Measurement of β-ray spectra

Ana C. Fabela Hinojosa School of Physics and Astronomy, Monash University

Experiment performed: Tuesday 18th August, 2020, Submitted: September 8, 2020

Abstract

Using a thin lens magnetic spectrometer, we measure the momentum spectrum of electrons emitted as β^- rays from a radioactive source of ^{137}Cs. The detected momentum of the radiated electrons is defined by the spectrometer's adjustable magnetic lens current and k a proportionality constant dependent on the geometry of the apparatus. The magnetic field of the lens is varied by changing the current passing though the lens coil which has the effect of modifying the trajectories of the electrons, focusing electrons with specific momenta onto the detector allowing us to measure their intensity. By converting the measured momentum to energy we are able to fit our data to a linear model based on the Fermi-Kurie plot. We find that the value of the kinetic energy of the nuclear transition is $T = 0.514 \pm 0.05$ MeV which is in agreement with the accepted value of $T = 0.512$ MeV[1].

1 Introduction

When Henri Becquerel first observed $\beta-$radiation, he determined that the observed radiated particle satisfied the same mass-to-charge ratio as the electron, discovered in 1897 by J.J Thompson[2].

Later experimental results showed that $\beta-$rays are detected with a continuous range of kinetic energies up to a maximum value[3]. The discovery of a continuous distribution of electron kinetic energies rather than a discrete predictable value led Wolfgang Pauli to propose in 1930 that the observed violation of conservation laws must be due emission of a yet unknown particle.

In 1934 Enrico Fermi called this apparently massless and undetectable particle the "neutrino", developing an advanced theory of beta decay. The neutrino was finally experimentally observed 1956.[4]

The process we currently know as β^- decay describes a neutron in a parent nucleus desintegrating into a proton in a daughter nucleus, an electron and an antineutrino.

In a β^-event, both nuclides (nuclear species) have the same number of nucleons. This means that the daughter nucleus will not experience a substantial change in kinetic energy (recoil) due to the decay event. Leaving most of the desintegration energy available to be carried-off by the leptons as kinetic energy.

A parent nucleus has a given initial energy w. The avalilable kinetic energy of the system is equal to the decrease in mass energy due to the creation of the radiated leptons. In relativistic units:

$$T = w - 1, \tag{1}$$

The observable count of β^-electrons n as a function of energy is described by the Kurie–Fermi Theory of β^-decay.

2 Background Theory

In this experiment we measure the momentum spectrum of emitted $\beta-$rays from a radioactive source of ^{137}Cs into an excited state of ^{137}Ba. This transition occurs with a probability of 94.6% at a maximum energy value $T = 0.512$ MeV.[1].

A set of electrons with a specific momentum range is focused onto the spectrometer detector, while electrons outside this range undergo chromatic aberration.

The use of coordinates of momentum instead of energy in $\beta-$ray spectroscopy is partly due to the fact that it is the momentum of the focused electrons that is rigorously proportional to the axially symmetric magnetic field.[5, 6] In our experimental setup, the magnetic field is proportional to the adjustable current I_{lens} going through the lens coils. The definition for the momentum of emitted electrons is:

$$p = e\rho B, \tag{2}$$

where B is the magnetic field strength, e is the electron charge, ρ is the gyroradius of the electrons due to B.

The magnetic rigidity P is a measure of the momentum of electrons[7]:

$$P = B\rho, \tag{3}$$

From this relation and the above definition of the momentum of electrons, we write:

$$p = kI_{lens}, \tag{4}$$

k is a constant determined by the geometry of the spectrometer alone[5].

2.1 K-peak Calibration

To calibrate the observed momentum distribution we use electrons emitted with a characteristic well-defined kinetic energy[1]. These electrons are named conversion electrons. In this experiment we study the most probable energy transition from ^{137}Cs to ^{137}Ba. in this transition ^{137}Ba is in an excited state. One way for the daughter atom to lose energy is by transferring the excess energy directly to an orbital electron[1].

The orbital will most likely be the K-shell since it is the lowest energy orbital. A higher energy group event is much rarer (probability of 6%), therefore little error is made by assuming that the peak is due to the K line only.[1].

