

```

1 # FHS3000 - LOGBOOK1
2 # Betarays - radioactive decay Cs - 137
3 # Ana Fabela, 15/08/2020
4 import monashpa.FHS3000 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 c = 299792458 # [m/s]
18 mass_e = 9.10938356e-31 # [kg]
19 eV = 1.602176634e-19 # [J]
20 MeV = 1e6 * eV
21 keV = 1e3 * eV
22 rel_energy_unit = mass_e * c**2 # to convert SI into relativistic or viceversa
23
24 data = spa.betaray.read_data(r'beta-ray_data.csv')
25
26 # valid data slicing from csv file
27 j = 0
28 for row in data:
29     # print(f'{j=}')
30     if row[0] == pd.Timestamp('2020-08-18 10:26:00+10:00', tz=pytz.FixedOffset(360)):
31         valid_data = data[j:]
32         continue
33     j+=1
34
35 background_count_data = []
36 count = []
37 lens_current = []
38 u_lens_current = []
39 for row in valid_data:
40     if row[3] == 'Closed':
41         # print(row[3])
42         background_count_data.append(row)
43         continue
44     count.append(row[5])
45     lens_current.append(row[6])
46     u_lens_current.append(row[7])
47
48 background_count = []
49 # correcting our data by removing avg background count
50 for row in background_count_data:
51     background_count.append(row[5])
52 avg_background_count = np.mean(background_count)
53 # print(f"We want to subtract the background count from our data (avg_background_count=)")
54 # calculating fractional uncertainty in total background count (delta_t = 24 min)
55 total_background = np.sum(background_count)
56 u_avg_background_count = np.sqrt(total_background) / 4
57
58 # uncertainty in the corrected count
59 corrected_count = count - avg_background_count
60 u_corrected_count = np.sqrt(count + u_avg_background_count**2)
61
62
63 # Finding constant of proportionality in  $p = kI$ 
64 # calibration peak (K) index of k peak is i=20
65 T_K = 624.21 * keV / rel_energy_unit
66 k = np.sqrt((T_K + 1)**2 - 1) / lens_current[20]
67 # print(f"(k=)")
68 u_k = k * (0.0005 / lens_current[20])
69 # print(f"absolute uncertainty: {u_k =}")
70 # print(f"fractional uncertainty: {(u_k / k) = }\n")
71
72 # The momentum spectrum
73 lens_current = np.array(lens_current)
74 p_rel = k * lens_current
75 u_p_rel = p_rel * np.sqrt((u_k / k)**2 + (0.0005 / lens_current)**2)
76 # print(f"absolute uncertainty u(p_rel):\n {u_p_rel}")
77 # print(f"fractional uncertainty u(p_rel) / p_rel:\n {(u_p_rel / p_rel)}")
78
79 # plot
80 plt.figure()
81 plt.errorbar(
82     p_rel, corrected_count, xerr=u_p_rel, yerr=u_corrected_count,
83     marker="None", ecolor="m", label=r"$n(p)_{corrected}$", color="g", barsabove=True
84 )
85
86 plt.title(r"$\beta^-$ particle momentum spectrum")
87 plt.xlabel("p [mc]")
88 plt.ylabel("n(p)")
89 plt.legend()
90 spa.savefig('count_vs_momentum_no_background_error.png')
91 plt.show()
92
93 ##### KURIE/Fermi PLOT #####
94
95 dp_rel = p_rel[1]-p_rel[0]
96
97 # getting interpolated!
98 fermi_data = spa.betaray.modified_fermi_function_data
99 interpolated_fermi = interp1d(fermi_data[:,0], fermi_data[:,1], kind='cubic')
100
101 ##### THEORETICAL #####
102 # Desintegration energy
103 # Cs-137 disintegrates by beta minus emission to the ground state of Ba-137 (5,6 %)
104 theory_w_0 = 1.174 * MeV
105 theory_w_0_rel = theory_w_0 / rel_energy_unit
106 p_0_rel = np.sqrt(theory_w_0_rel**2 - 1) / (mass_e * c)
107 # print(p_0_rel)
108
109 # # defining the theoretical count (Kuriefunction)
