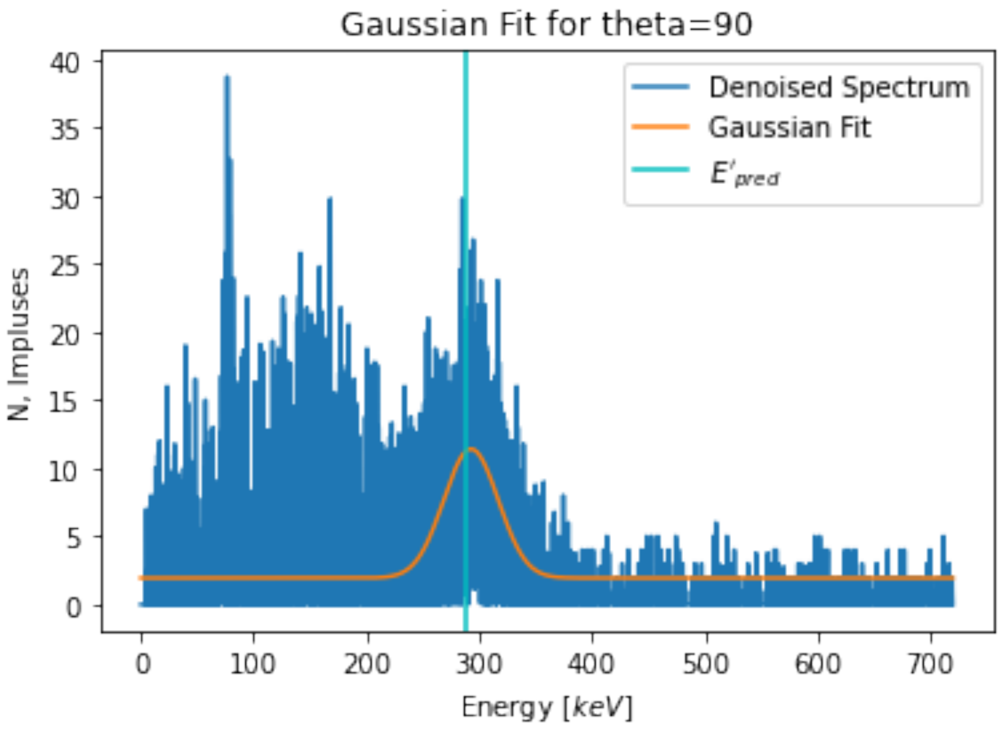
```
For theta=30: E_pred=564.09, E=517.90, sigma=39.67, a=7.41, and h=6.01
```



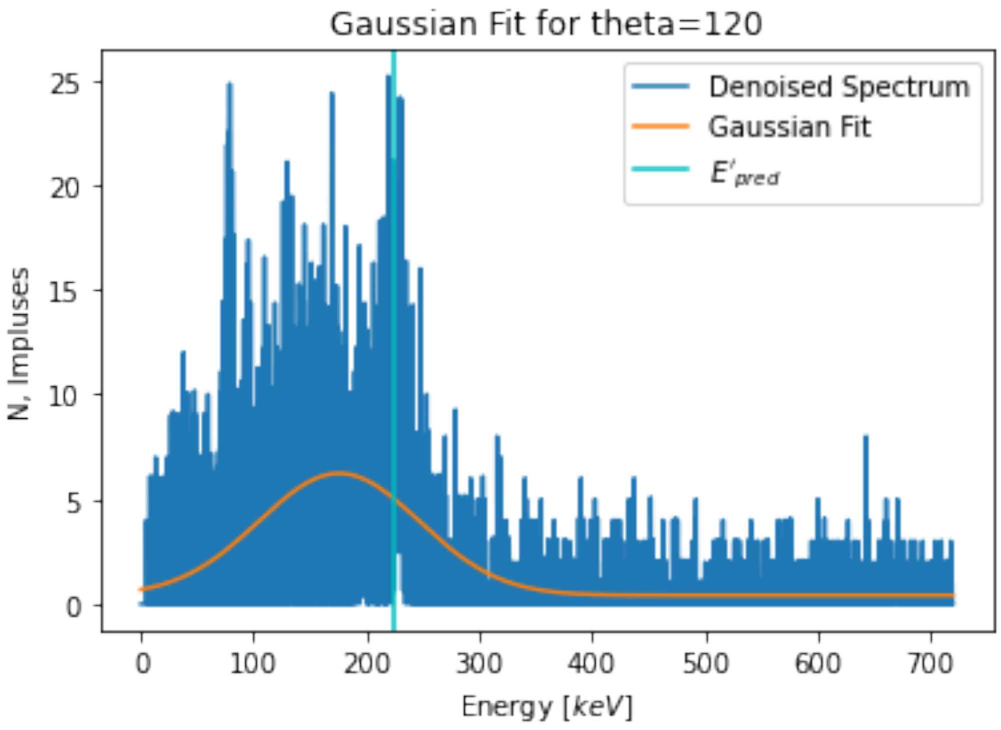
```
For theta=60: E_pred=401.76, E=383.33, sigma=35.90, a=5.38, and h=2.83
```



```
For theta=90: E_pred=288.39, E=292.70, sigma=24.18, a=9.47, and h=1.94
```



```
For theta=120: E_pred=224.92, E=175.78, sigma=71.23, a=5.82, and h=0.41
```