The constant k in (3) is determined by calibrating the observed spectrum to the well-known K-conversion peak with kinetic energy $T_k = 624.21$ keV. In relativistic units, the calibration calculation is as follows: we write equation (1) in terms of the momentum p_k

$$T_k = \sqrt{{p_k}^2 + 1} - 1, \tag{5}$$

$$\therefore p_k = \sqrt{(T_k + 1)^2 - 1}, \tag{6}$$

from equation (4)

$$kI_k = \sqrt{(T_k + 1)^2 - 1}, \tag{7}$$

$$\therefore k = \frac{\sqrt{(T_k + 1)^2 - 1}}{I_k}, \tag{8}$$

is the proportionality constant we are after.

2.2 Kurie–Fermi theory

The standard method for determining end-points of *beta*−ray groups and for examining the degree of forbidden-ness of the transitions[1] is known as the Kurie plot. The observed maximum value in the Kurie–Fermi plot represents the count of electrons that take the maximum possible kinetic energy whilst the antineutrinos carry close to zero kinetic energy from the transition.

The Kurie–Fermi plot may be written as:

$$n(p) = K_1 F(Z, w) p^2 (w_0 - w^2) S_n(w), \tag{9}$$

where K_1 is an arbitrary constant, p is the electron's momentum, $F(Z, w)$ is the Fermi function (which accounts for coulomb attraction between the electron and the daughter nuclleus). Z is the charge of the daughter nucleus, w_0 is the decay energy, and w is the total transition energy and $S_n(w)$ is known as the shape factor.

In our analysis we use a modified Fermi function: $G = \frac{pF(Z=55,w)}{w}$. We find The value of this function using an interpolation method based on a data set provided in the aditional resources of the experimental script.

In order to use this relation to obtain the transition energy, we are told linearise the Kurie plot. Written in terms of the interpolated fermi function, we write the linear kurie plot as:

$$\sqrt{\frac{n(p)}{p^2 w G S_n(w)}} = K_2(w_0 - w)^2, \tag{10}$$

The shape factor alters the shape of the spectrum depending on the level of "forbiddeness" of a transition. It is determined by the amount of orbital angular momentum, L, carried away by the electron-neutrino pair, as well as their linear momenta)[8].

We obtain an expression for the shape factor from Siegbahn, reference[6]:

$$S_n = w^2 - 1 + (w_n - w)^2, \tag{11}$$

The shape factor increases the precision of the result as the order icreases. The zeroth order linear Kurie plot is found by calculating

$$\sqrt{\frac{n(p)}{p^2 w G}} = K_2(w_0 - w)^2, \tag{12}$$

where the transition is allowed: $S_0(w) = 1$.

We proceed to find the zeroth order energy transition w_0

$$w_0 = \frac{c}{-K_2}, \tag{13}$$

After we have obtained w_0 we can iteratively repeat the previous analysis. From this we obtain higher order w_n and S_n values. We use the results from the iteration to find a convergent value for T the transition energy.

3 Method

Our experimental apparatus is a thin magnetic-lens spectrometer. The operation of β spectrometers depends on the behaviour of electrons subject to magnetic fields.

The spectrometer is aligned parallel to the horizontal component of the Earth's magnetic field, Then the horizontal component of the field does not affect the electron paths. The vertical component is nullified by an adjustable bias coil current from a pair of Helmholtz coils.

The magnetic field of the spectrometer lens is varied by changing the current passing though the lens coils. Modifying a cone of electron trajectories diverging from the source along the spectrometer's axis, causing them to spiral around the axis of the instrument towards detector[1].

3.1 Constant time vs constant counts

The counting controls of the experimental apparatus can be defined by the user. For constant time, the best estimation for the uncertainty in the number of counts n is $\sqrt{n}$[1] due to the poissonian nature of radioactive decay. The fractional uncertainty in counts is $\frac{\sqrt{n}}{n} = \frac{1}{\sqrt{n}}$ Fractional uncertainty is a preserved quantity. Therefore the uncertainty in time is :$\Delta t = \frac{t}{\sqrt{n}}$

From this result we note that with both methods we are able to pre-determine the fractional uncertainty for a data point. But an obvious constraint from choosing the constant counts method is that the alloted time for the experiments is not as easily monitored as with constant time. Given the possible time constraint. The counting controls are set to constant time. The interval used to count events isdefined by trying to mantain the uncertainty in the count as less than 5 percent.

$$\frac{t}{\sqrt{n}} \leq \frac{1}{20}, \tag{14}$$

$$\therefore n \leq 400, \tag{15}$$

From an earlier attempt at the experimental data acqusition we learned that the maximum time interval $t = 600s$ yielded $n \approx 2000$ counts. A simple calculation comparing the count ratios yields the time interval $t = 360s$ as the upper bound for our constant time interval.

