

Z Boson Lab Script

Yupeng Fan

October 2024

```
1  """
2  Title: PHYS20161 - Assignment 2: Z Boson
3
4
5  Main Goal:
6
7  This script is designed to analyze two particle collision data.
8      First, it need
9  to combine, validate the data sets. Second, it performs best fit
10     plot by
11     finding the minimum of the chi_square value. Meanwhile, this code
12     is able to
13     get the parameters value for best fit and life time for Z0, and
14     calculate the
15     uncertainties. Finally, this code also have several additional
16     features listed
17     below.
18
19 Additional Features:
20
21 1. Further validation on the cross_section datas.
22
23 2. Chi-Square Contour Plot: Provides a visualization of the
24    chi_square minimization process.
25
26 3. Extra 3D plot of the Contour Plot, optional output decided by
27    the user.
28
29 4. Validation whether the distribution is gaussian and the judge
30    the quality
31    of the fit: Estimate the geometric centre of the chi_square cntour
32    and
33    define a funtction to assess the quality of the fit.
34    (The user will see result clearly in output)
35
36 5. Plots of the Gaussian distribution for both the mass (mass_z0)
37    and the
38    partial width ( Z0 ).
39    (To decide whether it is gaussian is checked by No.4 addtional
40    features)
41
42 6. CSV Data Export Function: A function to save filtered data for
43    further use
```

```

33 | or record-keeping.
34 |
35 | 7. Interactive Menu: Allows users to customize the outputs based on
    | their
36 | preferences for further output options, enhancing user experience.
37 |
38 | Last updated: 13/12/2023
39 | @author: w23058yf
40 | """
41 |
42 | import matplotlib.pyplot as plt
43 | import numpy as np
44 | import scipy.constants as pc
45 | from scipy.optimize import fmin
46 |
47 | # Constants
48 | HBAR_GEV_S = pc.hbar / (1.602e-19 * 1e9) # Reduced Planck constant
    | in GeV*s
49 | PARTIAL_WIDTH_EE = 83.91 * 1e-3 # Convert MeV to GeV
50 |
51 | # File names
52 | FILE_NAME_1 = 'z_boson_data_1.csv'
53 | FILE_NAME_2 = 'z_boson_data_2.csv'
54 |
55 |
56 | def validate_data(data):
57 |     """
58 |     Validates the data by removing NaN values, zero or negative
    | values,
59 |
60 |     Args:
61 |     data (array): The data array.
62 |
63 |     Returns:
64 |     array: The validated and cleaned data array.
65 |     """
66 |
67 |     # Remove the NaN values
68 |     data = data[~np.isnan(data).any(axis=1)]
69 |
70 |     # Remove zero or negative values
71 |     data = data[np.all(data > 0, axis=1)]
72 |
73 |     return data
74 |
75 |
76 | def read_and_combine_data(file1, file2):
77 |     """
78 |     Reads data from two chi_squared_contour_plotV files, validates,
    | and
79 |     combines them into a single array.
80 |
81 |     Args:
82 |     file1 (string): The first data file.
83 |     file2 (string): The second data file.
84 |
85 |     Returns:

```

```

86         array: A combined and validated array with data from both files
87     ,
88     or None if an error occurs.
89     """
90     try:
91         data1 = np.genfromtxt(file1, delimiter=',', skip_header=1)
92         data2 = np.genfromtxt(file2, delimiter=',', skip_header=1)
93
94         data1 = validate_data(data1)
95         data2 = validate_data(data2)
96
97         # Check column consistency between the two data sets
98         if data1.shape[1] != data2.shape[1]:
99             raise ValueError(
100                 "Mismatch in the number of columns between datasets
101                 .")
102         # Combine two sets of data sets
103         combined_data = np.vstack((data1, data2))
104         return combined_data
105     except IOError as error:
106         print(f"File not found: {error.filename}")
107         return None
108     except ValueError as error:
109         print(f"Error reading file: {error}")
110         return None
111
112 def filter_data(data, num_std=3):
113     """
114     Filter data by removing outliers based on a specified number of
115     standard deviations.
116
117     Args:
118     data (array): The data array.
119     num_std (int): The number of standard deviations.
120
121     Returns:
122     array: The filtered data array.
123     """
124     # Calculate the mean of the cross-section values
125     mean_cross_section = np.mean(data[:, 1])
126
127     # Calculate the standard deviation of the cross-section values
128     std_dev_cross_section = np.std(data[:, 1])
129
130     # Filter out data points where the cross-section deviates from
131     # the mean by
132     # more than 'num_std' standard deviations.
133     return data[abs(data[:, 1] - mean_cross_section) <= num_std *
134                 std_dev_cross_section]
135
136 def min_chi_square(center_of_mass_energy, cross_section,
137                   uncertainty,
138                   initial_guesses):
139     """
140     Find the parameters that minimize the chi-square value.

