Section2

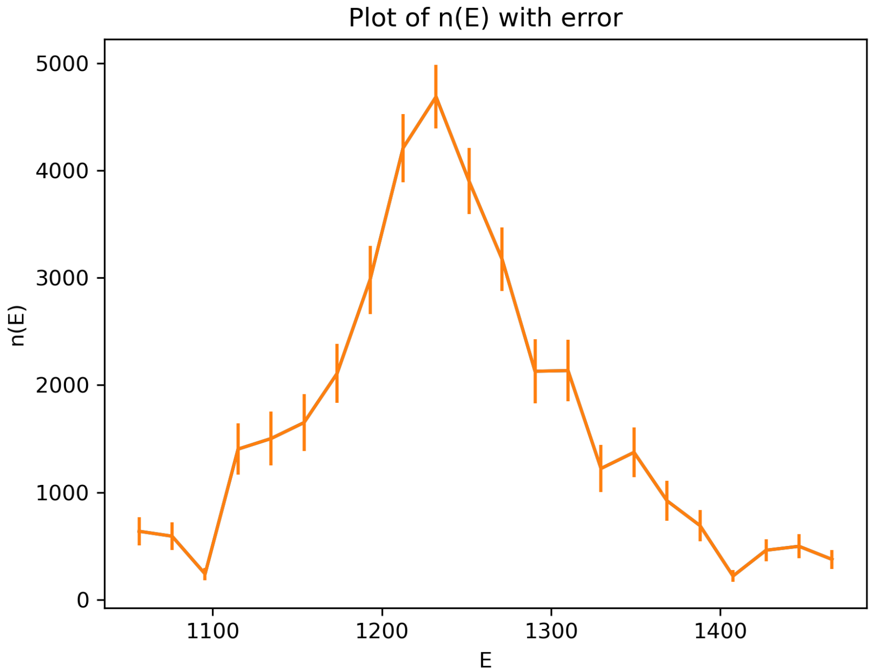
Elliott Ashby

October 27, 2023

1 q1

```
from numpy import linspace
from numpy import arange
from numpy.random import normal
from scipy.optimize import curve_fit
from numpy import array
import matplotlib.pyplot as plt
from minimise import gmin
from statistics import stdev
four = __import__('2_4')


def theory(e: float, w: float):
    th = [(1.4767e7 / (w ** 2 / 4 + (i - 1232) ** 2)) for
        ↪ i in e]
    return th


def parse_data(file: str):
    ea = []
    na = []
    dn = []
    with open(file, 'r') as f:
        for line in f:
            estr, nstr, errstr = line.split()
            ea.append(float(estr))
            na.append(float(nstr))
            dn.append(float(errstr))
        ea = array(ea)
        na = array(na)
        dn = array(dn)
    return [ea, na, dn]


def minimum_discrepancy(ea: list, na: list, dn: list):
    r = []
    minimum = []
```

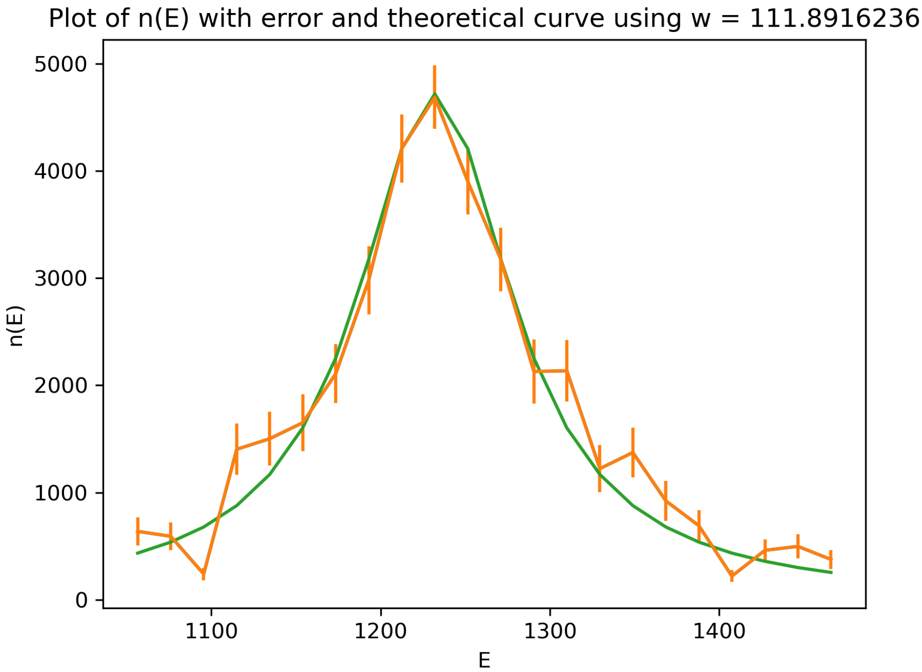
```
    ranger = arange(80, 120, 0.01)

    for i in ranger:
        w = i
        r = [(na - theory(ea, w))]
        minimum.append(four.discrepancy(r, dn))

    mindiscrep = min(minimum)
    w = ranger[minimum.index(min(minimum))]
    return (w, mindiscrep)


def one():
    ea, na, dn = parse_data('./data1.txt')

    def discrep(w: float):
        r = [(na - theory(ea, w))]
        return four.discrepancy(r, dn)

    # print('minimum discrepancy and w: ' + f'{
    ↪ minimum_discrepancy(ea, na, dn)}')
    # print('gmin discrepancy and w: ' + f'{gmin(discrep,
    ↪  80, 120, tol=3.0e-8)}')
    w = 111.8916236

    plt.title('Plot of n(E) with error')
    plt.plot(ea, na)
    plt.errorbar(ea, na, dn)
    plt.xlabel('E')
    plt.ylabel('n(E)')
    plt.savefig('2_1.png', dpi=300)

    plt.title('Plot of n(E) with error and theoretical
    ↪ curve using w = ' + f'{w}')
    plt.plot(ea, theory(ea, w))
    plt.savefig('2_3th.png', dpi=300)
    plt.clf()
```