3.2 Background radiation

After we obtain the raw spectrum n_{raw} as a function of the lens current. The first processing step is to correct our data from the background radiation count. The 4 measurements of the background radiation counts are added, averaged and the average is subtracted from the raw count.

3.3 Resolution

In a magnetic spectrometer with fixed geometry and variable B the resolution

$$R = \frac{\Delta(B\rho)}{B\rho}, \tag{16}$$

is constant (in this experiment $R = (2 - 3\%)$). Where $\Delta(B\rho)$ is a measure of the accepted momentum band (Δp). When plotting the momentum distribution it is necessary to divide the number of counts n(p) at each current setting by the corresponding current in order to get the correct form of the spectrum[6].

$$n(I) = \frac{n_{\text{net}}}{I_{lens}}, \tag{17}$$

3.4 Experimental procedure

Firstly, Decide what range and increments of lens current are adequate to resolve the momentum spectrum. Then, using the experimental control interface:

1. Calibrate the magnetic field probe. Coordinates of the probe must be set to: $(400, 190)$
2. Null Earth's magnetic field: Disable the lens current. Enable the bias coil current and increase this current until all displayed field values are as close to zero as possible. Note: (the x-component of the field will remain large compared to the other components.)
3. Set counting controls to either constant time or constant counts. Specify time interval or expected counts.
4. Background rate count: Close the source shutter. With the lens current disabled, proceed to run the experiment and count the background radiation (repeat this step 4 times).

After these steps are finalised we can start acquiring β^- radiation data from our source of ^{137}Cs.

3.5 Data acquisition algorithm

1. Enable the lens coil current.
2. Set the shutter status to open.
3. Run the experiment.
4. Increase the lens coil current. Repeat steps (3-4) until reaching the maximum value of the coil current range.

4 Results

The experimental parameters used are as follows: The lens current range is set from 0A to 3.6A Increments of $0.1A$ are chosen. To minimise earth's magnetic field we set the bias coil current to $I = 0.7167 \pm 0.0005$ Counting controls were set to constant time: The intervals $t = 360$s.

4.1 The momentum spectrum

We calibrate the measured spectrum by using the conversion peak energy T_k. Using equation (10) we proceed to find the constant of proportionality in equation (5).

Table I. K-peak parameters and uncertainties.

I_k	1.89
$u(I_k)$	0.05
k	1.05
$u(k)$	0.03

Table II. Corrected count, momentum, energy data and uncertainties. To be used in linear Kurie Plot, We compute the relativistic energy value from the momentum $w = \sqrt{p^2 + 1}$.

n	u(n)	p [mc]	u(p) [mc]	$w[mc^2]$	$u(w)[mc^2]$
1612	20	0.73	0.02	1.24	0.02
1441	20	0.83	0.02	1.30	0.02
1318	18	0.94	0.03	1.37	0.02
1291	16	1.04	0.03	1.44	0.03
1040	15	1.15	0.03	1.52	0.03
798	14	1.25	0.03	1.60	0.03
589	13	1.36	0.04	1.68	0.04
356	12	1.46	0.04	1.77	0.04
190	11	1.57	0.04	1.86	0.04
62	10	1.67	0.04	1.95	0.04

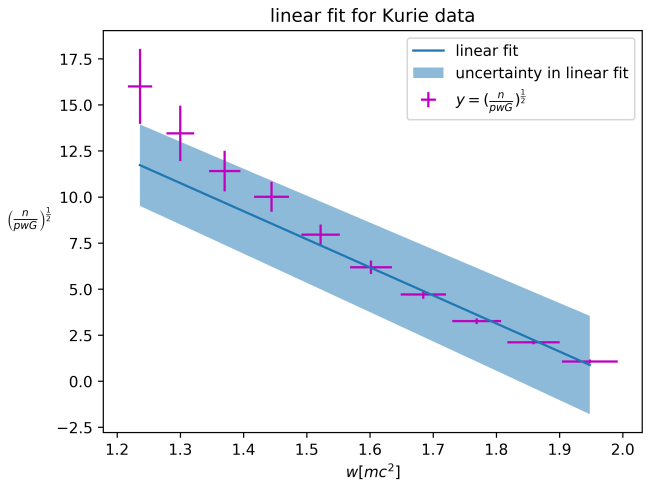

Figure 1: Linear fit was obtained using the curve_fit optimisation algorithm from the Scipy library.

