klein_nishina

December 17, 2019

[1]:
```
import numpy as np
import math as math
deg20Target = np.loadtxt("20DegTarget1.mca")
deg20Background = np.loadtxt("20DegBackground.mca")
# deg60Target = np.loadtxt("60DegDataTarget1.mca")
deg120Background = np.loadtxt("120DegDataBackground.mca")
deg120Target = np.loadtxt("120DegTarget1.mca")
deg160Target = np.loadtxt("160DegDataTarget1.mca")
deg160Background = np.loadtxt("160DegDataBackground1.mca")
```

[2]:
```
deg160Target2 = [x*1.277167903 for x in deg160Target]
# deg80Target2 = [x *  for x in deg160Target]
deg120Background1 = [x*41.62 for x in deg120Background]
deg20Background1 = [x*1.284 for x in deg20Background]
```

[3]:
```
deg20 = (deg20Target - deg20Background1)
deg120 = (deg120Target - deg120Background1)
deg160 = (deg160Target2 - deg160Background)
```

[4]:
```
# Computing the flux at the detector
rate_detector=84.90
area_detector = 0.09
flux_detector = rate_detector/area_detector
# units are counts/cm^2*s
print(flux_detector)
distance_source_detector = 17.5
#units are cm
distance_source = 0.01
flux_source = flux_detector*(distance_source_detector/distance_source)
print(flux_source)
```

```
943.3333333333335
1650833.3333333335
```

[5]:
```
# Klein nishina formula to get theoretical cross section values.
d_target = 2*1.69*.1
# diameter of target in units cm.
h_target = 10
```

```
rho = 2.7
# density of aluminum in units gm/cm^2
NO = 6*10**23
A = 27
Z = 13
# compute total number of electrons in target.
N = math.pi *(d_target/2)**2*h_target*rho*NO*Z/A
print(N)
solid_angle = 0.09/(17.5)**2
print(solid_angle)
```

```
6.998708166775891e+23
0.0002938775510204081
```

[6]:

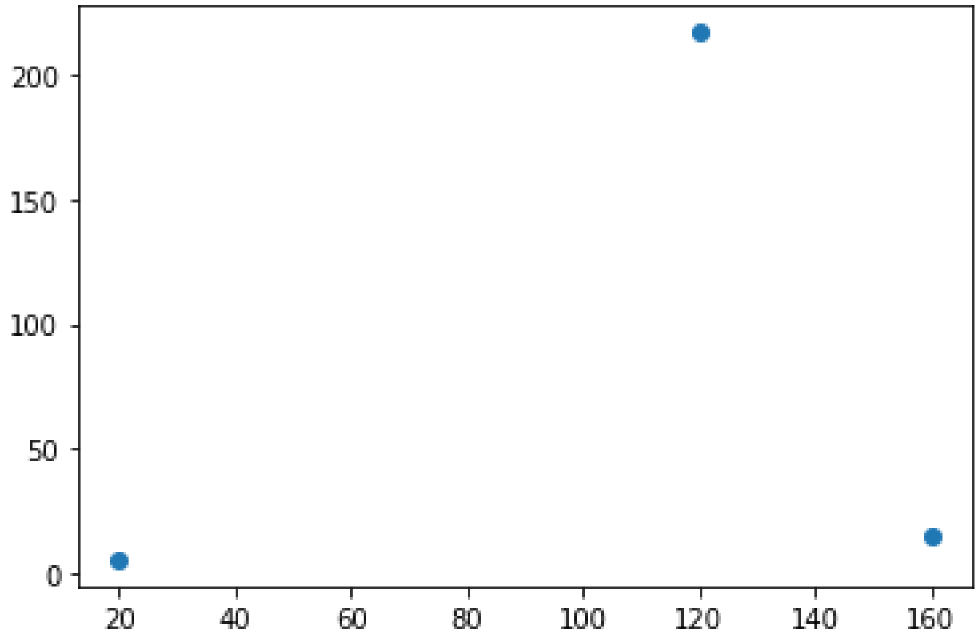
```
# To obtain the cross section we take
rate_detector=84.90
area_detector = 0.09
flux_detector = rate_detector/area_detector
# units are counts/cm^2*s
print(flux_detector)
distance_source_detector = 17.5
#units are cm
distance_source_target = 10.0
flux_target = flux_detector*(distance_source_detector/distance_source)
print(flux_source)
angle = [20, 120, 160]
data = [deg20, deg120, deg160]
counts = [abs(sum(x)) for x in data]
# The units for cross section is cm^2/steradians
cross_section = [i / (solid_angle*N*flux_target) for i in counts]
print(cross_section)
adjusted_cross_section = [i * 10**24 for i in cross_section]
print(adjusted_cross_section)
```

```
943.3333333333335
1650833.3333333335
[5.441887679555535e-24, 2.1718442715528925e-22, 1.5055404884865246e-23]
[5.441887679555535, 217.18442715528926, 15.055404884865245]
```

