Z Boson Lab Script

Yupeng Fan

October 2024

```
"""
Title: PHYS20161 - Assignment 2: Z Boson


Main Goal:

This script is designed to analyze two particle collision data. First,it need
to combine, validate the data sets. Second, it performs best fit plot by
finding the minimum of the chi_square value. Meanwhile, this code is able to
get the parameters value for best fit and life time for Z0, and calculate the
uncertainties. Finally, this code also have several additional features listed
below.

Additional Features:

1. Further validation on the cross_section datas.

2. Chi-Square Contour Plot: Provides a visualization of the
chi_square minimization process.

3. Extra 3D plot of the Contour Plot, optional output decided by the user.

4. Validation whether the distribution is gaussian and the jusdge the quality
of the fit: Estimate the geometric centre of the chi_square cntour and
define a funtction to assess the quality of the fit.
(The user will see result clearly in output)

5. Plots of the Gaussian distribution for both the mass (mass_z0) and the
partial width ( Z0 ).
(To decide whether it is gaussian is checked by No.4 addtional features)

6. CSV Data Export Function: A function to save filtered data for further use
```

```python
or record-keeping.

7. Interactive Menu: Allows users to customize the outputs based on their
preferences for further output options, enhancing user experience.

Last updated: 13/12/2023
@author: w23058yf
"""

import matplotlib.pyplot as plt
import numpy as np
import scipy.constants as pc
from scipy.optimize import fmin

# Constants
HBAR_GEV_S = pc.hbar / (1.602e-19 * 1e9)  # Reduced Planck constant in GeV*s
PARTIAL_WIDTH_EE = 83.91 * 1e-3  # Convert MeV to GeV

# File names
FILE_NAME_1 = 'z_boson_data_1.csv'
FILE_NAME_2 = 'z_boson_data_2.csv'


def validate_data(data):
    """
    Validates the data by removing NaN values, zero or negative values,

    Args:
    data (array): The data array.

    Returns:
    array: The validated and cleaned data array.
    """

    # Remove the NaN values
    data = data[~np.isnan(data).any(axis=1)]

    # Remove zero or negative values
    data = data[np.all(data > 0, axis=1)]

    return data


def read_and_combine_data(file1, file2):
    """
    Reads data from two chi_squared_contour_plotV files, validates, and
    combines them into a single array.

    Args:
    file1 (string): The first data file.
    file2 (string): The second data file.

    Returns:
```

```
    array: A combined and validated array with data from both files,
    or None if an error occurs.
    """
    try:
        data1 = np.genfromtxt(file1, delimiter=',', skip_header=1)
        data2 = np.genfromtxt(file2, delimiter=',', skip_header=1)

        data1 = validate_data(data1)
        data2 = validate_data(data2)

        # Check column consistency between the two data sets
        if data1.shape[1] != data2.shape[1]:
            raise ValueError(
                "Mismatch in the number of columns between datasets.")
        # Combine two sets of data sets
        combined_data = np.vstack((data1, data2))
        return combined_data
    except IOError as error:
        print(f"File not found: {error.filename}")
        return None
    except ValueError as error:
        print(f"Error reading file: {error}")
        return None


def filter_data(data, num_std=3):
    """
    Filter data by removing outliars based on a specified number of
    standard deviations.

    Args:
    data (array): The data array.
    num_std (int): The number of standard deviations.

    Returns:
    array: The filtered data array.
    """
    # Calculate the mean of the cross-section values
    mean_cross_section = np.mean(data[:, 1])

    # Calculate the standard deviation of the cross-section values
    std_dev_cross_section = np.std(data[:, 1])

    # Filter out data points where the cross-section deviates from the mean by
    # more than 'num_std' standard deviations.
    return data[abs(data[:, 1] - mean_cross_section) <= num_std *
                std_dev_cross_section]


def min_chi_square(center_of_mass_energy, cross_section, uncertainty,
                   initial_guesses):
    """
    Find the parameters that minimize the chi-square value.