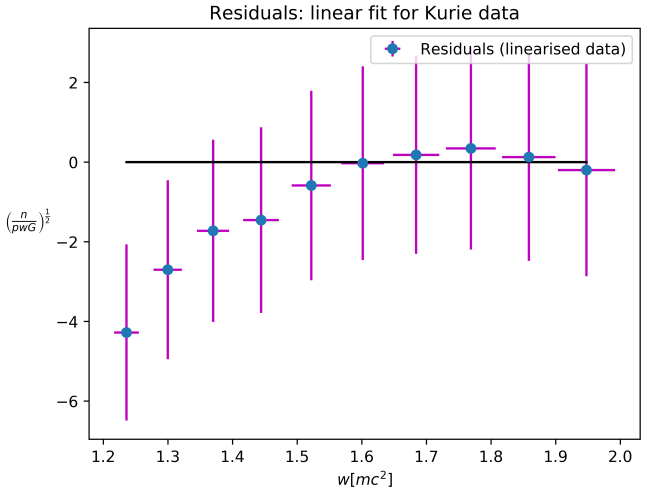

Figure 2: Linear fit residuals for optimised Kurie plot data.

In equation (14) we defined a zeroth order fit for the Kurie Plot. $S_0 = 1$ Using an optimised fitting algorithm we determine the optimal parameters for this linear fit. From these parameters and equation (15) we find $w_0 = 1.03 \pm 0.04$ MeV.

Using w_0 we calculate higher order approximations of shape factor S_n which in turn increases the order and accuracy of w_n. Finally we find that $T = 0.514 \pm 0.05$ MeV. This value is $\sigma = 0.04$ away from true result $T = 0.512$ MeV.

5 Discussion

The dominant problem in our analysis is the process by which we obtained the value of the proportionality constant k. It unlikely that we have found the true value I_k. The value of I_k was simply set to the corresponding current measured for the apparent conversion peak. Since there are only 2 data points visible in our conversion peak a better method would have been to create a lorentzian fit for the data. Unfortunately we were not able to write code and perform this fit adequately so the naive method remained.

Our experimental execution and the theoretical framework of *beta*-decay we use in this report ignores the mass of the neutrino. We hypothesize that the fact that the electrons are emitted with specific momentum ranges is likely closely related to the varying mass of emitted neutrinos.

There are three neutrino flavor states and three discrete neutrino masses with different values (they do not correspond uniquely to the three flavors). A neutrino created with a specific flavor has an associated specific quantum superposition of all three mass states[9].

It is presently known that neutrinos oscillate between different flavors during flight. This oscillation occurs because the three mass state components of the produced flavor travel at different speeds. This means that relative phase shifts between their wave packets develop. Their interaction with the detectors determines how their wave packets combine to produce a varying superposition of three flavors[9].

6 Conclusion

Using a magnetic spectrometer we measure the energy spectrum of electrons emitted as β radiation from a ^{137}Cs source. Our investigation focuses in determining the relationships between the spectrometer's lens current, electrons' momenta and their energy. We find that the value of the kinetic energy of the nuclear transition is $T = 0.514 \pm 0.05$ MeV which is in agreement with the accepted value of $T = 0.512$ MeV.

References

[1] Monash SPA. 4.4 measurement of β-ray spectra, 2020.

[2] Wikipedia. Beta particle, 2020.

[3] C A Moyer R A Serway, C J Moses. *Modern Physics, third edition.* 2005.

[4] Carl R Nave. Beta radioactivity, 2001.

[5] D Halliday EA Quade. A magnetic lens beta-and gamma-ray spectrometer, 1947.

[6] K Siegbahn. *Beta Ray Spectrometer Theory and Design.* 1968.

[7] Wikipedia. Rigidity (electromagnetism), 2019.

[8] University of Michigan. Chapter 15, course: Elements of nuclear engineering and radiological science, 2010.

[9] Wikipedia. Neutrino, 2020.