110 K_1 = 1 # ?
111 Sn = 1
112 def n(p_rel):
113     w_rel = np.sqrt(p_rel**2 + 1) # relativistic energy units
114     n = K_1 * Sn * (w_rel + interpolated_fermi(p_rel) / p_rel) * p_rel**2 * (theory_w_0_rel - w_rel)**2
115     return n, w_rel
116
117
118 n_p_rel, w_rel = n(p_rel[:22]) # call and unpack n(p)
119
120 # equation (3) in script
121 N = n_p_rel * dp_rel
122
123 # plot
124 plt.figure()
125 plt.plot(
126     p_rel[:22], N, marker="None",
127     linestyle="-"
128 )
129 plt.title("Kurie relation")
130 plt.xlabel("p [mc]")
131 plt.ylabel("n(p)dp")
132 spa.savefig('Kurie_plot.png')
133 plt.show()
134
135 ##### THEORETICAL #####
136 ##### EXPERIMENTAL #####
137
138 # plot
139 plt.figure()
140 plt.errorbar(
141     p_rel[:23], corrected_count[:23], xerr=u_p_rel[:23], yerr=u_corrected_count[:23],
142     marker="None", ecolor="m", label=r"$n(p)_{corrected}$", color="g", barsabove=True
143 )
144
145 plt.title(r"$\beta^-$ particle momentum spectrum")
146 plt.xlabel("p [mc]")
147 plt.ylabel("n(p)")
148 plt.legend()
149 spa.savefig('count_vs_momentum_no_background_error.png')

```

```

150 plt.show()
151
152 ##### EXPERIMENTAL #####
153 ##### KURIE/Fermi PLOT #####
154 ##### Linear fit #####
155 # initial slice [:23]
156 # second slice [8:18]
157 n_p_rel, w_rel = n(p_rel[8:18])
158
159 # our sliced data linearised
160 x = w_rel
161 u_x = u_p_rel[8:18]
162
163 # uncertainty in interpolated fermi
164 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])
165
166 # this clips negative counts which are non physical
167 corrected_count = corrected_count.clip(min=0)
168
169 # LINEARISED KURIE
170 y = np.sqrt(corrected_count[8:18] / (p_rel[8:18] * x * interpolated_fermi(p_rel[8:18])))
171 # regularising y to avoid zero u_y
172 y_regularised = np.sqrt(corrected_count[8:18].clip(min=1) / (p_rel[8:18] * x * interpolated_fermi(p_rel[8:18])))
173 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[8:18]))**2)
174
175 fit_results = spa.linear_fit(x, y, u_y=u_y)
176 # making our linear fit with one sigma uncertainty
177 y_fit = fit_results.best_fit
178 u_y_fit = fit_results.eval_uncertainty(sigma=1)
179
180 # calculating values from fit results
181 fit_parameters = spa.get_fit_parameters(fit_results)
182 # print(f"fit_parameters={}")
183
184 # using our results to find w_0
185 K_2 = - fit_parameters["slope"]
186 u_K_2 = fit_parameters["u_slope"]
187 intercept = fit_parameters["intercept"]
188 u_intercept = fit_parameters["u_intercept"]
189 w_0 = intercept / K_2
190 u_w_0 = np.sqrt((u_K_2 / K_2)**2 + (u_intercept / intercept)**2) * w_0
191
192 print(f"linear fit gradient: (K_2 = )")
193 print(f"linear fit intercept: (intercept = )\n")
194
195 print(f"EXPECTED RESULT (theory_w_0_rel = )")
196 # pre-optimisation result
197 print(f"pre-optimisation result (w_0 = ) ± (u_w_0)\n")
198
199 # plot
200 plt.figure()
201 plt.errorbar(
202     x, y, xerr=u_p_rel[8:18], yerr=u_y,
203     marker="None", linestyle="None", ecolor="m",
204     label=r"$y = (\frac{n}{p \cdot w \cdot G})^{1/2}$", color="g", barsabove=True
205 )
206 plt.plot(
207     x, y_fit, marker="None",
208     linestyle="--",
209     label="linear fit"
210 )
211 plt.fill_between(
212     x, y_fit - u_y_fit,
213     y_fit + u_y_fit,
214     alpha=0.5,
215     label="uncertainty in linear fit"
216 )
217 plt.title("Linearised Kurie data")
218 plt.xlabel(r"$w$ [mc2])")
219 plt.ylabel(r"$\left( \frac{n}{p \cdot w \cdot G} \right)^{1/2}$", rotation=0, labelpad=18)