```

```python

    Args:
    center_of_mass_energy (array): Array of center-of-mass energies.
    cross_section (array): Array of cross sections.
    uncertainty (array): Array of uncertainties in the cross section
    measurements.
    initial_guesses (list): List of initial guesses for the mass and partial
    width of the Z0 boson.

    Returns:
    result(tuple)
    """
    result = fmin(lambda parameters: chi_square(parameters,
                                                center_of_mass_energy,
                                                cross_section, uncertainty),
                  initial_guesses, full_output=True, disp=False)
    return result


def filter_data_2(center_of_mass_energy, cross_section,
                  mass_z0_fit, partial_width_z0_fit, filtered_data_1):
    """
    (This function is for No.1 additional feature)

    Filter data based on predicted cross sections compared to actual
    cross sections.


    Args:
    center_of_mass_energy (array): Array of center-of-mass energies.
    cross_section (array): Array of measured cross sections.
    uncertainty (array): Array of uncertainties
    mass_z0_fit (float): Fitted Z0 boson mass.
    partial_width_z0_fit (float): Fitted Z0 boson partial width.
    filtered_data_1 (array): Previously filtered data array.

    Returns:
    array: Further filtered data array.
    """
    # Perdict the value of cross section by the formean_valuela.
    perdict_cross_section = calculate_cross_section(
        center_of_mass_energy, mass_z0_fit, partial_width_z0_fit)
    # Get the standard ddeviation value of the cross section.
    std_dev_cross_section = np.std(perdict_cross_section)
    return filtered_data_1[np.abs(perdict_cross_section - cross_section) <=
                           0.5 * std_dev_cross_section]


```

```
def calculate_reduced_chi_square(chi_sq, num_data_points, num_parameters):
    """
    Calculate the reduced chi-square value.

    Args:
    chi_sq (float): The chi-square value from the fit.
    num_data_points (int): The number of data points used in the fit.
    num_parameters (int): The number of parameters.

    Returns:
    float: The reduced chi-square value.
    """
    degrees_of_freedom = num_data_points - num_parameters
    return chi_sq / degrees_of_freedom


def calculate_cross_section(center_of_mass_energy, mass_z0, partial_width_z0):
    """
    Calculate the cross section for a given center-of-mass energy,
    Z0 boson mass, and partial width.

    Args:
    center_of_mass_energy (array): Array of center-of-mass energies.
    mass_z0 (float): Z0 boson mass.
    partial_width_z0 (float): Z0 boson partial width.

    Returns:
    array: Calculated cross section values.
    """
    # Define the given formula to calculate the cross section
    numerator = 12 * np.pi * center_of_mass_energy**2 * (PARTIAL_WIDTH_EE**2)
    denominator = (center_of_mass_energy**2 - mass_z0**2)**2 + \
        (mass_z0**2) * (partial_width_z0**2)
    return numerator / (denominator * mass_z0**2) * 389400


def chi_square(parameters, center_of_mass_energy, cross_section_data,
               uncertainty):
    """
    Calculate the chi-square value for the fit.

    Args:
    parameters (list): List containing the mass and partial width of the Z0
    boson.
    center_of_mass_energy (array): Array of center-of-mass
    energies.
    cross_section_data (array): Array of measured cross
    sections.
    uncertainty (array): Array of uncertainties

```

```
    Returns:
    float: The calculated chi-square value.
    """
    mass_z0, partial_width_z0 = parameters
    # Get the perdiction value of cross section
    prediction = calculate_cross_section(center_of_mass_energy, mass_z0,
                                         partial_width_z0)
    # Get the chi_square value
    chi_sq = np.sum(((cross_section_data - prediction) / uncertainty) ** 2)
    return chi_sq


def plot_data_and_fit(center_of_mass_energy, cross_section_data, uncertainty,
                      mass_z0, partial_width_z0):
    """
    Plots the data points with error bars and the best fit curve.