09/08/20 08:50:21 /home/ana/Documents/uni/PHS3000/code/betarays.py

```
  1  # PHS3000
  2  # Betarays - Radioactive decay of Cs - 137
  3  # Ana Fabela, 08/09/2020
  4  import monashspa.PHS3000 as spa
  5  from scipy.interpolate import interp1d
  6  import numpy as np
  7  import pandas as pd
  8  import pytz
  9  import matplotlib
 10  import matplotlib.pyplot as plt
 11  from pprint import pprint
 12  import scipy.optimize
 13
 14  plt.rcParams['figure.dpi'] = 150
 15
 16  # Globals
 17  hbar = 1.0545718e-34 # [Js]
 18  c = 299792458 # [m/s]
 19  mass_e = 9.10938356e-31 # [kg]
 20  eV = 1.602176634e-19 # [J]
 21  MeV = 1e6 * eV
 22  keV = 1e3 * eV
 23  rel_energy_unit = mass_e * c**2 # to convert SI into relativistic or viceversa
 24
 25
 26  # Desintegration energy
 27  # Cs-137 disintegrates by beta minus emission to the excited state of Ba-137 (94.6 %)
 28  theory_T = 0.5120 * MeV
 29  theory_T_rel = theory_T / rel_energy_unit
 30  theory_w_0_rel =  theory_T_rel + 1
 31  p_0_rel = np.sqrt(theory_w_0_rel**2 - 1)
 32
 33  data = spa.betaray.read_data(r'beta-ray_data.csv')
 34
 35  def csv(data_file):
 36      # extracting valid data from csv file
 37      j = 0
 38      for row in data:
 39          # print(f'{j=}')
 40          if row[0] == pd.Timestamp('2020-08-18 10:26:00+10:00',
    tz=pytz.FixedOffset(360)):
 41              valid_data = data[j:]
 42              continue
 43          j+=1
 44
 45      background_count_data = []
 46      count = []
 47      lens_current = []
 48      u_lens_current = []
 49      for row in valid_data:
 50          if row[3] == 'Closed':
 51              # print(row[3])
 52              background_count_data.append(row)
 53              continue
 54          count.append(row[5])
 55          lens_current.append(row[6])
 56          u_lens_current.append(row[7])
 57
 58      # make lens current an np.array
 59      lens_current = np.array(lens_current)
 60      return background_count_data, count, lens_current, u_lens_current
 61
 62
 63  def correct_count(background_count_data):
```

```
    # correcting our data by removing avg background count and adjusting it for spectrometer resolution (3%)
    background_count = []
    for row in background_count_data:
        background_count.append(row[5])
    avg_background_count = np.mean(background_count)
    # print(f"We want to subtract the background count from our data {avg_background_count=}")
    # calculating fractional uncertainty in total background count (delta_t = 24 min)
    total_background = np.sum(background_count)
    u_avg_background_count = np.sqrt(total_background) / 4

    # uncertainty in the corrected count
    background_corrected_count = count - avg_background_count

    ###############################################################################
    # I chose the uncertainty in the count to be 15 counts
    u_background_corrected_count = np.sqrt(15**2 + u_avg_background_count**2)
    ###############################################################################

    # As per Siegbahn [9] correction for spectrometers resolution
    correct_count = background_corrected_count / lens_current
    u_correct_count = correct_count * np.sqrt((u_background_corrected_count / background_corrected_count)**2 + (u_lens_current / lens_current)**2)

    # print(f'\n{total_background=:.0f}')
    # print(f'{avg_background_count=:.0f}')
    # print(f'{u_avg_background_count=:.0f}')

    # print(f'\ncorrect counts:{correct_count[8:18]}')
    # print(f'uncertainty:{u_correct_count[8:18]}')

    return correct_count, u_correct_count

def compute_k(lens_current):