[9]:

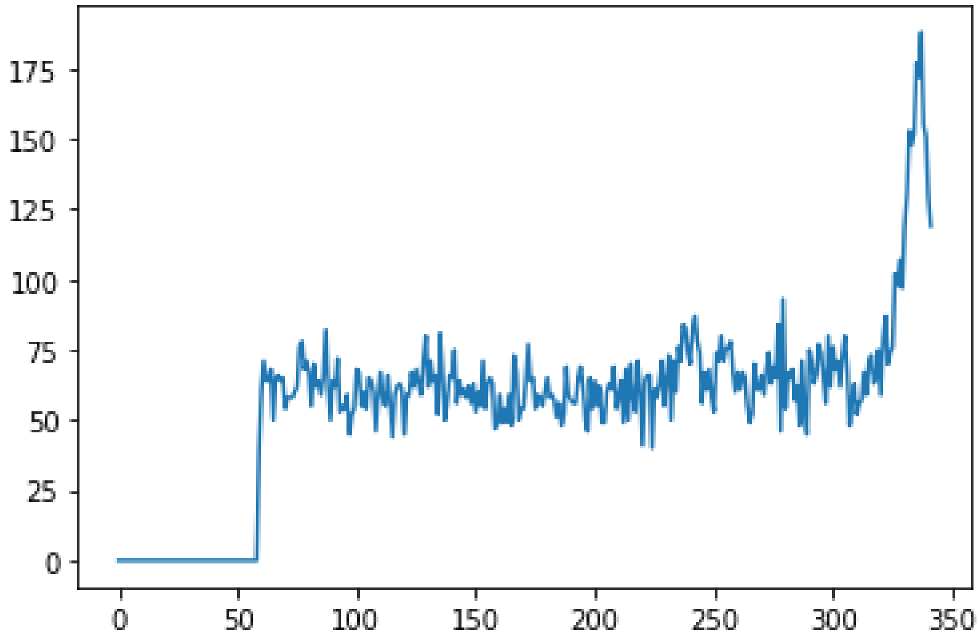
```
# scatter plot of the cross section on the y-axxis and angle on the x axis.
# Not what I was expecting for 160 degrees because it is so much lower than 120␣
 ↪degrees.
# Of course the 120 degrees is also very bad because it was actually -217 when␣
 ↪you take
# the yield of target minus the background.
import matplotlib.pyplot as plt
plt.scatter(angle, adjusted_cross_section)
```

```
plt.show()
```



[25]:
```
# Scattered electron energy distribution (compton continuum).
# The peak above channel 1800 would correspond to about 59.54 keV.
# But I know that the data should just be a range of 0-11 keV based on fact of␣
↪the
# photons back scattering or not scattering in the detector,
# The change in energy of the photon is equal to the energy imparted to the␣
↪electron in the
# semiconductor so it should range between 0-11 keV or channels
data = np.loadtxt("OverTheWeekend.mca")
# multiply by 1000 to convert back to keV from MeV.
energyPerChannel = 3.216639654240951e-05*1000
max_channel = 11/energyPerChannel
print(max_channel)
data = data[0:342]
plt.plot(data)
plt.show()
# The graph looks a bit like the data for the scattered electron energy␣
↪distribution.
# (compton continuum).  That is we see it increasing the number of counts and␣
↪thus the
# cross section as we get closer to the 11 keV mark.  But it dips to zero␣
↪instead of increasing
# as the energy is decreasing.  We expect the minimum to be at about 5.5 keV.
```

```
341.9717836748405
```