    Args:
    center_of_mass_energy (array): Array of center-of-mass energies.
    cross_section_data (array): Array of measured cross sections.
    uncertainty (array): Array of uncertainties
    mass_zo (float): Z boson mass parameter
    partial_width_z0(float): Z boson partial width parameter

    Returns:
    None
    """
    plt.figure(figsize=(10, 6))
    plt.errorbar(center_of_mass_energy, cross_section_data, yerr=uncertainty,
                 fmt='o', label='Data', color='royalblue', ecolor='lightgray',
                 elinewidth=3, capsize=0)

    # Create a smooth line for the fit
    center_of_mass_energy_fit = np.linspace(
        min(center_of_mass_energy), max(center_of_mass_energy), 1000)
    cross_section_fit = calculate_cross_section(
        center_of_mass_energy_fit, mass_z0, partial_width_z0)

    # Plot the fit
    plt.plot(center_of_mass_energy_fit, cross_section_fit,
             label='Fit', linewidth=2, color='darkorange')
    plt.xlabel('Centre-of-mass energy (GeV)', fontsize=14)
    plt.ylabel('Cross Section (nb)', fontsize=14)
    plt.title('Fitted Curve with Scattered Data', fontsize=16)
    plt.legend()
    plt.grid(True)
    plt.savefig('z_boson_fit.png')
    plt.show()


```

```python
def plot_chi_square_contours(center_of_mass_energy, cross_section, uncertainty,
                             mass_z0_range, partial_width_z0_range,
                             mass_z0_fit, partial_width_z0_fit):
    """
    (This function is for No.2 additional feature)

    Provide visualization of chi_square for 2 parameters with fit values,
    and contour line which is helpful in future analysis.

    Args:
    center_of_mass_energy (array): Array of center-of-mass energies.
    cross_section (array): Array of measured cross sections.
    uncertainty (array): Array of uncertainties in the cross section
    measurements.
    mass_z0_range (tuple): Range for the Z0 boson mass range.
    partial_width_z0_range (tuple): Range for the Z0 boson width range.
    mass_z0_fit (float): Fitted value of Z0 boson mass.
    partial_width_z0_fit (float): Fitted value of Z0 boson width.

    Returns:
    None
    """
    mass_z0_values = np.linspace(*mass_z0_range, 100)
    partial_width_z0_values = np.linspace(*partial_width_z0_range, 100)
    mass_z0_grid, partial_width_z0_grid = np.meshgrid(
        mass_z0_values, partial_width_z0_values)
    chi_sq_grid = np.zeros_like(mass_z0_grid)

    # Calculate chi-square values
    for i in range(mass_z0_grid.shape[0]):
        for j in range(mass_z0_grid.shape[1]):
            chi_sq_grid[i, j] = chi_square([mass_z0_grid[i, j],
                                            partial_width_z0_grid[i, j]],
                                           center_of_mass_energy,
                                           cross_section, uncertainty)
    # Find the minimum chi-square value
    best_chi_square = np.min(chi_sq_grid)
    level_1 = best_chi_square + 1
    # Plot Contours
    plt.contourf(mass_z0_grid, partial_width_z0_grid,
                 chi_sq_grid, levels=50, cmap='viridis')
    plt.colorbar(label='Chi-squared')
    # Add contour line
    plt.contour(mass_z0_grid, partial_width_z0_grid,
                chi_sq_grid, levels=[level_1],
                colors='red', linestyles='dashed')
    # Label for minimum chi-square + 1
    plt.plot([], [], 'r--', label='Min chi-square + 1')
    # Plotting the best fit values as a red dot
```

```python
    plt.plot(mass_z0_fit, partial_width_z0_fit, 'ro', label='Minimum Fit')
    plt.axhline(y=partial_width_z0_fit, color='r', linestyle='--')
    plt.axvline(x=mass_z0_fit, color='r', linestyle='--')
    plt.xlabel('Z boson mass (mass_z0) [GeV/c^2]')
    plt.ylabel('Z boson width (partial_width_z0) [GeV]')
    plt.title('Chi-squared Contours')
    plt.legend()
    plt.savefig("chi_square_contours.png", format='png')
    plt.show()


def plot_chi_square_3d_contours(center_of_mass_energy, cross_section,
                                uncertainty, mass_z0_range,
                                partial_width_z0_range):
    """
    (This function is for No.3 additional feature)

    Plot a 3D contour of the chi-square values, which is helpful in more vivid
    visualisasion for chisquare contour. This is an optional output, which is
    determined by user.