    # Finding constant of proportionality in p = kI
    # calibration peak (K) index of k peak is i=20
    T_K = 624.21 * keV / rel_energy_unit
    I_k = lens_current[20]
    k = np.sqrt((T_K + 1)**2 - 1) / I_k
    # defining appropriate uncertainty for our k peak
    u_I_k = 0.1 / 2
    u_k = k * (u_I_k / lens_current[20])
    # print(f'{T_K=:.3f} mc^2')
    # print(f'{k=:.3f}')
    # print(f'{u_I_k=:.3f}')
    # print(f'{u_k=:.3f}')
    return k, u_k

def compute_p_rel(lens_current, k, u_k):
    # The momentum spectrum (relativistic units)
    p_rel = k * lens_current
    u_p_rel = p_rel * np.sqrt((u_k / k)**2 + (0.0005 / lens_current)**2)
    dp_rel = p_rel[1]-p_rel[0]
    # print(f'\np :{p_rel[8:18]}')
    # print(f'uncertainty:{u_p_rel[8:18]}')
    # print(f'dp :{dp_rel}')
    return p_rel, u_p_rel, dp_rel

def interpolated_fermi(p_rel):
    # G = (p_rel * F(z=55, ,w_rel)) / w_rel
    fermi_data = spa.betaray.modified_fermi_function_data
    return interp1d(fermi_data[:,0], fermi_data[:,1], kind='cubic')(p_rel)

def compute_w(p_rel):
    # KURIE/Fermi PLOT
```

```
    w_rel = np.sqrt(p_rel**2 + 1) # relativistic energy units
    u_w_rel = u_p_rel[8:18]
    # print(f'\n{w_rel=}')
    # print(f'{u_w_rel=}')
    return w_rel, u_w_rel


def f(x, m, c):
    # linear model for optimize.curve_fit()
    return m * x + c

def compute_S_n(x, opt_w_n, u_opt_w_n):
    # shape factor from Siegbahn
    S_n = x**2 - 1 + (opt_w_n - x)**2
    u_S_n = np.sqrt((2 * u_x * x)**2 + (2 * np.sqrt(u_opt_w_n**2 + u_x**2) * (opt_w_n - x))**2)
    return S_n, u_S_n

def LHS(S_n, u_S_n):
    # left hand side of our linearised relation
    y = np.sqrt(correct_count[8:18] / (p_rel[8:18] * x * interpolated_fermi(p_rel[8:18]) * S_n))
    u_y = (y / 2) * np.sqrt((u_correct_count[8:18] / correct_count[8:18])**2 + (2 * (u_p_rel[8:18] / p_rel[8:18])**2) + (u_interpolated_fermi / interpolated_fermi(p_rel[8:18]))**2 + (u_S_n / S_n)**2)
    return y, u_y

def optimal_fit(f, x, y, u_y):
    # linear fit
    # unpack into popt, pcov
    popt, pcov = scipy.optimize.curve_fit(f, x, y, sigma=u_y, absolute_sigma=False)
    # To compute one standard deviation errors on the parameters use
    perr = np.sqrt(np.diag(pcov))

    # optimal parameters
    opt_K_2, opt_intercept = popt
    u_opt_K_2, u_opt_intercept = perr
    # print(f"\noptimised gradient {opt_K_2:.3f} ± {u_opt_K_2:.3f}")
    # print(f"optimised intercept {opt_intercept:.3f} ± {u_opt_intercept:.3f}")

    optimised_fit = f(x, opt_K_2, opt_intercept)
    # uncertainty in linear model f given optimal fit
    u_f = np.sqrt((x * u_opt_K_2)**2 + (u_opt_intercept)**2)
    # return optimal parameters
    return opt_K_2, opt_intercept, u_opt_K_2, u_opt_intercept, optimised_fit, u_f

def iterative_solve(x, w_n, u_w_n):
    # using our results to find T
    T = (w_n - 1) * rel_energy_unit

    # print("\nHenlo, this is the start of the while loop")
    while True:
        old_T = T
        S_n, u_S_n = compute_S_n(x, w_n, u_w_n)
        yn, u_yn = LHS(S_n, u_S_n)
        K_2, intercept, u_K_2, u_intercept, optimised_fit, u_f = optimal_fit(f, x, yn, u_yn)

        # using our results to find new w_n
        w_n = intercept / - K_2
        u_w_n = np.sqrt((u_K_2 / K_2)**2 + (u_intercept / intercept)**2) * w_n

        # new T in SI units
        T = (w_n - 1) * rel_energy_unit

        # print(f"T = {T / MeV} MeV")
```

```
        # print(f"old_T = {old_T / MeV} MeV\n")

        if abs(T - old_T) < 1e-10 * MeV:
            break
    # print("\nthis is the end of the while loop, yay bai.")

    u_T = (w_n - 1) * u_w_n / w_n * rel_energy_unit