It can be seen that all but $\theta = 120°$, the fit coincides reasonably with the denoised peaks. The Gaussian peak does not however coincide with the predicted peak, and that may be put down to a combination of: - resolution error of the scintillator + MCA setup, - innacurate calibration as each experiment progressed, - or experimental error in θ.

For the $\theta = 120°$ case however, the peak coincides so closely with the x-ray background that remained after denoising that the Gaussian cannot 'discern the two', and so contributes significantly to the fitted x_0 parameter, making any meaningful conclusion possible.

1.6 Conclusion

In conclusion, I investigated four methods of eliminating the background noise from the raw spectrum obtained. I found the simple max-scaled method most effective in leaving the Compton peak unimpacted whilst minimising the x-ray background peak at around 80keV.

Despite these efforts, the Gaussian fits were reasonable but not convincing enough to confirm confidently the Compton Scattering effect. This is suggested by the fitted E' values deviated up to 50keV in some θ trials, outside the common σ ranges of 15-40keV.

This is excluding the case of $\theta = 120°$ where no conclusions can be made due to the Compton peak and background x-ray peak being too close to one another.

Further experiments could be performed for longer time periods than 15 minutes, 30-60 minutes for instance may improve the confidence in analysis. I also suggest a more accurate method for setting up the angle of the source must be considered, as it's reasonably inaccurate to rely on visually aligning the source to the polar chart underneath.

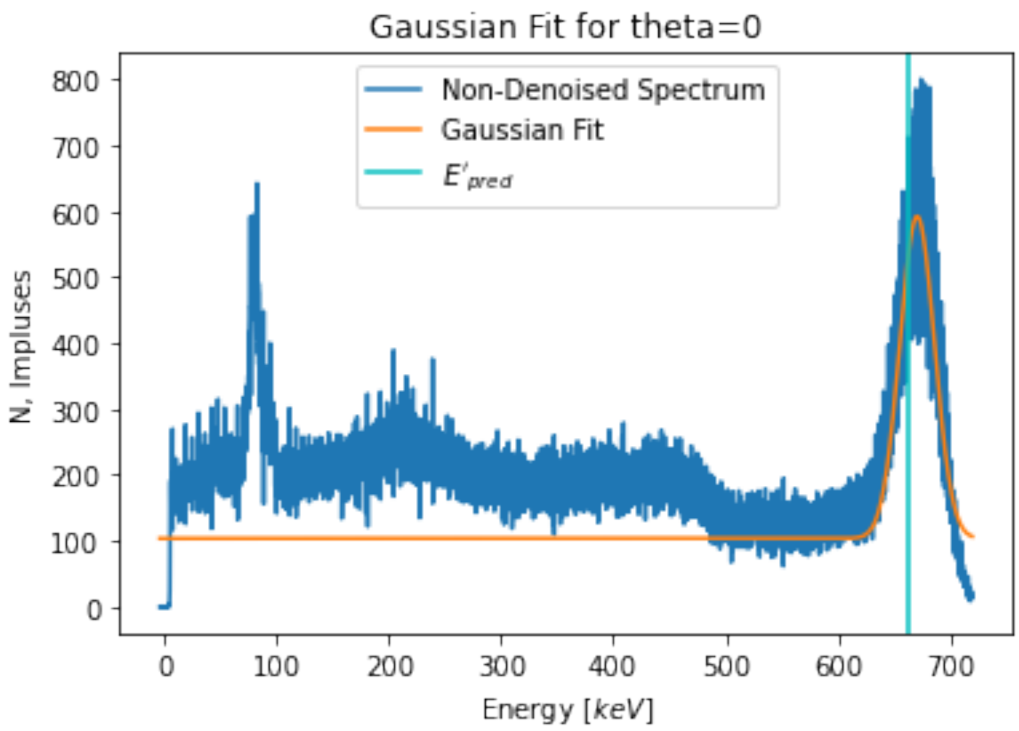
I also suggest further investigation into denoising method 4 mentioned prior. I believe this method holds significant potential in removing the background noise of the raw spectrum, and other functions than that which was used above may be more optimal for use. Investigation into what other noise reduction applications use may present better options than this.