```

```

139
140     Args:
141     center_of_mass_energy (array): Array of center-of-mass energies
142
143     cross_section (array): Array of cross sections.
144     uncertainty (array): Array of uncertainties in the cross
145         section
146     measurements.
147     initial_guesses (list): List of initial guesses for the mass
148         and partial
149     width of the Z0 boson.
150
151     Returns:
152     result(tuple)
153     """
154     result = fmin(lambda parameters: chi_square(parameters,
155                                                 center_of_mass_energy,
156                                                 cross_section,
157                                                 uncertainty),
158                 initial_guesses, full_output=True, disp=False)
159     return result
160
161 def filter_data_2(center_of_mass_energy, cross_section,
162                  mass_z0_fit, partial_width_z0_fit,
163                  filtered_data_1):
164     """
165     (This function is for No.1 additional feature)
166
167     Filter data based on predicted cross sections compared to
168     actual
169     cross sections.
170
171     Args:
172     center_of_mass_energy (array): Array of center-of-mass energies
173
174     cross_section (array): Array of measured cross sections.
175     uncertainty (array): Array of uncertainties
176     mass_z0_fit (float): Fitted Z0 boson mass.
177     partial_width_z0_fit (float): Fitted Z0 boson partial width.
178     filtered_data_1 (array): Previously filtered data array.
179
180     Returns:
181     array: Further filtered data array.
182     """
183     # Perdict the value of cross section by the formean_valuel.
184     perdict_cross_section = calculate_cross_section(
185         center_of_mass_energy, mass_z0_fit, partial_width_z0_fit)
186     # Get the standard ddeviation value of the cross section.
187     std_dev_cross_section = np.std(perdict_cross_section)
188     return filtered_data_1[np.abs(perdict_cross_section -
189                                 cross_section) <=
190                            0.5 * std_dev_cross_section]

```

```

187 def calculate_reduced_chi_square(chi_sq, num_data_points,
188                                   num_parameters):
189     """
190     Calculate the reduced chi-square value.
191
192     Args:
193     chi_sq (float): The chi-square value from the fit.
194     num_data_points (int): The number of data points used in the
195     fit.
196     num_parameters (int): The number of parameters.
197
198     Returns:
199     float: The reduced chi-square value.
200     """
201     degrees_of_freedom = num_data_points - num_parameters
202     return chi_sq / degrees_of_freedom
203
204 def calculate_cross_section(center_of_mass_energy, mass_z0,
205                             partial_width_z0):
206     """
207     Calculate the cross section for a given center-of-mass energy,
208     Z0 boson mass, and partial width.
209
210     Args:
211     center_of_mass_energy (array): Array of center-of-mass energies
212     .
213     mass_z0 (float): Z0 boson mass.
214     partial_width_z0 (float): Z0 boson partial width.
215
216     Returns:
217     array: Calculated cross section values.
218     """
219     # Define the given formula to calculate the cross section
220     numerator = 12 * np.pi * center_of_mass_energy**2 * (
221         PARTIAL_WIDTH_EE**2)
222     denominator = (center_of_mass_energy**2 - mass_z0**2)**2 + \
223         (mass_z0**2) * (partial_width_z0**2)
224     return numerator / (denominator * mass_z0**2) * 389400
225
226 def chi_square(parameters, center_of_mass_energy,
227                cross_section_data,
228                uncertainty):
229     """
230     Calculate the chi-square value for the fit.
231
232     Args:
233     parameters (list): List containing the mass and partial width
234     of the Z0
235     boson.
236     center_of_mass_energy (array): Array of center-of-mass
237     energies.
238     cross_section_data (array): Array of measured cross
239     sections.
240     uncertainty (array): Array of uncertainties