    return T, u_T, yn, u_yn, optimised_fit, u_f

def compare(T, u_T):
    # comparison to theory
    diff = 0.512 * MeV - T
    how_many_sigmas = diff / u_T
    print(f"\nEXPECTED RESULT T = {theory_T / MeV :.3f} MeV")
    print(f"(optimised) T = {T / MeV:.3f} ± {u_T / MeV:.2f} MeV")
    # print(f"difference {diff:.3f}")
    print(f"number of σ away from true result: {abs(how_many_sigmas):.3f}")



############################ Calling our functions ############################

# open, read and dissect data file
background_count_data, count, lens_current, u_lens_current = csv(data)

# find constant k
k, u_k = compute_k(lens_current)

# correct background count (accounting for background and resolution (3%))
correct_count, u_correct_count = correct_count(background_count_data)

# find momentum spectrum
p_rel, u_p_rel, dp_rel = compute_p_rel(lens_current, k, u_k)




# our sliced data linearised
x, u_x = compute_w(p_rel[8:18])

# uncertainty in interpolated fermi
u_interpolated_fermi = np.sqrt((u_p_rel[8:18] / p_rel[8:18])**2 + (u_x / x)**2) *
interpolated_fermi(p_rel[8:18])

# LINEARISED KURIE WITH RESOLUTION CORRECTION
y = np.sqrt(correct_count[8:18] / (p_rel[8:18] * x * interpolated_fermi(p_rel[8:18])))
u_y = (y / 2) * np.sqrt((u_correct_count[8:18] / correct_count[8:18].clip(min=1))**2 +
(2 * (u_p_rel[8:18] / p_rel[8:18])**2) + (u_interpolated_fermi /
interpolated_fermi(p_rel[8:18]))**2)

# first order fit
opt_K_2, opt_intercept, u_opt_K_2, u_opt_intercept, optimised_fit, u_f = optimal_fit(f,
x, y, u_y)
# using our parameters to find opt_w_0
opt_w_0 = opt_intercept / - opt_K_2
u_opt_w_0 = np.sqrt((u_opt_K_2 / opt_K_2)**2 + (u_opt_intercept / opt_intercept)**2) *
opt_w_0
print(f" w_0 = {opt_w_0 * rel_energy_unit / MeV:.3f} MeV")
print(f"u(w_0) = {u_opt_w_0 * rel_energy_unit / MeV:.3f} MeV")

# ITERATIVE ANALYSIS using Shape factor (higher order fits)
T, u_T, yn, u_yn, optimised_fit, u_f = iterative_solve(x, opt_w_0, u_opt_w_0)

# final comparison to theoretical value T = 0.512 MeV
compare(T, u_T)
```

```
249
250
251
252
253 ############################ plots ############################
254
255 # OPTIMISED FIT PLOT and residuals plot
256 plt.figure()
257 plt.errorbar(
258             x, yn, xerr=u_p_rel[8:18], yerr=u_yn,
259             marker="None", linestyle="None", ecolor="m",
260             label=r"$y = (\frac{n}{p w G})^{\frac{1}{2}}$", color="g", barsabove=True
261 )
262 plt.plot(
263         x, optimised_fit, marker="None",
264         linestyle="-",
265         label="linear fit"
266 )
267 plt.fill_between(
268                 x, optimised_fit - u_f,
269                 optimised_fit + u_f,
270                 alpha=0.5,
271                 label="uncertainty in linear fit"
272 )
273 plt.title("linear fit for Kurie data")
274 plt.xlabel(r"$w [mc^{2}]$")
275 plt.ylabel(r"$\left ( \frac{n}{p w G} \right )^{\frac{1}{2}}$", rotation=0,
    labelpad=18)
276 plt.legend()
277 spa.savefig('Kurie_linear_data_plot.png')
278 # plt.show()
279
280 residuals = optimised_fit - yn
281 plt.figure()
282 plt.errorbar(
283             x, residuals, xerr=u_p_rel[8:18], yerr=u_f,
284             marker="o", ecolor="m", linestyle="None",
285             label="Residuals (linearised data)"
286 )
287 plt.plot([x[0], x[-1]], [0,0], color="k")
288 plt.title("Residuals: linear fit for Kurie data")
289 plt.xlabel(r"$w [mc^{2}]$")
290 plt.ylabel(r"$\left ( \frac{n}{p w G} \right )^{\frac{1}{2}}$", rotation=0,
    labelpad=18)
291 plt.legend()
292 spa.savefig('linear_residuals_Kurie_linear_data.png')
293 # plt.show()
294 ############################ plots ############################
```