1.7 Appendix

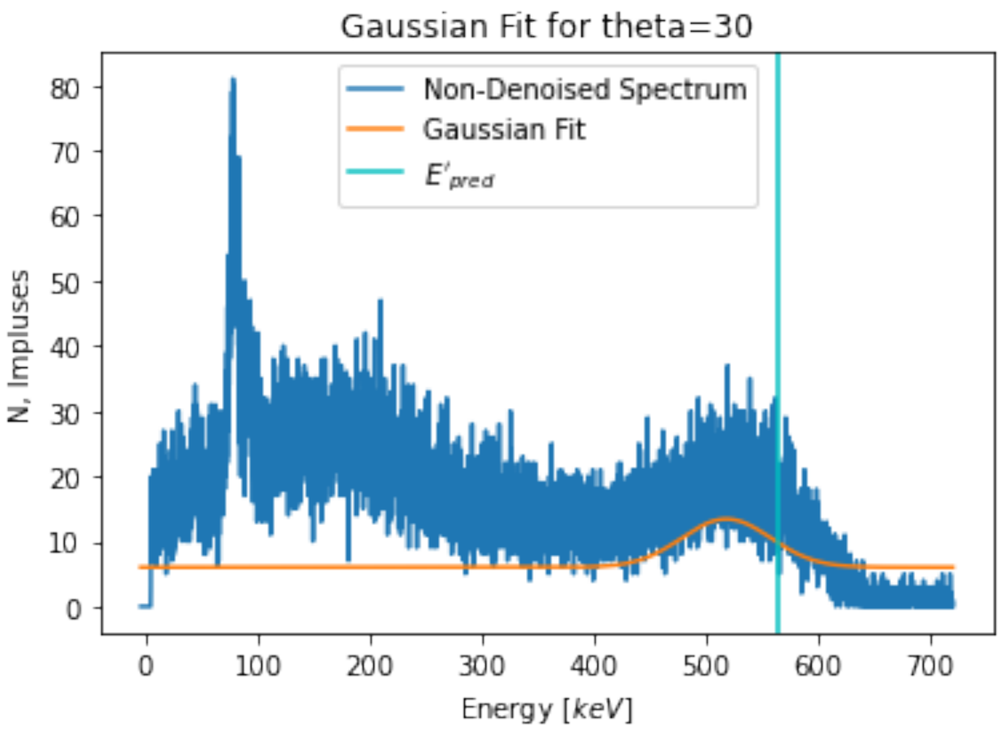
An interesting investigation into the Gaussian fit of the raw spectrum data found essentially no difference in the fitted parameters to that of the denoised spectrum. I found this startling and almost counter-intuitive, not to mention the feeling that I wasted so much time in constructing denoising methods that ultimately had negligble effect.

```
[26]: #%% Fitting non-denoised data
for theta in thetas:
    a, x_0, sigma, h = gauss_fit(theta)
    print(f'For theta={theta}: E_pred={E_pred[theta]:.2f}, E={x_0:.2f},␣
 ↪sigma={sigma:.2f}, a={a:.2f}, and h={h:.2f}')

    plt.figure()
    plt.title(f'Gaussian Fit for theta={theta}')
    plt.plot(data[theta]['E'], data[theta]['N'], label='Non-Denoised Spectrum')
    plt.plot(data[theta]['E'], gauss(data[theta]['E'], a, x_0, sigma, h),␣
↪label=f'Gaussian Fit')
    plt.xlabel('Energy [$k eV$]')
    plt.ylabel('N, Impluses')
    plt.axvline(x=E_pred[theta], label="$E'_{pred}$", color='c')
    plt.legend()
    plt.show()

del theta, a, x_0, sigma
```

