surface

November 25, 2021

[12]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
import warnings

warnings.filterwarnings("ignore")
```

[13]:
```
data = pd.read_excel("peak_data.xlsx")
channel = data["voltage"]
distance = data["distance"]
```

0.1 Functions

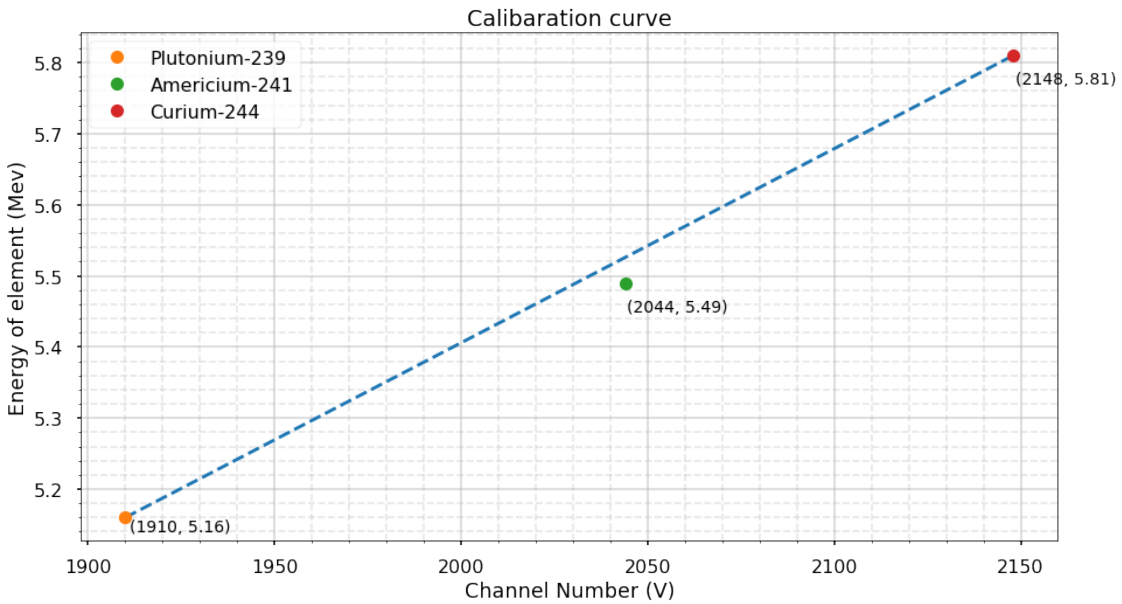
[14]:
```
# function for interpolation
def interpolate(x, y):
    f = interp1d(x, y, kind="linear", fill_value="extrapolate")
    a = np.arange(0, x[len(x) - 1], 0.001)
    b = f(a)
    return a, b
```

1 Calibration Curve

[65]:
```
# 244Cm, 241Am, 239Pu
res_name = ["Plutonium-239", "Americium-241", "Curium-244"]

peak_channel = [1910, 2044, 2148]
known_energy = [5.16, 5.49, 5.81]
pc = [1910, 2148]
ke = [5.16, 5.81]
for i in range(2):
    energy_ratio = known_energy[i + 1] / known_energy[i]
    peak_ratio = peak_channel[i + 1] / peak_channel[i]
    print(
        f"Energy ratios: \n {res_name[i + 1]} / {res_name[i]}: \n Energy =␣
↪{energy_ratio:.2f}, Peak = {peak_ratio:.2f}"
    )
```

```
plt.style.use("seaborn-poster")
plt.figure(figsize=(15, 8))
plt.title(f"Calibaration curve")
plt.xlabel("Channel Number (V)")
plt.ylabel("Energy of element (Mev)")

plt.plot(pc, ke, "--")
for i in range(len(res_name)):
    plt.plot(peak_channel[i], known_energy[i], "o", label=res_name[i])
    plt.annotate(f"({peak_channel[i]}, {known_energy[i]})",␣
 ↪xy=(peak_channel[i]+0.5,known_energy[i]-0.04), fontsize=14)


plt.annotate(f"({peak_channel[0]}, {known_energy[0]})",␣
 ↪xy=(peak_channel[0]+1,known_energy[0]-0.02), fontsize=14)
plt.legend(loc="upper left")
plt.grid(alpha=0.5, which="major")
plt.minorticks_on()
plt.grid(alpha=0.3, which="minor", ls="--")
plt.show()
```

```
Energy ratios:
 Americium-241 / Plutonium-239:
 Energy = 1.06, Peak = 1.07
Energy ratios:
 Curium-244 / Americium-241:
 Energy = 1.06, Peak = 1.05
```



1.1

[16]:
```
# print(peak_channel[0], peak_channel[len(peak_channel) - 1], channel.min(),␣
↪channel.max())
peak, energy = interpolate(peak_channel, known_energy)

# print(peak_channel, known_energy)
# print(peak, energy)
enc = []
for i in range(len(channel)):
    energy_cal = np.interp(channel[i], peak, energy)
    enc.append(energy_cal)
    print(f"{channel[i]}, {enc[i]:.2f}")
```

```
1808, 4.91
1744, 4.75
1678, 4.59
1592, 4.38
1535, 4.24
1447, 4.02
1379, 3.85
1289, 3.63
1223, 3.47
1154, 3.30
1070, 3.09
1000, 2.92
877, 2.62
743, 2.29
661, 2.08
553, 1.82
477, 1.63
310, 1.22
```

[17]:
```
delta_e = []
delta_ex = []