[ ]:

[ ]:

[ ]:

plotsssss

December 17, 2019

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
#energy
def scattered_energy(theta):
    E = 0.05954
#     tempE = 0.662
    restEnergyElectron = 0.511
    return E / (1 + (E / restEnergyElectron) *(1 - np.cos(np.deg2rad(theta))))
```

[2]:
```python
# Find the channel where the peak should be at with given angle.
def channel_finder(theta):
    Ef = scattered_energy(theta)
    energyPerChannel = 3.216639654240951e-05
#     energyPerChannel = 0.08327044025157233
    return Ef / energyPerChannel
```

[1]:
```python
# Convert the channel number to the corresponding energy value for that peak.
def channToEn(channel):
    energyPerChannel = 3.216639654240951e-05
#     energyPerChannel = 0.08327044025157233
    return  energyPerChannel * channel
```

[1]: 0.05950783360345759

[4]:
```python
def finale(channels, angles):
    energies = [1 / channToEn(x) for x in channels]
    x_real = [1-np.cos(np.deg2rad(x)) for x in angles]
    plt.plot(x_real, energies)
    plt.show()
    slope, _, _, _, _ = stats.linregress(x_real, energies)
    return slope
```

[5]:
```python
from scipy import stats
# Fundamental idea is that the energy is shifted by the different
# scattered angles of compton scattering.
def energyShiftCompton(energy, angle, error):
```

```python
    energy = [1 / x for x in energy]
    xAxis = [1-np.cos(np.deg2rad(x)) for x in angle]
    plt.errorbar(xAxis, energy, yerr = error, fmt = 'o', capsize = 3)
    plt.show()
    plt.scatter(xAxis, energy, c = "b", alpha = 1)
    plt.title('Scatter Plot of Compton Shift')
    slope, intercept, r_value, p_value, std_err = stats.linregress(xAxis, energy)
    print(slope)
```

[6]:

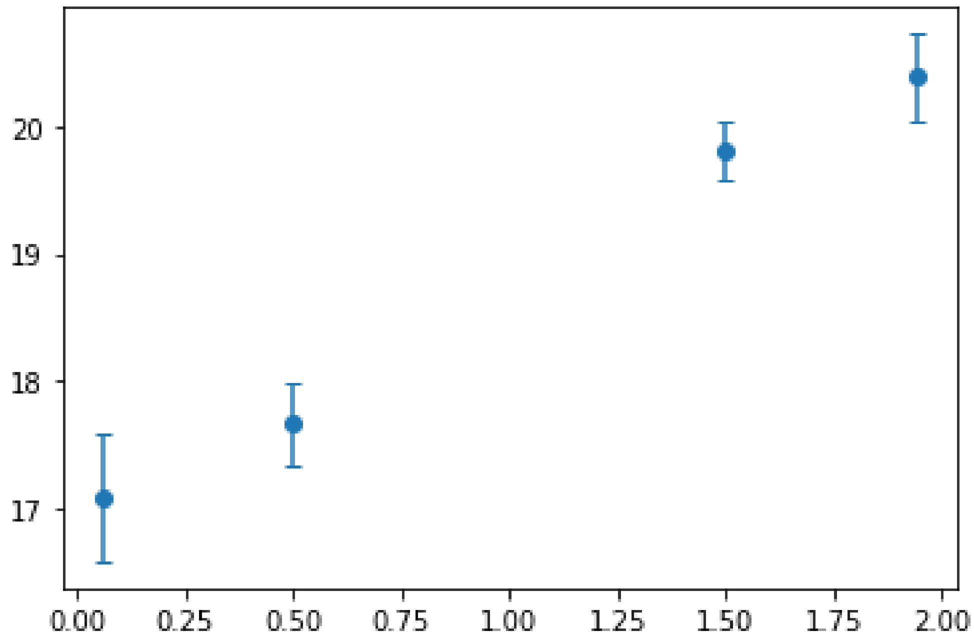
```python
z = [0, 20, 60, 80, 120]
z = [0, 20, 50, 60, 70, 90, 120]
channelList = [channel_finder(x) for x in z]
energies = [scattered_energy(x) for x in z]
print(energies)
```