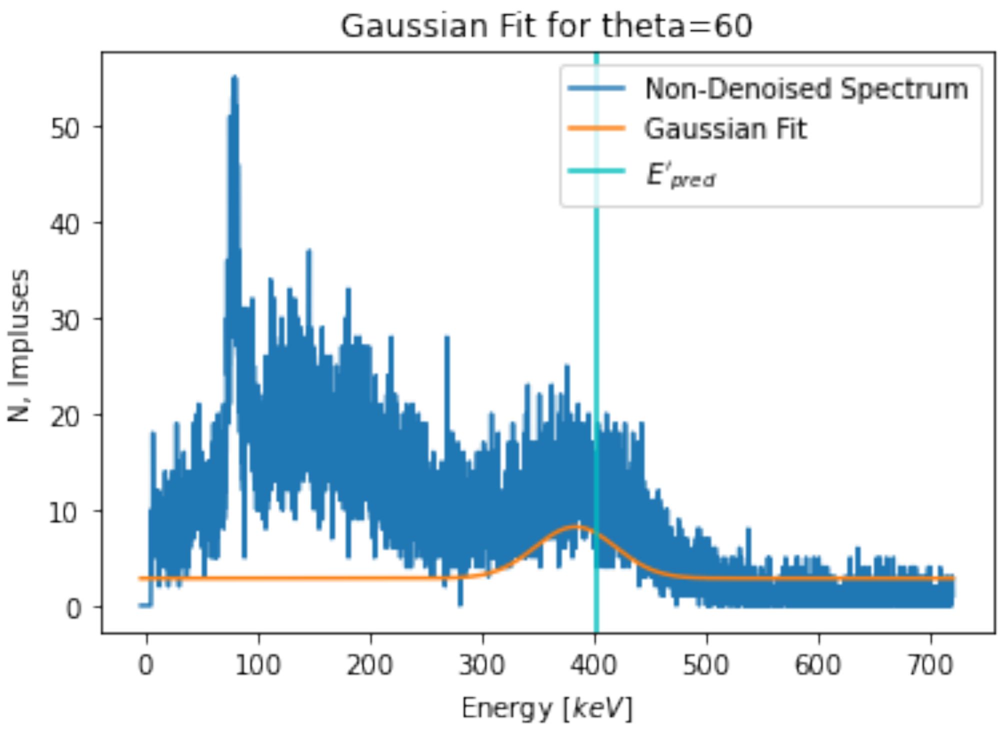
```
For theta=0: E_pred=662.00, E=670.85, sigma=15.45, a=488.31, and h=103.79
```