```

```

237 Returns:
238 float: The calculated chi-square value.
239 """
240 mass_z0, partial_width_z0 = parameters
241 # Get the prediction value of cross section
242 prediction = calculate_cross_section(center_of_mass_energy,
243                                     mass_z0,
244                                     partial_width_z0)
245 # Get the chi-square value
246 chi_sq = np.sum(((cross_section_data - prediction) /
247                  uncertainty) ** 2)
248 return chi_sq
249
250 def plot_data_and_fit(center_of_mass_energy, cross_section_data,
251                      uncertainty,
252                      mass_z0, partial_width_z0):
253     """
254     Plots the data points with error bars and the best fit curve.
255
256     Args:
257     center_of_mass_energy (array): Array of center-of-mass energies
258     cross_section_data (array): Array of measured cross sections.
259     uncertainty (array): Array of uncertainties
260     mass_z0 (float): Z boson mass parameter
261     partial_width_z0(float): Z boson partial width parameter
262
263     Returns:
264     None
265     """
266     plt.figure(figsize=(10, 6))
267     plt.errorbar(center_of_mass_energy, cross_section_data, yerr=
268                  uncertainty,
269                  fmt='o', label='Data', color='royalblue', ecolor='
270                  lightgray',
271                  elinewidth=3, capsize=0)
272
273     # Create a smooth line for the fit
274     center_of_mass_energy_fit = np.linspace(
275         min(center_of_mass_energy), max(center_of_mass_energy),
276         1000)
277     cross_section_fit = calculate_cross_section(
278         center_of_mass_energy_fit, mass_z0, partial_width_z0)
279
280     # Plot the fit
281     plt.plot(center_of_mass_energy_fit, cross_section_fit,
282              label='Fit', linewidth=2, color='darkorange')
283     plt.xlabel('Centre-of-mass energy (GeV)', fontsize=14)
284     plt.ylabel('Cross Section (nb)', fontsize=14)
285     plt.title('Fitted Curve with Scattered Data', fontsize=16)
286     plt.legend()
287     plt.grid(True)
288     plt.savefig('z_boson_fit.png')
289     plt.show()

```

```

287 def plot_chi_square_contours(center_of_mass_energy, cross_section,
288                               uncertainty,
289                               mass_z0_range, partial_width_z0_range,
290                               mass_z0_fit, partial_width_z0_fit):
291     """
292     (This function is for No.2 additional feature)
293
294     Provide visualization of chi-square for 2 parameters with fit
295     values,
296     and contour line which is helpful in future analysis.
297
298     Args:
299     center_of_mass_energy (array): Array of center-of-mass energies
300     .
301     cross_section (array): Array of measured cross sections.
302     uncertainty (array): Array of uncertainties in the cross
303     section
304     measurements.
305     mass_z0_range (tuple): Range for the Z0 boson mass range.
306     partial_width_z0_range (tuple): Range for the Z0 boson width
307     range.
308     mass_z0_fit (float): Fitted value of Z0 boson mass.
309     partial_width_z0_fit (float): Fitted value of Z0 boson width.
310
311     Returns:
312     None
313     """
314     mass_z0_values = np.linspace(*mass_z0_range, 100)
315     partial_width_z0_values = np.linspace(*partial_width_z0_range,
316     100)
317     mass_z0_grid, partial_width_z0_grid = np.meshgrid(
318     mass_z0_values, partial_width_z0_values)
319     chi_sq_grid = np.zeros_like(mass_z0_grid)
320
321     # Calculate chi-square values
322     for i in range(mass_z0_grid.shape[0]):
323         for j in range(partial_width_z0_grid.shape[1]):
324             chi_sq_grid[i, j] = chi_square([mass_z0_grid[i, j],
325             partial_width_z0_grid[i, j]],
326             center_of_mass_energy,
327             cross_section,
328             uncertainty)
329
330     # Find the minimum chi-square value
331     best_chi_square = np.min(chi_sq_grid)
332     level_1 = best_chi_square + 1
333     # Plot Contours
334     plt.contourf(mass_z0_grid, partial_width_z0_grid,
335     chi_sq_grid, levels=50, cmap='viridis')
336     plt.colorbar(label='Chi-squared')
337     # Add contour line
338     plt.contour(mass_z0_grid, partial_width_z0_grid,
339     chi_sq_grid, levels=[level_1],
340     colors='red', linestyle='dashed')
341     # Label for minimum chi-square + 1
342     plt.plot([], [], 'r--', label='Min chi-square + 1')
343     # Plotting the best fit values as a red dot