220 plt.legend()
221 spa.savefig('Kurie_linear_data_plot_.png')
222 plt.show()
223
224 ##### Linear fit #####
225 ##### Linear fit residuals #####
226
227 linear_residuals = y_fit - y # linear residuals (linear best fit - linearised data)
228
229 # plot
230 plt.figure()
231 plt.errorbar(
232     x, linear_residuals, xerr=u_p_rel[8:18], yerr=u_y,
233     marker="o", ecolor="m", linestyle="None",
234     label="Residuals (linearised data)"
235 )
236 plt.plot([x[0], x[-1]], [0,0], color="k")
237 plt.title("Residuals: linearised Kurie data")
238 plt.xlabel(r"$w$ [mc2])")
239 plt.ylabel(r"$\left( \frac{n}{p \cdot w \cdot G} \right)^{1/2}$", rotation=0, labelpad=18)
240 plt.legend()
241 spa.savefig('linear_residuals_Kurie_linear_data.png')
242 plt.show()
243
244 # ##### Linear fit residuals #####
245
246 # linear model for optimize.curve_fit()
247 def f(x, m, c):
248     return m * x + c
249
250 # optimising our fit, unpack into popt, pcov
251 popt, pcov = scipy.optimize.curve_fit(f, x, y, sigma=u_y, absolute_sigma=False)
252 # To compute one standard deviation errors on the parameters use
253 perr = np.sqrt(np.diag(pcov))
254
255 opt_K_2, opt_intercept = popt
256 u_opt_K_2, u_opt_intercept = perr
257
258 print(f"Optimised gradient (opt_K_2) ± (u_opt_K_2)")
259 print(f"Optimised intercept (opt_intercept) ± (u_opt_intercept)\n")
260
261 optimised_fit = f(x, opt_K_2, opt_intercept)
262 # uncertainty in linear model f given optimal fit
263 u_f = np.sqrt((opt_K_2 * u_x)**2 + (x * u_opt_K_2)**2 + (u_opt_intercept)**2)
264
265 # using our results to find opt_w_0
266 opt_w_0 = opt_intercept / - opt_K_2
267 u_opt_w_0 = np.sqrt((u_opt_K_2 / opt_K_2)**2 + (u_opt_intercept / opt_intercept)**2) * opt_w_0
268
269 print(f"EXPECTED RESULT (theory_w_0_rel = )")
270 print(f"post-optimisation result (opt_w_0 = ) ± (u_opt_w_0)\n")
271 print(f"non-relativistic w_0 = (opt_w_0 * rel_energy_unit / MeV) ± (u_opt_w_0 * rel_energy_unit / MeV)\n")
272
273 # OPTIMISED FIT PLOT
274 plt.figure()
275 plt.errorbar(
276     x, y, xerr=u_p_rel[8:18], yerr=u_y,
277     marker="None", linestyle="None", ecolor="m",
278     label=r"$y = (\frac{n}{p \cdot w \cdot G})^{1/2}$", color="g", barsabove=True
279 )
280 plt.plot(
281     x, optimised_fit, marker="None",
282     linestyle="--",
283     label="linear fit"
284 )
285 plt.fill_between(
286     x, optimised_fit - u_f,
287     optimised_fit + u_f,
288     alpha=0.5,
289     label="uncertainty in linear fit"
290 )
291 plt.title("Optimised linear fit for Kurie data")
292 plt.xlabel(r"$w$ [mc2])")
293 plt.ylabel(r"$\left( \frac{n}{p \cdot w \cdot G} \right)^{1/2}$", rotation=0, labelpad=18)
294 plt.legend()
295 spa.savefig('OPTIMISED_Kurie_linear_data_plot_.png')
296 plt.show()
297
298 ##### Optimised fit residuals #####

```

```

299
300 optimised_residuals = optimised_fit - y
301 # plot
302 plt.figure()
303 plt.errorbar(
304     x, optimised_residuals, xerr=u_p_rel[8:18], yerr=u_f,
305     marker="o", ecolor="m", linestyle="None",
306     label="Residuals (linearised data)"
307 )
308 plt.plot([x[0], x[-1]], [0,0], color="k")
309 plt.title("Residuals: optimised fit for linear Kurie data")
310 plt.xlabel(r"$\mu$ [mc^2]")
311 plt.ylabel(r"$\left( \frac{n}{p w G} \right)^{\frac{1}{2}}$", rotation=0, labelpad=18)
312 plt.legend()
313 spa.savefig('OPTIMISED_linear_residuals_Kurie_linear_data.png')
314 plt.show()
315
316 # # #####optimised fit residuals#####

```