Homework_03_WI25

February 13, 2025

1 Problem 1

1.1 Part 1

Using the avialable files here, create *all* pairs of photons with the same event number. This means that you create all possible pairs of photons within the same event and you do this for each event. If the event contains 4 photons there are six photon pairs: AB, AC, AD, BC, BD, BC (the order doesn't matter). Two of these pairs are from real pions and 4 are combinatoric. If you have 1000 events with 4 photons each, you have 2000 real photons and 4000 combinatoric pairs. Some of those combinatoric pairs will have a mass near the pion mass and some will be far away.

You will notice that the files contain different numbers of photons per event and this will change the ratio of pions to background pairs. Histogram the mass distribution with an appropriate binning. Make a plot of the mass distribution for each data set.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
```

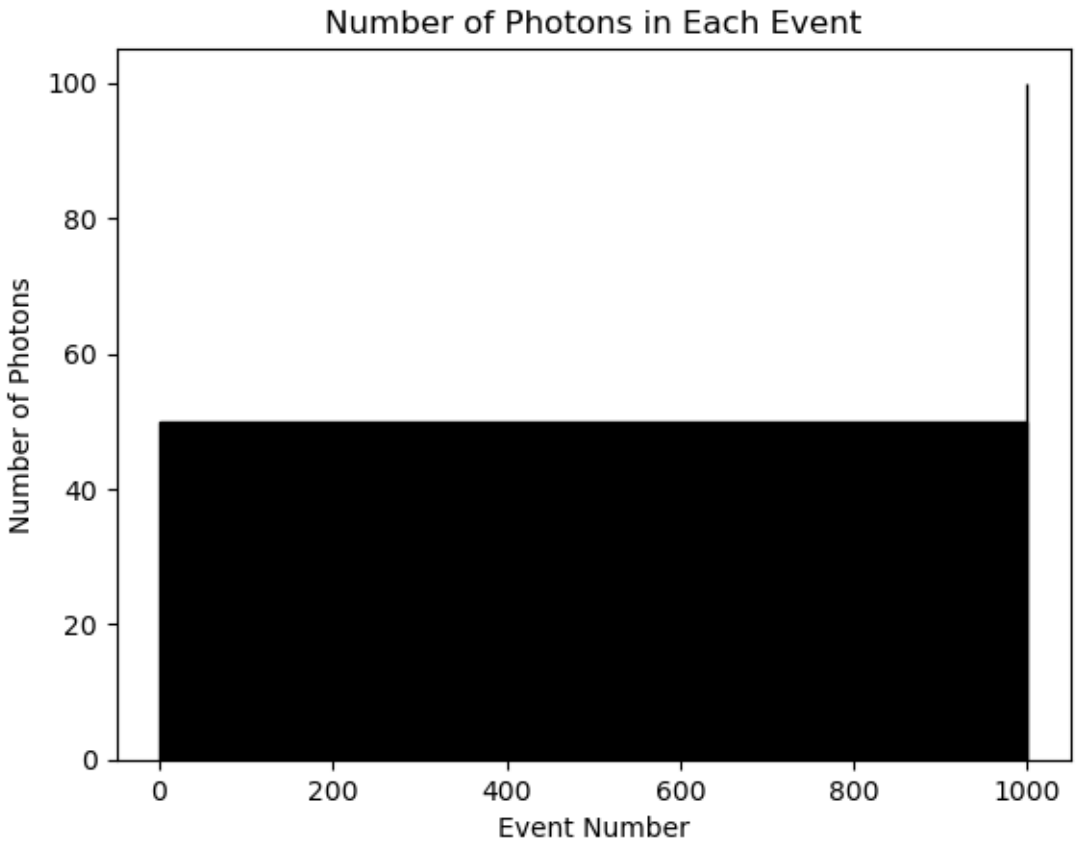
[2]:
```
# Read data, here we use pandas and read the files as dataframes for latter␣
 ↪operations
data_1 = pd.read_csv('pi0s_1.txt', sep='\s+', header=0, names=['event_number',␣
 ↪"px", "py", "pz", "E"])
data_5 = pd.read_csv('pi0s_5.txt', sep='\s+', header=0, names=['event_number',␣
 ↪"px", "py", "pz", "E"])
data_25 = pd.read_csv('pi0s_25.txt', sep='\s+', header=0,␣
 ↪names=['event_number', "px", "py", "pz", "E"])
```

[3]:
```
# You can check that for data_1, there are only 1 pions in each event so only␣
 ↪two photons with the same event_number
# get the headers of the dataframe
data_5.columns
```

[3]:
```
Index(['event_number', 'px', 'py', 'pz', 'E'], dtype='object')
```

[4]:
```
# Plot the number of photons in each event
plt.hist(data_25['event_number'], bins=range(1, 1001), edgecolor='black')
```