```
[0.05954, 0.059124542903097176, 0.057160895762107734, 0.056262255672467044,
0.05530036447810277, 0.053326567812949144, 0.05068204760873549]
```

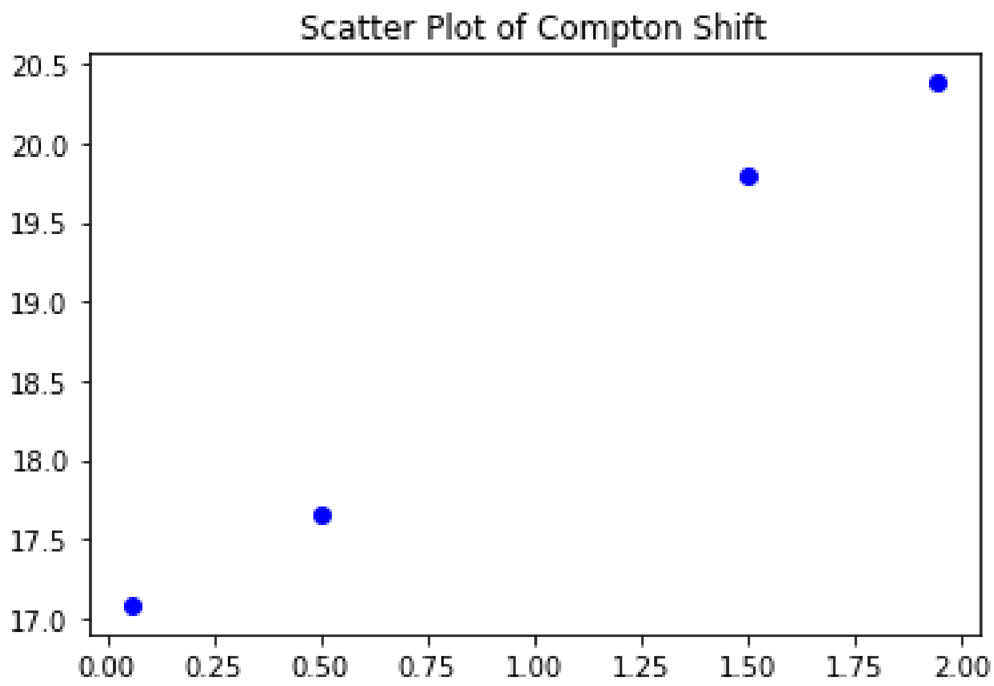
[10]:

```python
# Energy that was collected by analyzing the photopeaks locations and then
# computing the energy based on what channel the peak is located at.
channels = [1820, 1760, 1570, 1525]
energy = [channToEn(i) for i in channels]
print(energy)
angle = [20, 60, 120, 160]
energyError = [0.5, 0.324, 0.234, 0.34]
energyShiftCompton(energy, angle, energyError)
xReal = [1-np.cos(np.deg2rad(x)) for x in angle]
```

```
[0.0585428417071853, 0.056612857914640734, 0.05050124257158293,
0.0490537547271745]
```




```
1.8419096230421097
```



[11]:
```
MeV = (1/
1.8419096230421097)
print(MeV)
```