    Args:
    center_of_mass_energy (array): Array of center-of-mass energies.
    cross_section (array): Array of measured cross sections.
    uncertainty (array): Array of uncertainties in the cross section
    measurements.
    mass_z0_range (tuple): Range for the Z0 boson mass
    range.
    partial_width_z0_range (tuple): Range for the Z0
    boson width range.
    mass_z0_fit (float): Fitted value of Z0 boson mass.
    partial_width_z0_fit (float): Fitted value of Z0 boson width.

    Returns:
    None
    """
    mass_z0_values = np.linspace(*mass_z0_range, 100)
    partial_width_z0_values = np.linspace(*partial_width_z0_range, 100)
    mass_z0_grid, partial_width_z0_grid = np.meshgrid(
        mass_z0_values, partial_width_z0_values)
    chi_sq_grid = np.zeros_like(mass_z0_grid)

    # Calculate chi-square values
    for i in range(mass_z0_grid.shape[0]):
        for j in range(mass_z0_grid.shape[1]):
            chi_sq_grid[i, j] = chi_square([mass_z0_grid[i, j],
                                            partial_width_z0_grid[i, j]],
                                           center_of_mass_energy,
                                           cross_section,
```

```python
                                                    uncertainty)
    fig = plt.figure()
    axes = fig.add_subplot(111, projection='3d')

    # Plot a 3D contour
    axes.contour3D(mass_z0_grid, partial_width_z0_grid,
                   chi_sq_grid, 50, cmap='viridis')

    # Label the axes
    axes.set_xlabel('Z boson mass (mass_z0) [GeV/c^2]')
    axes.set_ylabel('Z boson width (partial_width_z0) [GeV]')
    axes.set_zlabel('Chi-squared')
    axes.legend()
    # Save the plot in png version
    plt.savefig("3D_chi_square_contours.png", format='png')
    plt.show()


def find_uncertainty_hill_climbing(data, best_fit_parameters,
                                   chi_squared_min, param_index,
                                   delta_chi_square=1.0):
    """
    Estimate the uncertainty of the fit parameters using the hill-climbing
    method.

    Args:
    data (array): data array.
    best_fit_parameters (list): List of the best fit parameters.
    chi_squared_min (float): The minimum chi-square.
    param_index (int): Index of the parameter to calculate the uncertainty.
    step_size (float): The step size
    delta_chi_sq (float): The change in chi-square value.
    Returns:
    float: The uncertainties of two parameters.
    """
    # Set proper step size
    step_size = 0.00001
    parameter_plus = best_fit_parameters[param_index]
    parameter_minus = best_fit_parameters[param_index]
    chi_squared_plus = chi_squared_min
    chi_squared_minus = chi_squared_min
    # Apply hill climbing method from two directions
    while chi_squared_plus - chi_squared_min < delta_chi_square:
        parameter_plus += step_size
        parameters = best_fit_parameters.copy()
        parameters[param_index] = parameter_plus
        chi_squared_plus = chi_square(
            parameters, data[:, 0], data[:, 1], data[:, 2])

    while chi_squared_minus - chi_squared_min < delta_chi_square:
        parameter_minus -= step_size
        parameters = best_fit_parameters.copy()
        parameters[param_index] = parameter_minus
        chi_squared_minus = chi_square(
            parameters, data[:, 0], data[:, 1], data[:, 2])
```

```python
    uncertainty = (parameter_plus - parameter_minus) / 2
    return uncertainty


def calculate_lifetime(partial_width_z0):
    """
    Calculate the lifetime of the Z0 boson.