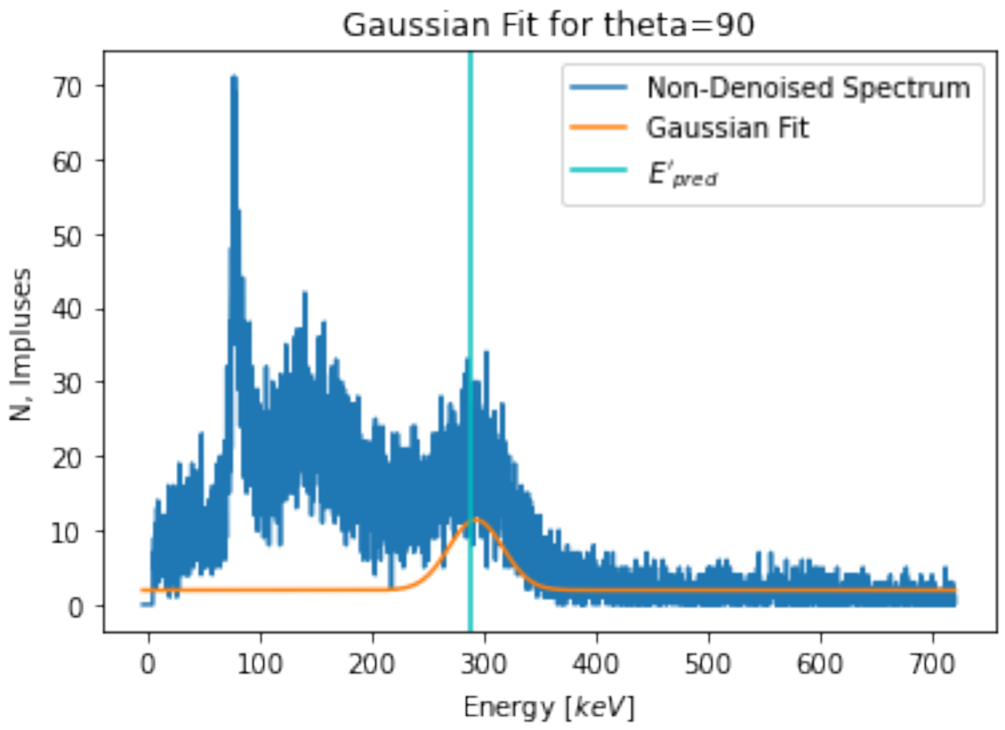
```
For theta=30: E_pred=564.09, E=517.90, sigma=39.67, a=7.41, and h=6.01
```



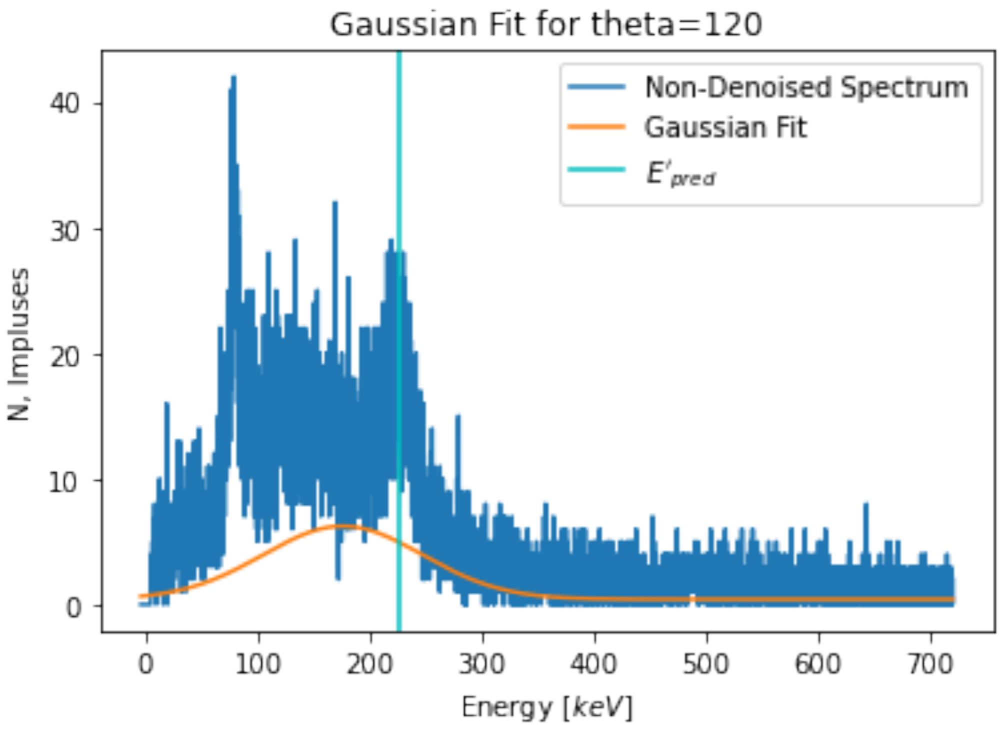
```
For theta=60: E_pred=401.76, E=383.33, sigma=35.90, a=5.38, and h=2.83
```



```
For theta=90: E_pred=288.39, E=292.70, sigma=24.18, a=9.47, and h=1.94
```



```
For theta=120: E_pred=224.92, E=175.78, sigma=71.23, a=5.82, and h=0.41
```