2 q2

As E increases from 1000 n(E) increases at an increasing rate until around 1200 where the gradient change decreases to a peak at around 1250. This shape is mirrored on the other side ending around 1500.

3 q3



$$w = \frac{600\sqrt{163}}{\sqrt{4687}} \tag{1}$$

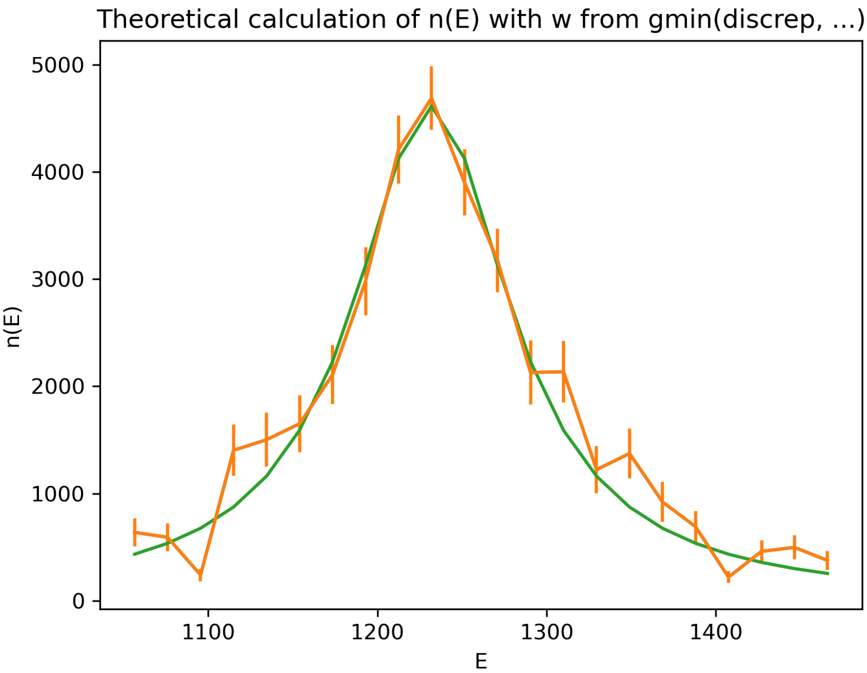
Changing w effects the maximum value of the peak of the theoretical curve without changing the minimum values at roughly $n(E) = 500$. Increasing w decreases the maximum (since w^2 is divided by in $n(E)$). Whereas decreasing w increases the maximum of $n(E)$.

Bonus

```
def one_theory():
    ea, na, dn = parse_data('./data1.txt')

    def discrep(w):
        r = [(na - theory(ea, w))]
        return four.discrepancy(r, dn)

    plt.title('Theoretical calculation of n(E) with w
        from gmin(discrep, ...)')
    plt.plot(ea, na)
    plt.errorbar(ea, na, dn)
    plt.xlabel('E')
    plt.ylabel('n(E)')
```

```
plt.plot(ea, theory(ea, gmin(discrep, 80, 120, tol
    ↪ =3.0e-8)[0]))
plt.savefig('2_5.png', dpi=300)

x = linspace(start=ea[0], stop=ea[-1], num=len(theory
    ↪ (ea, gmin(discrep, 80, 120, tol=3.0e-8)[0])))
popt, pcov = curve_fit(theory, x, theory(ea, gmin(
    ↪ discrep, 80, 120, tol=3.0e-8)[0]), sigma=dn,
    ↪ absolute_sigma=True)
# print('Theory: curve fit w: ' + f'{popt}' + ' +- '
    ↪ + f'{pcov}')
plt.clf()
```

Manually using trial and error to reduce the discrepancy results in a minimum discrepancy of 86.76 which is when $w = 113.15$. The method used is just iterating on the value of w and returning the value of the discrepancy each time as seen in the "minimum_discrepancy" function.
Using gmin() on the discrepancy function reveals a minimum w of 113.14847521305097 where the discrepancy is 86.76529920076851. Using these values in a plot yields:



4 q4

```
from numpy import sum as npsm
```

```
def discrepancy(r, dn):
    '''where r is the residuals,
    and dn is the error on the residuals'''
    return npsm([(r[i] / dn[i]) ** 2 for i in range(len(r
        ↪ ))])
```

This function takes in the values of the residuals and their errors and returns a sum over each residual divided by their error squared.

