```python
# PHS3000 - LOGBOOK1
# Betarays - radioactive decay Cs - 137
# Ana Fabela, 15/08/2020
import monashspa.PHS3000 as spa
from scipy.interpolate import interp1d
import numpy as np
import pandas as pd
import pytz
import matplotlib
import matplotlib.pyplot as plt
from pprint import pprint
import scipy.optimize

plt.rcParams['figure.dpi'] = 150

# Globals
c = 299792458 # [m/s]
mass_e = 9.10938356e-31 # [kg]
eV = 1.602176634e-19 # [J]
MeV = 1e6 * eV
keV = 1e3 * eV
rel_energy_unit = mass_e * c**2 # to convert SI into relativistic or viceversa

data = spa.betaray.read_data(r'beta-ray_data.csv')

# valid data slicing from csv file
j = 0
for row in data:
    # print(f'{j=}')
    if row[0] == pd.Timestamp('2020-08-18 10:26:00+10:00', tz=pytz.FixedOffset(360)):
        valid_data = data[j:]
        continue
    j+=1

background_count_data = []
count = []
lens_current = []
u_lens_current = []
for row in valid_data:
    if row[3] == 'Closed':
        # print(row[3])
        background_count_data.append(row)
        continue
    count.append(row[5])
    lens_current.append(row[6])
    u_lens_current.append(row[7])

background_count = []
# correcting our data by removing avg 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 substract thie 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
corrected_count = count - avg_background_count
u_corrected_count = np.sqrt(count + u_avg_background_count**2)


# 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
k = np.sqrt((T_K + 1)**2 - 1) / lens_current[20]
# print(f"{k=}")
u_k = k * (0.0005 / lens_current[20])
# print(f"absolute uncertainty: {u_k = }")
# print(f"fractional uncertainty: {(u_k / k) = }\n")

# The momentum spectrum
lens_current = np.array(lens_current)
p_rel = k * lens_current
u_p_rel = p_rel * np.sqrt((u_k / k)**2 + (0.0005 / lens_current)**2)
# print(f"absolute uncertainty u(p_rel):\n {u_p_rel}")
# print(f"fractional uncertainty u(p_rel) / p_rel:\n {(u_p_rel / p_rel)}")

# plot
plt.figure()
plt.errorbar(
            p_rel, corrected_count, xerr=u_p_rel, yerr=u_corrected_count,
            marker="None", ecolor="m", label=r"$n(p)_{corrected}$", color="g", barsabove=True
)

plt.title(r"$\beta^{-}$ particle momentum spectrum")
plt.xlabel("p [mc]")
plt.ylabel("n(p)")
plt.legend()
spa.savefig('count_vs_momentum_no_background_error.png')
plt.show()

####################### KURIE/Fermi PLOT #####################################

dp_rel = p_rel[1]-p_rel[0]

# getting interpolated!
fermi_data = spa.betaray.modified_fermi_function_data
interpolated_fermi = interp1d(fermi_data[:,0], fermi_data[:,1], kind='cubic')

######################## THEORETICAL ###############################
# Desintegration energy
# Cs-137 disintegrates by beta minus emission to the ground state of Ba-137 (5,6 %)
theory_w_0 = 1.174 * MeV
theory_w_0_rel = theory_w_0 / rel_energy_unit
p_0_rel = np.sqrt(theory_w_0_rel**2 - 1) / (mass_e * c)
# print(p_0_rel)

# # defining the theoretical count (Kuriefunction)
K_1 = 1 # ?
Sn = 1
def n(p_rel):
    w_rel = np.sqrt(p_rel**2 + 1) # relativistic energy units
    n = K_1 * Sn * (w_rel * interpolated_fermi(p_rel) / p_rel) * p_rel**2 * (theory_w_0_rel - w_rel)**2
    return n, w_rel


n_p_rel, w_rel = n(p_rel[:22]) #  call and unpack n(p)

# equation (3) in script
N = n_p_rel * dp_rel

# plot
plt.figure()
plt.plot(
        p_rel[:22], N, marker="None",
        linestyle="-"
)
plt.title("Kurie relation")
plt.xlabel("p [mc]")
plt.ylabel("n(p)dp")
spa.savefig('Kurie_plot.png')
plt.show()

######################## THEORETICAL ###############################
####################### EXPERIMENTAL ###############################

# plot
plt.figure()
plt.errorbar(
            p_rel[:23], corrected_count[:23], xerr=u_p_rel[:23], yerr=u_corrected_count[:23],
            marker="None", ecolor="m", label=r"$n(p)_{corrected}$", color="g", barsabove=True
)

plt.title(r"$\beta^{-}$ particle momentum spectrum")
plt.xlabel("p [mc]")
plt.ylabel("n(p)")
plt.legend()
spa.savefig('count_vs_momentum_no_background_error.png')
```

```python
plt.show()

###################### EXPERIMENTAL ###############################
###################### KURIE/Fermi PLOT ####################################
######################### Linear fit ###########################
# initial slice [:23]
# second slice [8:18]
n_p_rel, w_rel = n(p_rel[8:18])

# our sliced data linearised
x = w_rel
u_x = u_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])

# this clips negative counts which are non physical
corrected_count = corrected_count.clip(min=0)

# LINEARISED KURIE
y = np.sqrt(corrected_count[8:18] / (p_rel[8:18] * x * interpolated_fermi(p_rel[8:18])))