```

```

336 plt.plot(mass_z0_fit, partial_width_z0_fit, 'ro', label='
    Minimum Fit')
337 plt.axhline(y=partial_width_z0_fit, color='r', linestyle='--')
338 plt.axvline(x=mass_z0_fit, color='r', linestyle='--')
339 plt.xlabel('Z boson mass (mass_z0) [GeV/c^2]')
340 plt.ylabel('Z boson width (partial_width_z0) [GeV]')
341 plt.title('Chi-squared Contours')
342 plt.legend()
343 plt.savefig("chi_square_contours.png", format='png')
344 plt.show()
345
346
347 def plot_chi_square_3d_contours(center_of_mass_energy,
    cross_section,
348                                 uncertainty, mass_z0_range,
349                                 partial_width_z0_range):
350     """
351     (This function is for No.3 additional feature)
352
353     Plot a 3D contour of the chi-square values, which is helpful in
354     more vivid
355     visualisation for chisquare contour. This is an optional output
356     , which is
357     determined by user.
358
359     Args:
360     center_of_mass_energy (array): Array of center-of-mass energies
361     .
362     cross_section (array): Array of measured cross sections.
363     uncertainty (array): Array of uncertainties in the cross
364     section
365     measurements.
366     mass_z0_range (tuple): Range for the Z0 boson mass
367     range.
368     partial_width_z0_range (tuple): Range for the Z0
369     boson width range.
370     mass_z0_fit (float): Fitted value of Z0 boson mass.
371     partial_width_z0_fit (float): Fitted value of Z0 boson width.
372
373     Returns:
374     None
375     """
376     mass_z0_values = np.linspace(*mass_z0_range, 100)
377     partial_width_z0_values = np.linspace(*partial_width_z0_range,
378     100)
379     mass_z0_grid, partial_width_z0_grid = np.meshgrid(
380     mass_z0_values, partial_width_z0_values)
381     chi_sq_grid = np.zeros_like(mass_z0_grid)
382
383     # Calculate chi-square values
384     for i in range(mass_z0_grid.shape[0]):
385         for j in range(partial_width_z0_grid.shape[1]):
386             chi_sq_grid[i, j] = chi_square([mass_z0_grid[i, j],
387             partial_width_z0_grid[i, j]],
388             center_of_mass_energy,
389             cross_section,

```



```

uncertainty)
385 fig = plt.figure()
386 axes = fig.add_subplot(111, projection='3d')
387
388 # Plot a 3D contour
389 axes.contour3D(mass_z0_grid, partial_width_z0_grid,
390               chi_sq_grid, 50, cmap='viridis')
391
392 # Label the axes
393 axes.set_xlabel('Z boson mass (mass_z0) [GeV/c^2]')
394 axes.set_ylabel('Z boson width (partial_width_z0) [GeV]')
395 axes.set_zlabel('Chi-squared')
396 axes.legend()
397 # Save the plot in png version
398 plt.savefig("3D_chi_square_contours.png", format='png')
399 plt.show()
400
401
402 def find_uncertainty_hill_climbing(data, best_fit_parameters,
403                                   chi_squared_min, param_index,
404                                   delta_chi_square=1.0):
405     """
406     Estimate the uncertainty of the fit parameters using the hill-
407         climbing
408         method.
409
410     Args:
411     data (array): data array.
412     best_fit_parameters (list): List of the best fit parameters.
413     chi_squared_min (float): The minimum chi-square.
414     param_index (int): Index of the parameter to calculate the
415         uncertainty.
416     step_size (float): The step size
417     delta_chi_sq (float): The change in chi-square value.
418     Returns:
419     float: The uncertainties of two parameters.
420     """
421     # Set proper step size
422     step_size = 0.00001
423     parameter_plus = best_fit_parameters[param_index]
424     parameter_minus = best_fit_parameters[param_index]
425     chi_squared_plus = chi_squared_min
426     chi_squared_minus = chi_squared_min
427     # Apply hill climbing method from two directions
428     while chi_squared_plus - chi_squared_min < delta_chi_square:
429         parameter_plus += step_size
430         parameters = best_fit_parameters.copy()
431         parameters[param_index] = parameter_plus
432         chi_squared_plus = chi_square(
433             parameters, data[:, 0], data[:, 1], data[:, 2])
434
435     while chi_squared_minus - chi_squared_min < delta_chi_square:
436         parameter_minus -= step_size
437         parameters = best_fit_parameters.copy()
438         parameters[param_index] = parameter_minus
439         chi_squared_minus = chi_square(
440             parameters, data[:, 0], data[:, 1], data[:, 2])