```
0.5429148029252345
```

[12]:
```
abs(1-MeV/0.511)
```

[12]:
```
0.062455583023942385
```

[13]:
```
#optimize curve fit
import numpy as np
from scipy import stats
angle = [20, 60, 120, 160]
x = np.array([1-np.cos(np.deg2rad(j)) for j in angle])
channels = [1820, 1760, 1570, 1525]
energy = [channToEn(i) for i in channels]
y = np.array([1 / j for j in energy])
x0 = np.array([0,0,0])
sigma = np.array([8.377, 14.252, 12.088, 11.629])
import scipy.optimize as optimization
def func(x, a, b, c):
    return a + b*x + c*x*x
print(optimization.curve_fit(func, x, y, x0, sigma))
```

```
(array([ 1.69356865e+01,  1.84214594e+00, -1.48990564e-02]), array([[
0.03099602, -0.07646379,  0.03266061],
       [-0.07646379,  0.46357467, -0.23353293],
       [ 0.03266061, -0.23353293,  0.12204485]]))
```

[14]:
```
def func(params, x, y):
    return (y - np.dot(x, params))
x = np.transpose(np.array([[1.0, 1.0, 1.0, 1.0], [1-np.cos(np.deg2rad(j)) for j␣
 ↪in angle]]))
args = (x, y)
x0 = np.array([0,0])
print(optimization.leastsq(func, x0, args = (x, y)))
# Least Squares estimate gives the following results.
# slope = b = 1.84190962
# y-intercept = a = 16.89124832
```

```
(array([16.89124832,  1.84190962]), 1)
```

[ ]:
```

```

[ ]:
```

```

```
[ ]:
```

```
[ ]:
```

subtractBackground

December 17, 2019

[4]:
```
import numpy as np
np.set_printoptions(threshold=1000)
import matplotlib.pyplot as plt
```

[5]:
```
# Loading all of my data for analysis.
deg0Target = np.loadtxt("0DegDataTarget1.mca")
deg0Background = np.loadtxt("CalibrationData1.mca")
deg20Target = np.loadtxt("20DegTarget1.mca")
deg20Background = np.loadtxt("20DegBackground.mca")
deg60Target = np.loadtxt("60DegDataTarget1.mca")
deg80Target = np.loadtxt("80DegDataTarget1.mca")
deg80Background = np.loadtxt("80DegDataBackground1.mca")
deg120Background = np.loadtxt("120DegDataBackground.mca")
deg120Target = np.loadtxt("120DegTarget1.mca")
deg160Target = np.loadtxt("160DegDataTarget1.mca")
deg160Background = np.loadtxt("160DegDataBackground1.mca")
a = list(deg0Background)
a.index(max(a)) + 1
```

[5]: 1851

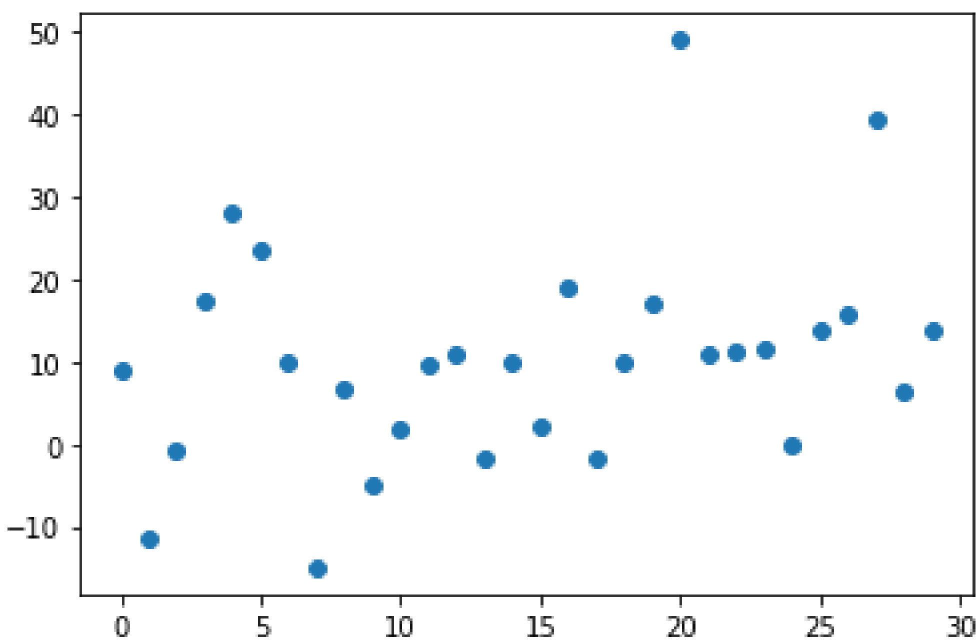
[6]:
```
# Calibration to find the energy scale.
# gain = 8.92
calibration = list(deg0Background)
peakIndex = calibration.index(max(calibration)) + 1
# Units of MeV / channel
energyPerChannel = 0.05954 / peakIndex
print(energyPerChannel)
```