# print(f"de      dex")
for i in range(len(channel)):
    if i <= 16:
        de = enc[i] - enc[i + 1]
        dex = de / 2
        delta_e.append(de)
        delta_ex.append(dex)
        print(f" {dex:.2f}")
```

```
 0.08
```

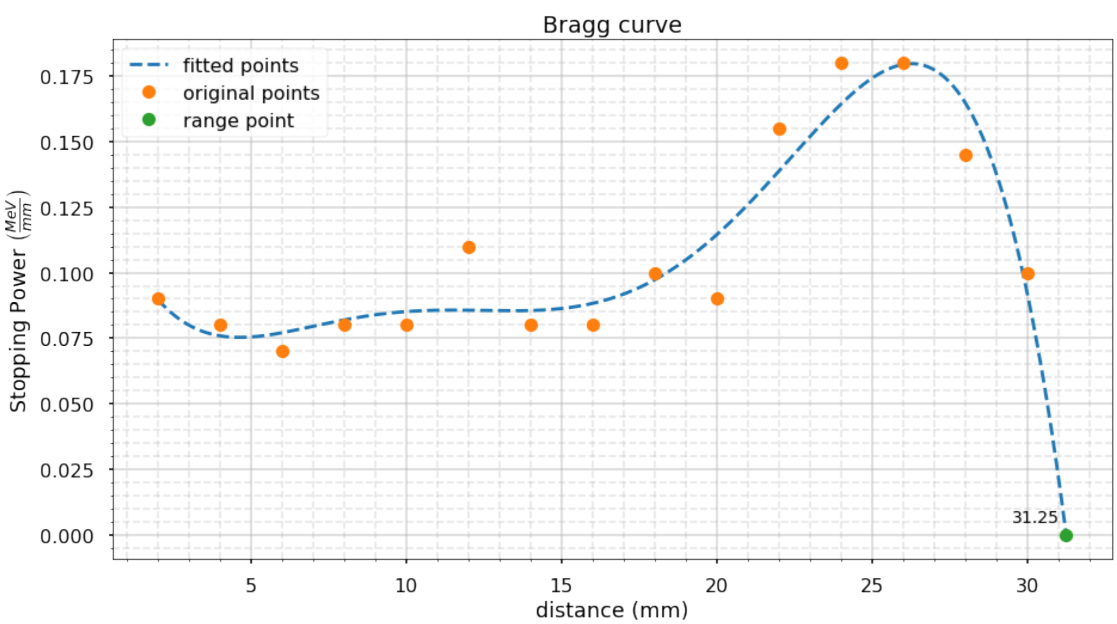
```
0.08
0.11
0.07
0.11
0.08
0.11
0.08
0.08
0.10
0.09
0.15
0.17
0.10
0.13
0.09
0.21
```

1.2 Bragg Curve

[59]:

```
# function for interpolation
def intercube(x, y):
    f = interp1d(x, y, kind="cubic", fill_value="extrapolate")
    a = np.arange(x[0], x[len(x) - 1] + 1, 0.001)
    b = f(a)
    return a, b


# funciton for polynomial fitting
def polfit(a, b, c):
    z = np.polyfit(a, b, c)
    f = np.poly1d(z)

    x = np.arange(a[0], a[len(a) - 1] + 1.25, 0.001)
    y = f(x)
    return x, y


dis1 = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]
dex1 = [0.09, 0.08, 0.07, 0.08, 0.08, 0.11, 0.08, 0.08, 0.10, 0.09, 0.155, 0.
 ↪18, 0.18, 0.145, 0.10]

ds, dexs = polfit(dis1, dex1, 5)
# ds_int, dexs_int = intercube(dis1, dex1)


plt.style.use("seaborn-poster")
plt.figure(figsize=(15, 8))
```

```
plt.title(f"Bragg curve")
plt.xlabel("distance (mm)")
plt.ylabel(r"Stopping Power $\left(\frac{MeV}{mm}\right)$")

plt.plot(ds, dexs, "--", label="fitted points")
# plt.plot(ds_int, dexs_int, "--", label="fitted points")
plt.annotate(f"31.25", xy=(29.5,0.005), fontsize=14)
plt.plot(dis1, dex1, "o", label="original points")
plt.plot(31.25,0, "o", label="range point")

plt.legend(loc="upper left")
plt.grid(alpha=0.5, which="major")
plt.minorticks_on()
plt.grid(alpha=0.3, which="minor", ls="--")
plt.show()

plt.show()
```



[51]:
```
len(dex1)
print(dex1)
dx = 2*dex1
print(dx)
```

```
[0.09, 0.08, 0.07, 0.08, 0.08, 0.11, 0.08, 0.08, 0.1, 0.09, 0.155, 0.18, 0.18,
0.145, 0.1]
[0.09, 0.08, 0.07, 0.08, 0.08, 0.11, 0.08, 0.08, 0.1, 0.09, 0.155, 0.18, 0.18,
0.145, 0.1, 0.09, 0.08, 0.07, 0.08, 0.08, 0.11, 0.08, 0.08, 0.1, 0.09, 0.155,
```

```
0.18, 0.18, 0.145, 0.1]
```