```

```

439     uncertainty = (parameter_plus - parameter_minus) / 2
440     return uncertainty
441
442
443 def calculate_lifetime(partial_width_z0):
444     """
445     Calculate the lifetime of the Z0 boson.
446
447     Args:
448     partial_width_z0 (float): The partial width of the Z0 boson.
449
450     Returns:
451     float: The calculated lifetime of the Z0 boson in seconds.
452     """
453     # Get the lifetime value for Z0
454     lifetime_z0 = HBAR_GEV_S / partial_width_z0
455     return lifetime_z0
456
457
458 def calculate_lifetime_uncertainty(partial_width_z0,
459                                   partial_width_uncertainty):
460     """
461     Calculate the uncertainty in the lifetime of the Z0 boson.
462
463     Args:
464     partial_width_z0 (float): The partial width of the Z0 boson in
465                             GeV.
466     partial_width_uncertainty (float): The uncertainty in the
467                                     partial width.
468
469     Returns:
470     float: The uncertainty in the lifetime of the Z0 boson in
471           seconds.
472     """
473     lifetime_uncertainty = HBAR_GEV_S / (partial_width_z0 ** 2) * \
474                             partial_width_uncertainty
475     return lifetime_uncertainty
476
477
478 def approximate_ellipse_center(center_of_mass_energy, cross_section
479                                ,
480                                uncertainty, mass_z0_range,
481                                partial_width_z0_range):
482     """
483     (This function is for No.4 additional feature)
484     Approximate the geometric center of the chi-square ellipse for
485     one
486     sigma level. It is really helpful for further analysis of the
487     quality of the
488     fit.
489
490     Args:
491     center_of_mass_energy, cross_section, uncertainty: Data arrays.
492     mass_z0_range, partial_width_z0_range: Ranges for Z0 boson mass
493                                     and width.
494
495     Returns:

```

```

489         tuple: Approximate geometric center coordinates of the chi-
               square ellipse.
490         """
491         sigma_level = 1
492         mass_z0_values = np.linspace(*mass_z0_range, 100)
493         partial_width_z0_values = np.linspace(*partial_width_z0_range,
494                                                100)
495         mass_z0_grid, partial_width_z0_grid = np.meshgrid(
496             mass_z0_values, partial_width_z0_values)
497         chi_sq_grid = np.zeros_like(mass_z0_grid)
498
499         # Calculate chi-square values
500         for i in range(mass_z0_grid.shape[0]):
501             for j in range(partial_width_z0_grid.shape[1]):
502                 chi_sq_grid[i, j] = chi_square(
503                     [mass_z0_grid[i, j], partial_width_z0_grid[i, j]],
504                     center_of_mass_energy, cross_section, uncertainty)
505
506         min_chi_sq = chi_sq_grid.min()
507         chi_sq_threshold = min_chi_sq + sigma_level**2
508
509         # Find points within the sigma level contour
510         within_contour = np.where(chi_sq_grid <= chi_sq_threshold)
511         contour_mass_z0 = mass_z0_grid[within_contour]
512         contour_partial_width_z0 = partial_width_z0_grid[within_contour]
513
514         # Calculate the approximate geometric center
515         center_mass_z0 = np.mean(contour_mass_z0)
516         center_partial_width_z0 = np.mean(contour_partial_width_z0)
517         return center_mass_z0, center_partial_width_z0
518
519 def gaussian(x_value, mean_value, sigma):
520     """Returns the value of a Gaussian probability density function
521         at x."""
522     return 1 / (sigma * np.sqrt(2 * np.pi)) * \
523         np.exp(-0.5 * ((x_value - mean_value) / sigma) ** 2)
524
525 def plot_gaussian_distribution(x_value, mean_value,
526                               x_value_uncertainty,
527                               title, filename):
528     """
529     (This function is for No.5 additional feature)
530
531     Plots a Gaussian distribution for a given parameter.
532
533     """
534     x_values = np.linspace(x_value - 10 * x_value_uncertainty,
535                           x_value + 10 *
536                               x_value_uncertainty, 1000)
537     y_values = gaussian(x_values, mean_value, x_value_uncertainty)
538
539     plt.figure(figsize=(12, 6))
540     plt.plot(x_values, y_values, label=f'Gaussian of {title}')
541     plt.xlabel('Values')