# regularising y to avoid zero u_y
y_regularised = np.sqrt(corrected_count[8:18].clip(min=1) / (p_rel[8:18] * x * interpolated_fermi(p_rel[8:18])))
u_y = (y_regularised / 2) * np.sqrt((u_corrected_count[8:18] / corrected_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[8:1

fit_results = spa.linear_fit(x, y, u_y=u_y)
# making our linear fit with one sigma uncertainty
y_fit = fit_results.best_fit
u_y_fit = fit_results.eval_uncertainty(sigma=1)

# calculating values from fit results
fit_parameters = spa.get_fit_parameters(fit_results)
# print(f"{fit_parameters=}")

# using our results to find w_0
K_2 = - fit_parameters["slope"]
u_K_2 = fit_parameters["u_slope"]
intercept = fit_parameters["intercept"]
u_intercept = fit_parameters["u_intercept"]
w_0 = intercept / K_2
u_w_0 = np.sqrt((u_K_2 / K_2)**2 + (u_intercept / intercept)**2) * w_0

print(f"linear fit gradient: {K_2 = }")
print(f"linear fit intercept: {intercept = }\n")

print(f"EXPECTED RESULT {theory_w_0_rel = }")
# pre-optimisation result
print(f"pre-optimisation result  {w_0 = } ± {u_w_0}\n")

# plot
plt.figure()
plt.errorbar(
            x, y, xerr=u_p_rel[8:18], yerr=u_y,
            marker="None", linestyle="None", ecolor="m",
            label=r"$y = (\frac{n}{p w G})^{\frac{1}{2}}$", color="g", barsabove=True
)
plt.plot(
        x, y_fit, marker="None",
        linestyle="-",
        label="linear fit"
)
plt.fill_between(
                x, y_fit - u_y_fit,
                y_fit + u_y_fit,
                alpha=0.5,
                label="uncertainty in linear fit"
)
plt.title("Linearised Kurie data")
plt.xlabel(r"$w [mc^{2}]$")
plt.ylabel(r"$\left ( \frac{n}{p w G} \right )^{\frac{1}{2}}$", rotation=0, labelpad=18)
plt.legend()
spa.savefig('Kurie_linear_data_plot_.png')
plt.show()

############################ Linear fit ############################
#########################Linear fit residuals#######################

linear_residuals = y_fit - y # linear residuals (linear best fit - linearised data)

# plot
plt.figure()
plt.errorbar(
            x, linear_residuals, xerr=u_p_rel[8:18], yerr=u_y,
            marker="o", ecolor="m", linestyle="None",
            label="Residuals (linearised data)"
)
plt.plot([x[0], x[-1]], [0,0], color="k")
plt.title("Residuals: linearised Kurie data")
plt.xlabel(r"$w [mc^{2}]$")
plt.ylabel(r"$\left ( \frac{n}{p w G} \right )^{\frac{1}{2}}$", rotation=0, labelpad=18)
plt.legend()
spa.savefig('linear_residuals_Kurie_linear_data.png')
plt.show()

# #########################Linear fit residuals#######################

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

# optimising our 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))

opt_K_2, opt_intercept = popt
u_opt_K_2, u_opt_intercept = perr

print(f"optimised gradient {opt_K_2} ± {u_opt_K_2}")
print(f"optimised intercept {opt_intercept} ± {u_opt_intercept}\n")

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

# using our results 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"EXPECTED RESULT {theory_w_0_rel = }")
print(f"post-optimisation result  {opt_w_0 = } ± {u_opt_w_0}\n")
print(f"non-relativistic w_0 = {opt_w_0 * rel_energy_unit / MeV} ± {u_opt_w_0 * rel_energy_unit / MeV}\n")

# OPTIMISED FIT PLOT
plt.figure()
plt.errorbar(
            x, y, xerr=u_p_rel[8:18], yerr=u_y,
            marker="None", linestyle="None", ecolor="m",
            label=r"$y = (\frac{n}{p w G})^{\frac{1}{2}}$", color="g", barsabove=True
)
plt.plot(
        x, optimised_fit, marker="None",
        linestyle="-",
        label="linear fit"
)
plt.fill_between(
                x, optimised_fit - u_f,
                optimised_fit + u_f,
                alpha=0.5,
                label="uncertainty in linear fit"
)
plt.title("Optimised linear fit for Kurie data")
plt.xlabel(r"$w [mc^{2}]$")
plt.ylabel(r"$\left ( \frac{n}{p w G} \right )^{\frac{1}{2}}$", rotation=0, labelpad=18)
plt.legend()
spa.savefig('OPTIMISED_Kurie_linear_data_plot_.png')
plt.show()

#########################optimised fit residuals#######################
```

```python

optimised_residuals = optimised_fit - y
# plot
plt.figure()
plt.errorbar(
            x, optimised_residuals, xerr=u_p_rel[8:18], yerr=u_f,
            marker="o", ecolor="m", linestyle="None",
            label="Residuals (linearised data)"
)
plt.plot([x[0], x[-1]], [0,0], color="k")
plt.title("Residuals: optimised fit for linear Kurie data")
plt.xlabel(r"$w [mc^{2}]$")
plt.ylabel(r"$\left ( \frac{n}{p w G} \right )^{\frac{1}{2}}$", rotation=0, labelpad=18)
plt.legend()
spa.savefig('OPTIMISED_linear_residuals_Kurie_linear_data.png')
plt.show()

# # ##########################optimised fit residuals########################
```