```
3.216639654240951e-05
```

[7]:
```
# Balancing my data so that it matches.
deg160Target2 = [x*1.277167903 for x in deg160Target]
# deg80Target2 = [x *  for x in deg160Target]
deg120Background1 = [x*41.62 for x in deg120Background]
deg20Background1 = [x*1.284 for x in deg20Background]
```

[22]:
```
# Analysis ohe 160 degrees spectra collected.
# 68% of values should be within 1 standard deviation of th emean.
from scipy.signal import find_peaks
from scipy.optimize import curve_fit
from scipy import asarray as ar, exp
deg160 = (deg160Target2 - deg160Background)
itemIndex = np.where(max(deg160)==deg160)
deg160List = list(deg160)
y = deg160[1500:1530]
x = [i for i in range(0, len(y))]
n = len(y)
mean = sum(x*y)/sum(y)
print(mean)
sigma = np.sqrt(sum(y*(x-mean)**2)/sum(y))
print(sigma)
std_error = sigma / np.sqrt(n)
print(std_error)
peaks, _ = find_peaks(x, height = 0)
plt.scatter(x, y)
# plt.scatter(peaks, x[peaks], "x")
# plt.scatter(np.zeros_like(x), "--", color = "gray")
plt.show()
deg160Channel = 1525
deg160ChannelError = 1.52943
```

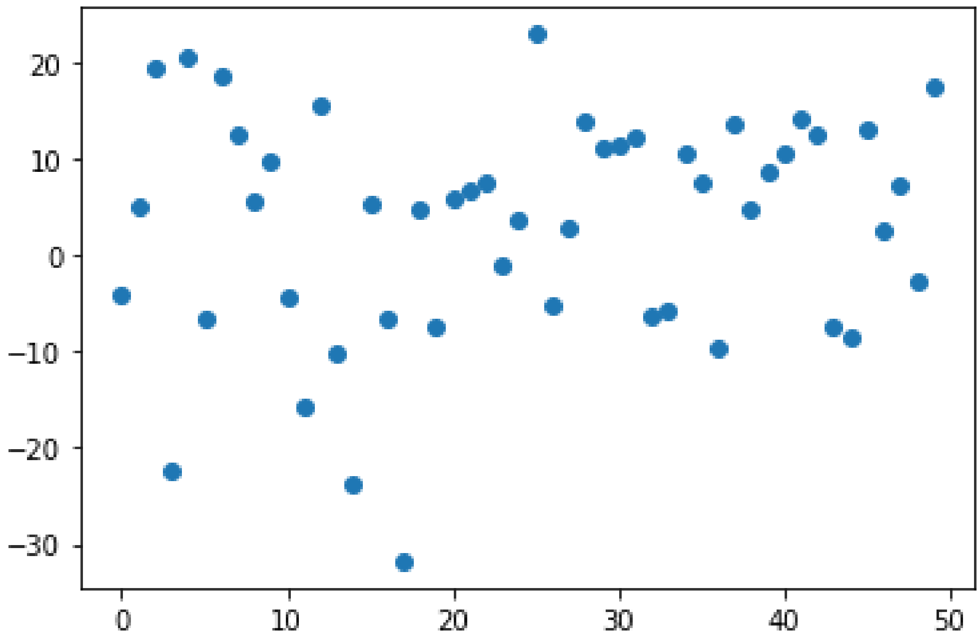
```
17.765834246861843
8.377018162902417
1.52942727081738
```



[38]:

```
# Analyze the data collected from when the detector was at 120 degrees.
deg120 = (deg120Target - deg120Background1)
#print(deg160)
itemIndex = np.where(max(deg120)==deg120)
print(itemIndex)
# print(itemIndex)
deg120List = list(deg120)
y = deg160[1550:1600]
x = [i for i in range(0, len(y))]
n = len(y)
mean = sum(x*y)/sum(y)
print(mean)
sigma = np.sqrt(sum(y*(x-mean)**2)/sum(y))
print(sigma)
std_error = sigma / np.sqrt(n)
print(std_error)
peaks, _ = find_peaks(x, height = 0)
plt.scatter(x, y)
# plt.plot(peaks, x[peaks], "x")
# plt.plot(np.zeros_like(x), "--", color = "gray")
plt.show()
deg120Channel = 1570
deg120ChannelError = 2.01554
```

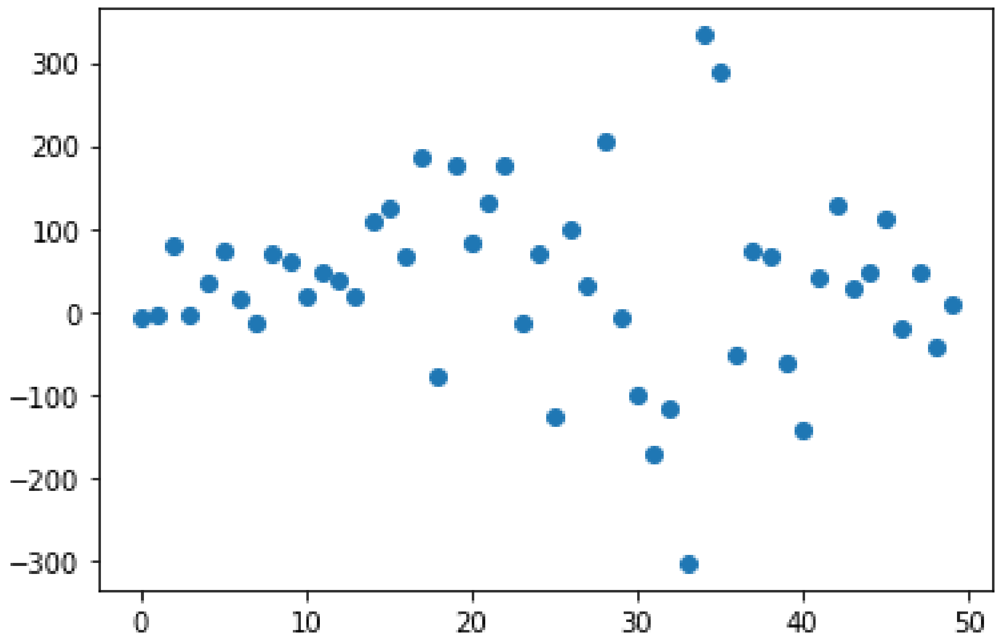
```
(array([1567], dtype=int64),)
32.72620728000071
14.252044987672779
2.0155435313118333
```




```
[50]: # Analyze the degree 20 results.
deg20 = (deg20Target - deg20Background1)
#print(deg160)
itemIndex = np.where(max(deg20)==deg20)
print(itemIndex)
# print(itemIndex)
deg20List = list(deg20)
y = deg160[1820:1870]
x = [i for i in range(0, len(y))]
n = len(y)
mean = sum(x*y)/sum(y)
print(mean)
sigma = np.sqrt(sum(y*(x-mean)**2)/sum(y))
print(sigma)
std_error = sigma / np.sqrt(n)
print(std_error)
peaks, _ = find_peaks(x, height = 0)
plt.scatter(x, y)
# plt.plot(peaks, x[peaks], "x")
# plt.plot(np.zeros_like(x), "--", color = "gray")
plt.show()
deg20Channel = 1820
deg20ChannelError = 1.7094
```