```

```

540 plt.ylabel('Probability Density')
541 plt.title(f'Distribution for Z0 Boson {title}')
542 plt.legend()
543 plt.grid(True)
544 plt.savefig(filename)
545 plt.show()
546
547
548 def save_data_to_csv_file(filename, data):
549     """
550     (This function is for No.6 additional feature)
551
552     Save the final filtered data as a CSV file for future use. This
553     is an
554     optional output determined by user.
555
556     Args:
557     filename (str): Name of the CSV file to be saved.
558     data (list of tuples): Data to be saved
559     """
560     with open(filename, mode='w', newline='', encoding='utf-8') as
561         file:
562         # Write the header
563         file.write("Time (s),Fractional Intensity\n")
564         # Write the data rows
565         for time, frac_intensity in data:
566             file.write(f"{time},{frac_intensity}\n")
567
568 def assess_fit_quality(fitted_mass_z0, fitted_width_z0,
569                       center_mass_z0,
570                       center_partial_width_z0):
571     """
572     Assess the quality of the fit by checking if the Gaussian
573     center points
574     for mass_z0 and width_z0 are within the uncertainty range of
575     the fitted
576     values.
577
578     Args:
579     fitted_mass_z0 (float): Fitted Z boson mass.
580     fitted_width_z0 (float): Fitted Z boson width.
581     center_mass_z0 (float): Center point of the Gaussian for
582     mass_z0.
583     center_width_z0 (float): Center point of the Gaussian for
584     width_z0.
585
586     Returns:
587     None
588     """
589     tolerance = 0.0001
590     mass_in_good_quality = abs(
591         fitted_mass_z0 - center_mass_z0) <= tolerance
592     width_in_good_quality = abs(
593         fitted_width_z0 - center_partial_width_z0) <= tolerance
594
595     if mass_in_good_quality and width_in_good_quality:

```

```

590         print("Fit Quality: Good. Best fit is off-center but within
                    tolerance,"
591               "indicating the distributions are Gaussian
                    distribution.")
592
593     else:
594         print("Fit Quality: Not Good. Best fit point deviates from
                    the"
595               "ellipse's center. the output distribution plots are
                    not"
596               "gaussian distribution, Consider re-optimizing chi-
                    square.")
597
598
599 def main():
600     """
601     Main function
602
603     """
604     # Combine two data sets
605     combined_data = read_and_combine_data(FILE_NAME_1, FILE_NAME_2)
606     if combined_data is None:
607         return
608     filtered_data_1 = filter_data(combined_data, num_std=3)
609
610     # Get the data arrays filtered by filter_1 function
611     center_of_mass_energy = filtered_data_1[:, 0]
612     cross_section = filtered_data_1[:, 1]
613     uncertainty = filtered_data_1[:, 2]
614
615     # Take the initial guess for two parameters
616     initial_guesses = [90, 3]
617     result = min_chi_square(
618         center_of_mass_energy, cross_section, uncertainty,
619         initial_guesses)
620     mass_z0_fit, partial_width_z0_fit = result[0]
621
622     # Get the final data arrays which filtered by filter_1 &
623     filter_2 functions
624     final_filter_data = filter_data_2(
625         center_of_mass_energy, cross_section, mass_z0_fit,
626         partial_width_z0_fit, filtered_data_1)
627     center_of_mass_energy_2 = final_filter_data[:, 0]
628     cross_section_2 = final_filter_data[:, 1]
629     uncertainty_2 = final_filter_data[:, 2]
630
631     result_2 = min_chi_square(
632         center_of_mass_energy_2, cross_section_2, uncertainty_2,
633         initial_guesses)
634     mass_z0_fit, partial_width_z0_fit = result_2[0]
635
636     # Calculate the reduced chi_square
637     num_data_points = len(cross_section_2)
638     num_parameters = len(result_2[0])
639     reduced_chi_sq_2 = calculate_reduced_chi_square(
        result_2[1], num_data_points, num_parameters)