    Args:
    partial_width_z0 (float): The partial width of the Z0 boson.

    Returns:
    float: The calculated lifetime of the Z0 boson in seconds.
    """
    # Get the lifetime value for Z0
    lifetime_z0 = HBAR_GEV_S / partial_width_z0
    return lifetime_z0


def calculate_lifetime_uncertainty(partial_width_z0,
                                   partial_width_uncertainty):
    """
    Calculate the uncertainty in the lifetime of the Z0 boson.

    Args:
    partial_width_z0 (float): The partial width of the Z0 boson in
        GeV.
    partial_width_uncertainty (float): The uncertainty in the
        partial width.

    Returns:
    float: The uncertainty in the lifetime of the Z0 boson in
        seconds.
    """
    lifetime_uncertainty = HBAR_GEV_S / (partial_width_z0 ** 2) * \
        partial_width_uncertainty
    return lifetime_uncertainty


def approximate_ellipse_center(center_of_mass_energy, cross_section
    ,
                               uncertainty, mass_z0_range,
                               partial_width_z0_range):
    """
    (This function is for No.4 additional feature)
    Approximate the geometric center of the chi-square ellipse for
        one
    sigma level. It is really helpful for further anlysis of the
        quality of the
    fit.

    Args:
    center_of_mass_energy, cross_section, uncertainty: Data arrays.
    mass_z0_range, partial_width_z0_range: Ranges for Z0 boson mass
         and width.

    Returns:
```

```
    tuple: Approximate geometric center coordinates of the chi-square ellipse.
    """
    sigma_level = 1
    mass_z0_values = np.linspace(*mass_z0_range, 100)
    partial_width_z0_values = np.linspace(*partial_width_z0_range, 100)
    mass_z0_grid, partial_width_z0_grid = np.meshgrid(
        mass_z0_values, partial_width_z0_values)
    chi_sq_grid = np.zeros_like(mass_z0_grid)

    # Calculate chi-square values
    for i in range(mass_z0_grid.shape[0]):
        for j in range(mass_z0_grid.shape[1]):
            chi_sq_grid[i, j] = chi_square(
                [mass_z0_grid[i, j], partial_width_z0_grid[i, j]],
                center_of_mass_energy, cross_section, uncertainty)

    min_chi_sq = chi_sq_grid.min()
    chi_sq_threshold = min_chi_sq + sigma_level**2

    # Find points within the sigma level contour
    within_contour = np.where(chi_sq_grid <= chi_sq_threshold)
    contour_mass_z0 = mass_z0_grid[within_contour]
    contour_partial_width_z0 = partial_width_z0_grid[within_contour]

    # Calculate the approximate geometric center
    center_mass_z0 = np.mean(contour_mass_z0)
    center_partial_width_z0 = np.mean(contour_partial_width_z0)
    return center_mass_z0, center_partial_width_z0


def gaussian(x_value, mean_value, sigma):
    """Returns the value of a Gaussian probability density function at x."""
    return 1 / (sigma * np.sqrt(2 * np.pi)) * \
        np.exp(-0.5 * ((x_value - mean_value) / sigma) ** 2)


def plot_gaussian_distribution(x_value, mean_value, x_value_uncertainty,
                               title, filename):
    """
    (This function is for No.5 additional feature)

    Plots a Gaussian distribution for a given parameter.