```
plt.xlabel('Event Number')
plt.ylabel('Number of Photons')
plt.title('Number of Photons in Each Event')
plt.show()
```



Define a function to calculate the invairant masses of possible photon pairs. Hint: Use "groupby" to group the data by *event_number*, and for entries with the same *event_number*, use "np.triu_indices" to avoid slow for loop and get possible combinators

[5]:
```
def calculate_invariant_mass_pairs(data):
    """
    Calculate the invariant mass for each photon pair in the dataset.

    Parameters
    ----------
    data : pandas.DataFrame
        DataFrame containing photon data with columns: 'event_number', 'E',␣
 ↪'px', 'py', 'pz'.

    Returns
```

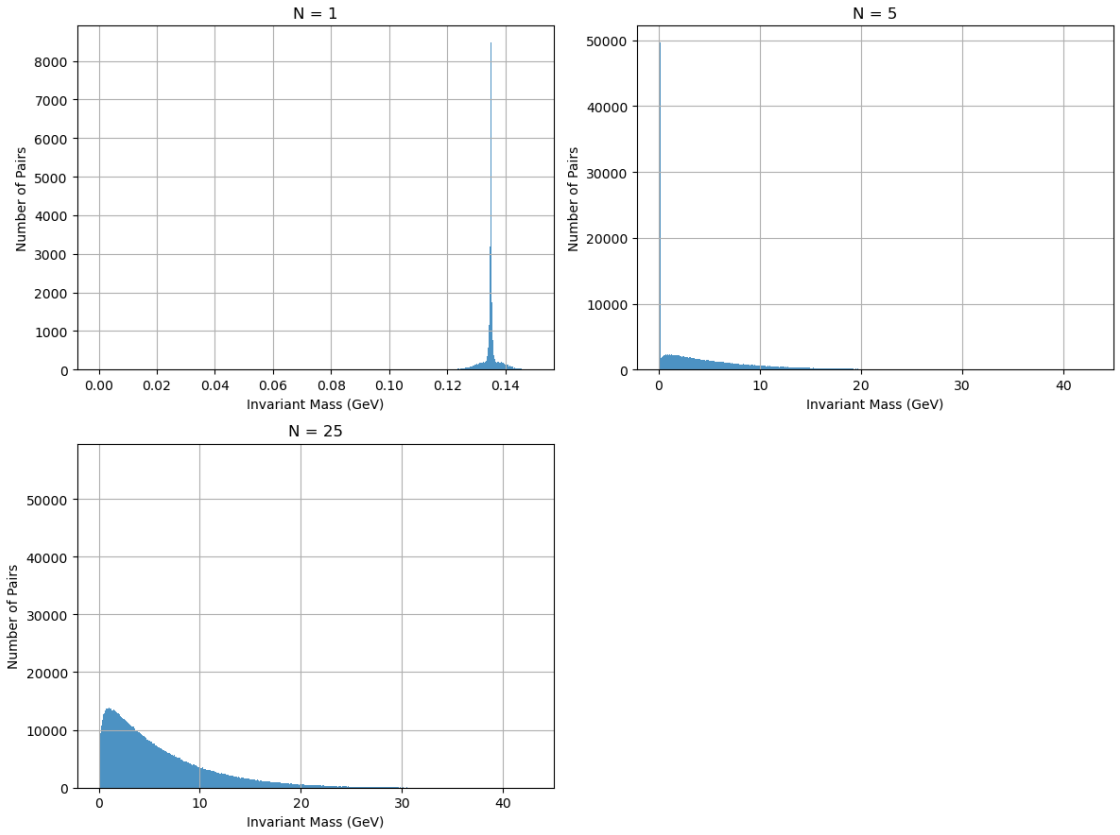
```
    -------
    mass_list : list of floats
        List of invariant mass values for each photon pair.
    """
    mass_list = []