5 q5

```
    plt.plot(ea, theory(ea, gmin(discrep, 80, 120, tol
        ↪ =3.0e-8)[0]))
    plt.savefig('2_5.png', dpi=300)

    x = linspace(start=ea[0], stop=ea[-1], num=len(theory
        ↪ (ea, gmin(discrep, 80, 120, tol=3.0e-8)[0])))
```

This outputs a value of w of 113.14847521 with an uncertainty of 4.84317108 (This is the value of w that minimises the discrepancy function which was found with the method in my bonus section.)

6 q6

```
def six():

    def random_sample(na, dn) -> float:
        return normal(na, dn)

    def student(ea: list, na: list, dn: list) -> float:
        m = []
        for j, k in zip(na, dn):
            m.append(random_sample(j, k))
        popt, pcov = curve_fit(theory, ea, m)
        return float(popt[0])

    def all_students(ea: list, na: list, dn: list,
        ↪ students: int) -> list:
        w = []
        for i in range(students):
            w.append(student(ea, na, dn))
        sigma = stdev(w)
        return (sum(w) / students, sigma, w)

    students: float = 1000
    ea, na, dn = parse_data('./data1.txt')
    w, sigma, wlist = all_students(ea, na, dn, students)
    # print('q6: w: ' + f'{w}' + ' +- ' + f'{sigma}')
    plt.xlabel('w')
    plt.ylabel('count')
```

```python
plt.title('Histogram of w ensemble values for 1000
    students')
plt.hist(wlist, bins=20)
plt.savefig('2_7.png', dpi=300)
plt.clf()

standard_error = sigma / (students ** 0.5)
# print(standard_error)
```

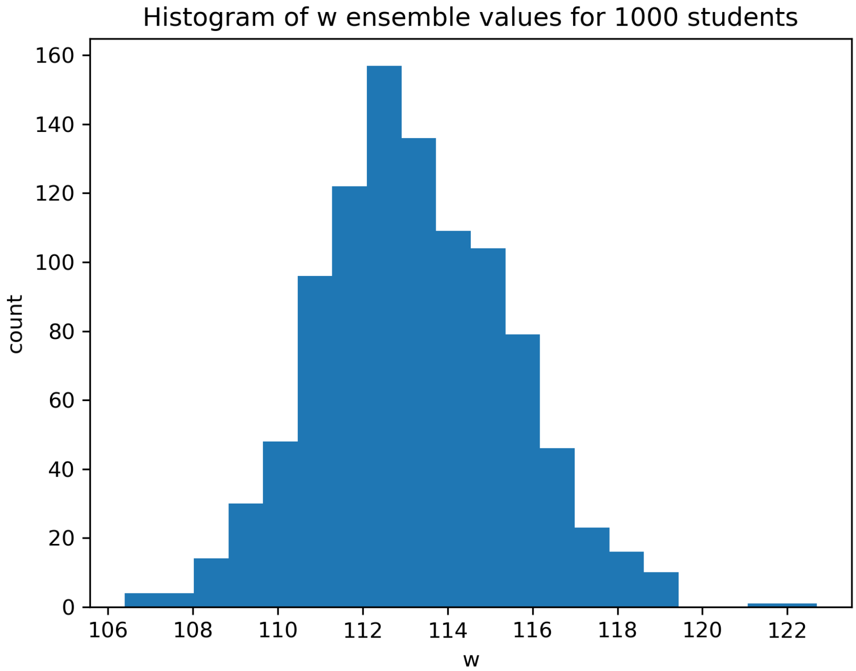
We implement the algorithm described in the question. We start with finding the w of each individual student in the function student, this returns a value of w calculated from a curve fit of randomized values in the shape of a normal distribution with mean and variance defined by data1.txt.
We then define an all_students function which takes in the number of students and the data from data1.txt where we append every students individual value of w into a new list. From this we can find the standard deviation using stdev. This function then returns the mean of all students w values and the standard deviation of all students w values along with the raw list of w.

Using this algorithm we find that the mean of all students w values is:

$$113.20577213567762 \pm 2.278191315242165$$

Obviously, since we are using a random number generator to generate the values of w for each student, the values of w will change each time the program is run. However, the mean and standard deviation of the values of w will remain roughly the same.

7 q7



The plot shows the distribution of the values of w from each student. I picked 20 bins as there are roughly 20 integer value of w that capture the majority of the normal distribution.

There is some uncertainty in the w (or the histogram's mean) which can be calculated as standard error like:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}} \tag{2}$$

Where $\sigma_{\bar{x}}$ is the standard error of the mean, σ is the standard deviation of the sample and n is the number of samples.
Using this equation we can calculate the standard error of the mean of the histogram as:

$$\sigma_{\bar{x}} = \frac{2.278191315242165}{\sqrt{1000}} = 0.07204273501779916 \tag{3}$$