```
(array([1833], dtype=int64),)
20.865504589546152
```

```
12.087565819235161
1.7094399517639816
```

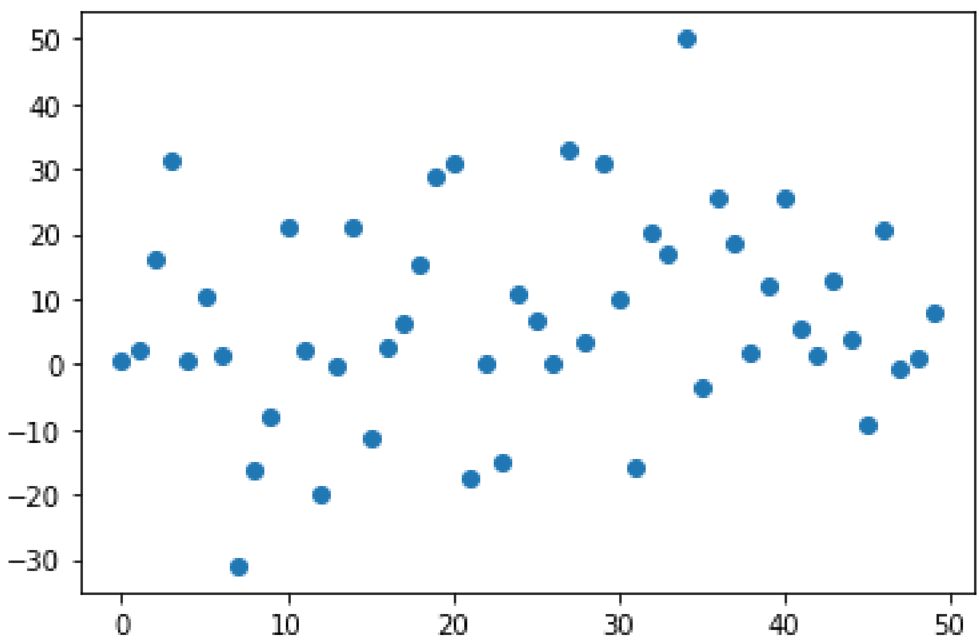


[66]:
```
# Analyze the degree 0 results.
# deg0 = (deg - deg20Background1)
# #print(deg160)
# itemIndex = np.where(max(deg20)==deg20)
# print(itemIndex)
# # print(itemIndex)
# deg20List = list(deg20)
# # print(max(deg160))
# # print(deg160List)
# # print(deg160List[1507])
# x = deg20[1700:1850]
# peaks, _ = find_peaks(x, height = 0)
# plt.plot(x)
# plt.plot(peaks, x[peaks], "x")
# plt.plot(np.zeros_like(x), "--", color = "gray")
# plt.show()
# deg20Channel = 1820
# deg20ChannelError = 5
```

[65]:
```
# Analyze the degree 60 results.
#print(deg160)
itemIndex = np.where(max(deg60Target)==deg60Target)
print(itemIndex)
```

```
# peaks, _ = find_peaks(x, height = 0)
y = deg160[1740:1790]
x = [i for i in range(0, len(y))]
n = len(y)
mean = sum(x*y)/sum(y)
print(mean)
sigma = np.sqrt(sum(y*(x-mean)**2)/sum(y))
print(sigma)
std_error = sigma / np.sqrt(n)
print(std_error)
peaks, _ = find_peaks(x, height = 0)
plt.scatter(x, y)
# plt.plot(peaks, x[peaks], "x")
# plt.plot(np.zeros_like(x), "--", color = "gray")
plt.show()
deg20Channel = 1760
deg20ChannelError = 1.64466
```

```
(array([339], dtype=int64),)
29.84507088134634
11.629486674236265
1.6446577778142106
```



[47]:
```

```

[47]: [range(0, 10)]

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```