    # Group the data by event_number
    grouped = data.groupby('event_number')

    # Loop over each event with a progress bar
    for _, group in tqdm(grouped, desc="Processing events"):
        # We expect at least two photons per event
        if len(group) < 2:
            continue

        # Extract the columns of interest as a NumPy array
        # The columns order is: E, px, py, pz.
        arr = group[['E', 'px', 'py', 'pz']].values
        n = arr.shape[0]

        # Get indices for the upper triangle of an n x n matrix (excluding␣
↪diagonal)
        i, j = np.triu_indices(n, k=1)

        # Sum the energies and momenta for each photon pair
        E_sum  = arr[i, 0] + arr[j, 0]
        px_sum = arr[i, 1] + arr[j, 1]
        py_sum = arr[i, 2] + arr[j, 2]
        pz_sum = arr[i, 3] + arr[j, 3]

        # Compute the invariant mass squared for all pairs
        mass_sq = E_sum**2 - (px_sum**2 + py_sum**2 + pz_sum**2)
        masses_event = np.sqrt(np.clip(mass_sq, 0, None))

        mass_list.extend(masses_event.tolist())

    return mass_list
```

[6]:
```
masses_1 = calculate_invariant_mass_pairs(data_1)
masses_5 = calculate_invariant_mass_pairs(data_5)
masses_25 = calculate_invariant_mass_pairs(data_25)
```

```
Processing events: 100%|
                          | 50000/50000 [00:14<00:00,
3385.44it/s]
Processing events: 100%|
                          | 10000/10000 [00:03<00:00,
```

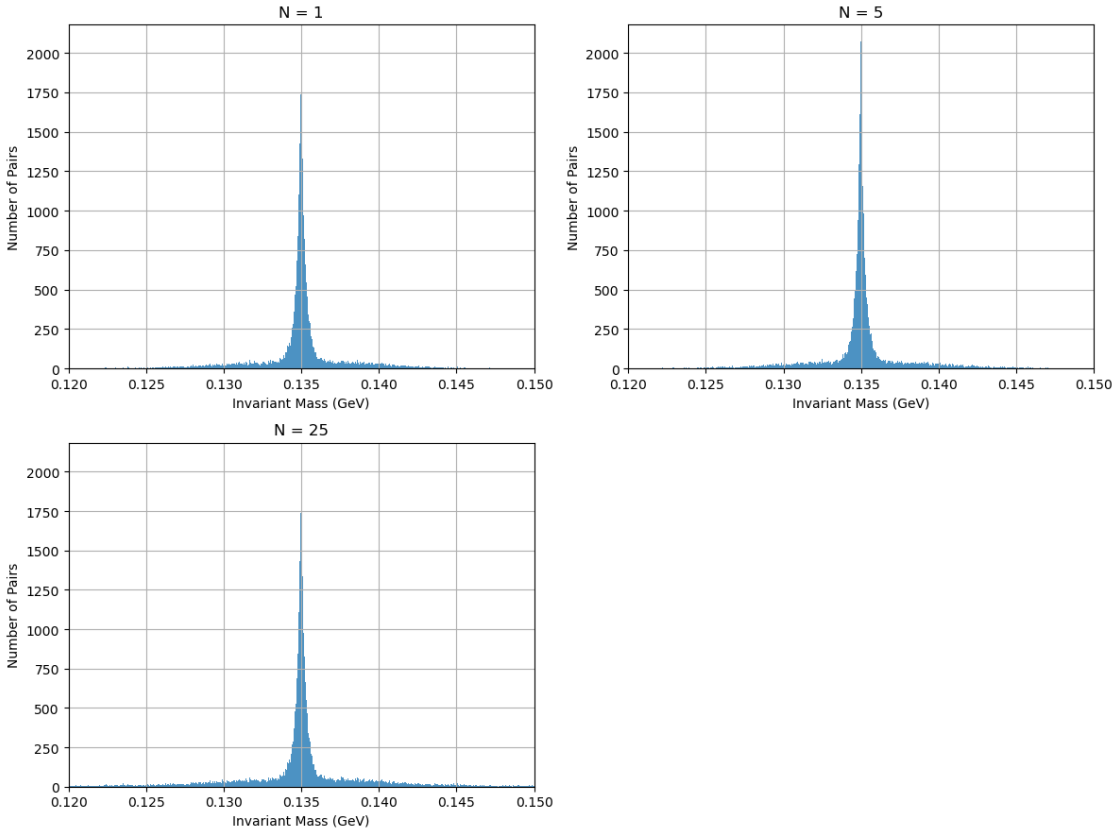
```
3307.29it/s]
Processing events: 100%|
                                | 2000/2000 [00:00<00:00,
2705.36it/s]
```

[7]:
```python
# Plot the invariant mass histogram
plt.figure(figsize=(12, 9))