    """
    x_values = np.linspace(x_value - 10 * x_value_uncertainty, x_value + 10 *
                           x_value_uncertainty, 1000)
    y_values = gaussian(x_values, mean_value, x_value_uncertainty)

    plt.figure(figsize=(12, 6))
    plt.plot(x_values, y_values, label=f'Gaussian of {title}')
    plt.xlabel('Values')
```

```
    plt.ylabel('Probability Density')
    plt.title(f'Distribution for Z0 Boson {title}')
    plt.legend()
    plt.grid(True)
    plt.savefig(filename)
    plt.show()


def save_data_to_csv_file(filename, data):
    """
    (This function is for No.6 additional feature)

    Save the final filtered data as a CSV file for future use. This is an
    optional output determined by user.

    Args:
    filename (str): Name of the CSV file to be saved.
    data (list of tuples): Data to be saved
    """
    with open(filename, mode='w', newline='', encoding='utf-8') as file:
        # Write the header
        file.write("Time (s),Fractional Intensity\n")
        # Write the data rows
        for time, frac_intensity in data:
            file.write(f"{time},{frac_intensity}\n")


def assess_fit_quality(fitted_mass_z0, fitted_width_z0, center_mass_z0,
                       center_partial_width_z0):
    """
    Assess the quality of the fit by checking if the Gaussian center points
    for mass_z0 and width_z0 are within the uncertainty range of the fitted
    values.

    Args:
    fitted_mass_z0 (float): Fitted Z boson mass.
    fitted_width_z0 (float): Fitted Z boson width.
    center_mass_z0 (float): Center point of the Gaussian for mass_z0.
    center_width_z0 (float): Center point of the Gaussian for width_z0.

    Returns:
    None
    """
    tolerance = 0.0001
    mass_in_good_quality = abs(
        fitted_mass_z0 - center_mass_z0) <= tolerance
    width_in_good_quality = abs(
        fitted_width_z0 - center_partial_width_z0) <= tolerance

    if mass_in_good_quality and width_in_good_quality:
```

```python
        print("Fit Quality: Good. Best fit is off-center but within tolerance,"
              "indicating the distributions are Gaussian distribution.")

    else:
        print("Fit Quality: Not Good. Best fit point deviates from the"
              "ellipse's center. the output distribution plots are not"
              "gaussion distribution,Consider re-optimizing chi-square.")


def main():
    """
    Main function

    """
    # Combine two data sets
    combined_data = read_and_combine_data(FILE_NAME_1, FILE_NAME_2)
    if combined_data is None:
        return
    filtered_data_1 = filter_data(combined_data, num_std=3)

    # Get the data arrays filtered by filter_1 function
    center_of_mass_energy = filtered_data_1[:, 0]
    cross_section = filtered_data_1[:, 1]
    uncertainty = filtered_data_1[:, 2]

    # Take the initial guess for two parameters
    initial_guesses = [90, 3]
    result = min_chi_square(
        center_of_mass_energy, cross_section, uncertainty, initial_guesses)
    mass_z0_fit, partial_width_z0_fit = result[0]

    # Get the final data arrays which filtered by filter_1 & filter_2 functions
    final_filter_data = filter_data_2(
        center_of_mass_energy, cross_section, mass_z0_fit,
        partial_width_z0_fit, filtered_data_1)
    center_of_mass_energy_2 = final_filter_data[:, 0]
    cross_section_2 = final_filter_data[:, 1]
    uncertainty_2 = final_filter_data[:, 2]

    result_2 = min_chi_square(
        center_of_mass_energy_2, cross_section_2, uncertainty_2,
        initial_guesses)
    mass_z0_fit, partial_width_z0_fit = result_2[0]

    # Calculate the reduced chi_square
    num_data_points = len(cross_section_2)
    num_parameters = len(result_2[0])
    reduced_chi_sq_2 = calculate_reduced_chi_square(
        result_2[1], num_data_points, num_parameters)

```

```python