Gaussian Fit for theta=120
Non-Denoised Spectrum
Gaussian Fit
E'_{pred}
N, Impluses
Energy [keV]
0
10
20
30
40
0
100
200
300
400
500
600
700

Compton Effect

Aim: Confirm Compton Formula

$$\Delta \lambda = \lambda' - \lambda = \underbrace{\frac{h}{m_e c}}_{= 2.43\,\text{pm}}(1-\cos\Theta)$$

or

$$\underbrace{E'}_{\text{scattered photon}} = \left(\frac{1}{m_e c^2}(1-\cos\Theta) + \frac{1}{\underbrace{E_0}_{\text{incident photon}}}\right)^{-1}$$

Summary: ✱

1. Pre-Lab /
2. Safe Work Procedure /
3. Calibrate Multi-Channel Analyser (MCA) /

3.5. No source - noise profile

4. Collect data for 4-5 angles between 30°-120° /

Sources:

- Na-22 (blue) - 74 kBq } calibration
- Cs-137 (little red) - 37 kBq } calibration
- Cs-137 (big red) - 18.5 MBq } measurement = E_0 = 662 keV

Scintallator: NaI

✱ avoid impacts

Calibration:

✱ Scintaillator - 6.00 - stabilise for 10 mins /

✱ Measure -> MCA -> Calibration - 3pt calibration //

✱ Place small Cs inside with tweezers /

✱ set up Na in clamp pointing in /

~~~20-30 mins~~ 25 mins

Recording

✱ Measure -> Record spectra =

counts
Energy keV

• Export w/ *.txt
~~~

Q. Why Gaussian ~~[illegible]~~?
Q. Tary

<u>Predictions</u>

θ	$\Delta\lambda$
0°	0.326 pm ↙
30°	" - "
60°	1.215 pm
~~120°~~ 90°	2.43 pm
~~180°~~	
180°	4.86 pm

<u>E's</u>

Q. Why broadening of experimental peaks? Repeated scattering?

Record name, calibration, equation

Look NaI scintillator noise properties

Lab

1. Calibration

$E_1 = 32$ keV

$E_2 = 662$ keV

$E_3 = 511$ keV

Calibration - lvl 1 - offset 0%.

0.2<channel> keV - 4.4 keV

PLAN

Started 10am

10am - ~~10:20am~~ 20mins

~~10:30am~~

~~10:20am - 10:40am~~ no sources

10:40am - 10:45am set up ~~hot~~ source

10:45 -

<u>Plan</u>

2~~0~~5 mins - Calibration - 10:00am - 10:

" " - no source -

~~[illegible]~~ - set-up hot -

15 mins ~~20~~ mins - 30° -

" " - 60° -

" " - 90° -

" " - 120° -

" " - ? -

" " - ? -

Noise Profile

- No nearby ~~[illegible]~~ sources
- Set up steel rod and Pb blocks

$\frac{h}{m_e c}$

$h = 6.626 \times 10^{-34}$

$m_e = 9.109 \times 10^{-31}$

$h = 4.1357 \times 10^{-15}$

$$E' = \left(\frac{1}{m_e c^2}(1-\cos\theta) + \frac{1}{E_0}\right)^{-1}$$

$m_e = 9.109 \times 10^{-31}$ kg
$c = 3 \times 10^{8}$ m s^{-2}
$E_0 = 662$ keV

$m_e = 0.5109 \times 10^{6}$ eV/c2

$$E' = \left(\frac{1}{0.5109 \times 10^{6}\,\text{eV}}(1-\cos\theta) + \frac{1}{\cancel{E_0}\,662 \times 10^{3}\,\text{eV}}\right)^{-1}$$

$E = hf = \frac{hc}{\lambda}$

$\frac{\text{eV}\ \cancel{s}}{\cancel{\text{eV}}\ \cancel{s}\ m^{-2}\ s^{-1}}$ $\cancel{m}\ \cancel{s}^{2}$

~~m~~

θ°	E'/keV
0	662
30°	564.1
45°	479.89
60°	401.7
75°	337.7
90°	288.4
120°	224.9
135°	206.1

Recording.

1. 30° - 15 mins
2. 60° - 15 mins
3. 90° - 15 mins
4. 120° - 15 mins
5. 0° - 15 mins