plt.subplot(2, 2, 1)
plt.hist(masses_1, bins=1000, alpha=0.8)
# set the x-axis range to(0.12, 0.15)
# plt.xlim(0.12, 0.15)
plt.xlabel('Invariant Mass (GeV)')
plt.ylabel('Number of Pairs')
plt.title('N = 1')
plt.grid(True)

plt.subplot(2, 2, 2)
plt.hist(masses_5, bins=1000, alpha=0.8)
# set the x-axis range to(0.12, 0.15)
# plt.xlim(0.12, 0.15)
plt.xlabel('Invariant Mass (GeV)')
plt.ylabel('Number of Pairs')
plt.title('N = 5')
plt.grid(True)

plt.subplot(2, 2, 3)
plt.hist(masses_25, bins=1000, alpha=0.8)
# set the x-axis range to(0.12, 0.15)
# plt.xlim(0.12, 0.15)
plt.xlabel('Invariant Mass (GeV)')
plt.ylabel('Number of Pairs')
plt.title('N = 25')
plt.grid(True)

plt.tight_layout()
plt.show()
```



[8]:
```
# Define the bins: 1000 bins from 0.12 to 0.15
bins = np.linspace(0.12, 0.15, 1001)

# Plot the invariant mass histogram
plt.figure(figsize=(12, 9))

plt.subplot(2, 2, 1)
plt.hist(masses_1, bins=bins, alpha=0.8)
plt.xlim(0.12, 0.15)
plt.xlabel('Invariant Mass (GeV)')
plt.ylabel('Number of Pairs')
plt.title('N = 1')
plt.grid(True)

plt.subplot(2, 2, 2)
plt.hist(masses_5, bins=bins, alpha=0.8)
plt.xlim(0.12, 0.15)
plt.xlabel('Invariant Mass (GeV)')
plt.ylabel('Number of Pairs')
plt.title('N = 5')
plt.grid(True)
```

```
plt.subplot(2, 2, 3)
plt.hist(masses_25, bins=bins, alpha=0.8)
plt.xlim(0.12, 0.15)
plt.xlabel('Invariant Mass (GeV)')
plt.ylabel('Number of Pairs')
plt.title('N = 25')
plt.grid(True)

plt.tight_layout()
plt.show()
```



1.2 Part 2

Look at the histograms. If the mass of a photon pair is far from the pion mass, it probably didn't come from a pion. However, if the mass is near the pion mass, either it came from a pion or it is a combinatoric pair.

Find a function to fit the invariant mass distribution. What is the signal to background in the region of the pion mass (answering this question requires defining a region around the pion mass to calculate your signal; make sure you provide some justification for your definition).

Requirement: state your signal and background model, plot the invariant mass distribution and fit result, clarify your region definition, and give signal to background ratio in that region.

[9]:
```
!pip install iminuit
```