```

```

640 # Plot the min_chi_square fit with scatted datas
641 plot_data_and_fit(center_of_mass_energy_2,
642                   cross_section_2, uncertainty_2, mass_z0_fit,
643                   partial_width_z0_fit)
644
645 # Plot the chi_square contours
646 mass_z0_range = [mass_z0_fit - 0.03, mass_z0_fit + 0.03]
647 partial_width_z0_range = [
648     partial_width_z0_fit - 0.03, partial_width_z0_fit + 0.03]
649 plot_chi_square_contours(
650     center_of_mass_energy_2, cross_section_2, uncertainty_2,
651     mass_z0_range, partial_width_z0_range, mass_z0_fit,
652     partial_width_z0_fit)
653
654 # Get the result of Z boson mass, Z boson width and minimum
655     chi_square
656 mass_z0_fit, partial_width_z0_fit = result_2[0]
657 chi_squared_min = result_2[1]
658
659 # Calculate uncertainties for mass_z0, partial_width_z0
660 mass_z0_uncertainty = find_uncertainty_hill_climbing(
661     final_filter_data, result_2[0], chi_squared_min,
662     param_index=0)
663 partial_width_z0_uncertainty = find_uncertainty_hill_climbing(
664     final_filter_data, result_2[0], chi_squared_min,
665     param_index=1)
666
667 # Calculate the lifetime_z0
668 lifetime_z0 = calculate_lifetime(partial_width_z0_fit)
669 # Cal
670 lifetime_z0_uncertainty = calculate_lifetime_uncertainty(
671     partial_width_z0_fit, partial_width_z0_uncertainty)
672
673 # Plot the Gaussian distributions for mass_z0 and
674     partial_width_z0
675 center_mass_z0, center_partial_width_z0 =
676     approximate_ellipse_center(
677         center_of_mass_energy_2, cross_section_2, uncertainty_2,
678         mass_z0_range, partial_width_z0_range)
679
680 plot_gaussian_distribution(mass_z0_fit, center_mass_z0,
681     mass_z0_uncertainty, 'Mass (mass_z0)',
682     'gaussian_distribution_mass_z0.png')
683
684 plot_gaussian_distribution(partial_width_z0_fit,
685     center_partial_width_z0,
686     partial_width_z0_uncertainty,
687     'Partial Width (partial_width_z0)',
688     'gaussian_distribution_partial_width_z0.png')
689
690 # Check the quality of the fit with the gaussian plots
691 assess_fit_quality(mass_z0_fit, partial_width_z0_fit,
692     center_mass_z0,
693     center_partial_width_z0)

```

```

687 # Print result values and their uncertainties
688 print(
689     f"Z boson mass (mass_z0): "
690     f"{mass_z0_fit:.4g}      {mass_z0_uncertainty:.5f} GeV/c^2"
691 )
692 print(
693     f"Z boson width (partial_width_z0): "
694     f"{partial_width_z0_fit:.4g}      {
695         partial_width_z0_uncertainty:.5f} GeV"
696 )
697 print(
698     f"Z boson lifetime: "
699     f"{lifetime_z0:.2e} s      {lifetime_z0_uncertainty:.2e} s"
700 )
701 print(f"Minimum_value chi-squared: {result_2[1]:.3f}")
702 print(f"Reduced chi-squared: {reduced_chi_sq_2:.3f}")
703
704 # Interactive menu
705 while True:
706     user_input = input(
707         "Enter '1' for 3D Chi-Square Contours, "
708         "'2' to save filtered data, or 'quit' to exit: "
709     )
710
711     if user_input.lower() == 'quit':
712         print("Exiting the program.")
713         break
714
715     if user_input == '1':
716         # Plot chi-square contours with contour lines
717         plot_chi_square_3d_contours(center_of_mass_energy_2,
718                                     cross_section_2,
719                                     uncertainty_2,
720                                     mass_z0_range,
721                                     partial_width_z0_range)
722
723     elif user_input == '2':
724         # Save filtered data
725         filename_data = 'final_filtered_data.csv'
726         np.savetxt(filename_data, final_filter_data, delimiter=
727             ',',
728             header='Center_of_Mass_Energy, Cross_Section
729             , \
730             Uncertainty',
731             comments='', fmt='%f')
732         print(f"Filtered data saved to {filename_data}.")
733
734 if __name__ == "__main__":
735     main()

```