    # Plot the min_chi_square fit with scatted datas
    plot_data_and_fit(center_of_mass_energy_2,
                      cross_section_2, uncertainty_2, mass_z0_fit,
                      partial_width_z0_fit)

    # Plot the chi_square contours
    mass_z0_range = [mass_z0_fit - 0.03, mass_z0_fit + 0.03]
    partial_width_z0_range = [
        partial_width_z0_fit - 0.03, partial_width_z0_fit + 0.03]
    plot_chi_square_contours(
        center_of_mass_energy_2, cross_section_2, uncertainty_2,
        mass_z0_range, partial_width_z0_range, mass_z0_fit,
        partial_width_z0_fit)

    # Get the result of Z boson mass, Z boson width and minimum chi_square
    mass_z0_fit, partial_width_z0_fit = result_2[0]
    chi_squared_min = result_2[1]

    # Calculate uncertainties for mass_z0, partial_width_z0
    mass_z0_uncertainty = find_uncertainty_hill_climbing(
        final_filter_data, result_2[0], chi_squared_min, param_index=0)
    partial_width_z0_uncertainty = find_uncertainty_hill_climbing(
        final_filter_data, result_2[0], chi_squared_min, param_index=1)
    # Calculate the lifetime_z0
    lifetime_z0 = calculate_lifetime(partial_width_z0_fit)
    # Cal
    lifetime_z0_uncertainty = calculate_lifetime_uncertainty(
        partial_width_z0_fit, partial_width_z0_uncertainty)

    # Plot the Gaussian distributions for mass_z0 and partial_width_z0
    center_mass_z0, center_partial_width_z0 = approximate_ellipse_center(
        center_of_mass_energy_2, cross_section_2, uncertainty_2,
        mass_z0_range, partial_width_z0_range)

    plot_gaussian_distribution(mass_z0_fit, center_mass_z0,
                               mass_z0_uncertainty, 'Mass (mass_z0)',
                               'gaussian_distribution_mass_z0.png')

    plot_gaussian_distribution(partial_width_z0_fit, center_partial_width_z0,
                               partial_width_z0_uncertainty,
                               'Partial Width (partial_width_z0)',
                               'gaussian_distribution_partial_width_z0.png')

    # Check the quality of the fit with the gaussian plots
    assess_fit_quality(mass_z0_fit, partial_width_z0_fit, center_mass_z0,
                       center_partial_width_z0)
```

```python
    # Print result values and their uncertainties
    print(
        f"Z boson mass (mass_z0): "
        f"{mass_z0_fit:.4g}    {mass_z0_uncertainty:.5f} GeV/c^2"
    )
    print(
        f"Z boson width (partial_width_z0): "
        f"{partial_width_z0_fit:.4g}    {partial_width_z0_uncertainty:.5f} GeV"
    )
    print(
        f"Z boson lifetime: "
        f"{lifetime_z0:.2e} s    {lifetime_z0_uncertainty:.2e} s"
    )
    print(f"Minimum_value chi-squared: {result_2[1]:.3f}")
    print(f"Reduced chi_squared: {reduced_chi_sq_2:.3f}")

    # Interactive menu
    while True:
        user_input = input(
            "Enter '1' for 3D Chi-Square Contours, "
            "'2' to save filtered data, or 'quit' to exit: "
        )

        if user_input.lower() == 'quit':
            print("Exiting the program.")
            break

        if user_input == '1':
            # Plot chi-square contours with contour lines
            plot_chi_square_3d_contours(center_of_mass_energy_2,
                                        cross_section_2, uncertainty_2,
                                        mass_z0_range, partial_width_z0_range)
        elif user_input == '2':
            # Save filtered data
            filename_data = 'final_filtered_data.csv'
            np.savetxt(filename_data, final_filter_data, delimiter=',',
                       header='Center_of_Mass_Energy, Cross_Section, \
                       Uncertainty',
                       comments='', fmt='%f')
            print(f"Filtered data saved to {filename_data}.")


if __name__ == "__main__":
    main()
```