```
Collecting iminuit
  Obtaining dependency information for iminuit from https://files.pythonhosted.o
rg/packages/38/bc/a3386afa6046b187b9a5063009b87b1972885c834877beb7f3e700d52312/i
minuit-2.30.1-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
  Downloading iminuit-2.30.1-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (12 kB)
Requirement already satisfied: numpy>=1.21 in /opt/conda/lib/python3.11/site-
packages (from iminuit) (1.24.3)
Downloading
iminuit-2.30.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (440
kB)
                           440.8/440.8 kB
7.6 MB/s eta 0:00:00a 0:00:01
Installing collected packages: iminuit
Successfully installed iminuit-2.30.1
```

[17]:
```
from iminuit import Minuit
from scipy.special import erf

def fit_signal_background(mass_list, mass_min, mass_max, n_bins,␣
 ↪make_plot=True, return_fit=False):
    """
    Fit the invariant mass histogram using a Breit-Wigner signal plus linear␣
 ↪background,
    and return the estimated signal and background counts in the signal region.
    """

    # Implement your code below
    # Create the histogram from mass_list
    counts, bin_edges = np.histogram(mass_list, bins=n_bins, range=(mass_min,␣
 ↪mass_max))
    bin_width = (bin_edges[-1] - bin_edges[0]) / n_bins
    bin_centers = 0.5 * (bin_edges[1:] + bin_edges[:-1])

    # Define the fit function
    def fit_function(m, A, m0, Gamma, B0, B1):
        signal = A * Gamma**2 / ((m - m0)**2 + (Gamma**2)/4)
        background = B0 + B1 * m
        return signal + background

    # Define the chi-square function to minimize.
```

```
    def chi2(A, m0, Gamma, B0, B1):
        model = fit_function(bin_centers, A, m0, Gamma, B0, B1)
        # Use sqrt(max(count, 1)) to avoid zero uncertainty
        error = np.sqrt(np.maximum(counts, 1))
        return np.sum(((counts - model) / error)**2)

    # Implement the Minuit fitting below
    # --- Define the signal region ---
    m_low = mass_min
    m_high = mass_max
    m = Minuit(chi2, A=1000, m0=0.135, Gamma=0.005, B0=10, B1=0)
    m.migrad()

    if make_plot:
        # Print the fitted parameter values:
        print("Fitted parameters:")
        print(m.hesse())

        # Plot the histogram with error bars and the fitted function:
        plt.figure(figsize=(8,6))
        plt.errorbar(bin_centers, counts, yerr=np.sqrt(np.maximum(counts, 1)),␣
↪fmt='o', label='Data')
        # Fine grid for plotting the fit function
        m_fine = np.linspace(mass_min, mass_max, 1000)
        plt.plot(m_fine, fit_function(m_fine, *m.values), 'r-', label='Fit')
        plt.xlabel("Invariant Mass (GeV)")
        plt.ylabel("Counts")
        plt.title("Invariant Mass Distribution with Signal+Background Fit")
        plt.legend()
        plt.grid(True)
        plt.show()

    # Total data counts in the signal region (summing histogram bins)
    region_mask = (bin_centers >= m_low) & (bin_centers <= m_high)
    N_total = np.sum(counts[region_mask])

    # Estimate the background counts in the signal region by integrating the␣
↪background function:
    B0_fit = m.values["B0"]
    B1_fit = m.values["B1"]
    # Analytical integration of B0+B1*m over m_low to m_high:
    B = B0_fit*(m_high - m_low) + 0.5 * B1_fit * (m_high**2 - m_low**2)

    # Then, the estimated signal counts are the excess counts:
    S = N_total - B

    # Return the Siganl and Background counts
```

```
    return m if return_fit else S, B
```

1.3 Signal Model: Breit-Wigner

1.4 Background Model: Linear model

1.5 Signal region was chosen to be (0.12, 0.15) to fully accomodate the signal peak while letting in some background

[11]:
```
mass_dict = {'N=1': masses_1,
             'N=5': masses_5,
             'N=25': masses_25
            }
results_dict = {'N=1': (),
             'N=5': (),
             'N=25': ()
            }
for key in mass_dict.keys():
    S, B = fit_signal_background(mass_dict[key], 0.12, 0.15, 100)
    print (f'{key}, S=%.2f, B=%.2f, S/B=%.2f, uncertainty=%.4f%%'%(S, B, S/B,␣
 ↪100*np.sqrt(S+B)/S))
    N_total = S + B
    sigma = np.sqrt(N_total) # The uncertainty is defined as sqrt(N_signal +␣
 ↪N_background)
    results_dict[key] = (S, sigma)
```

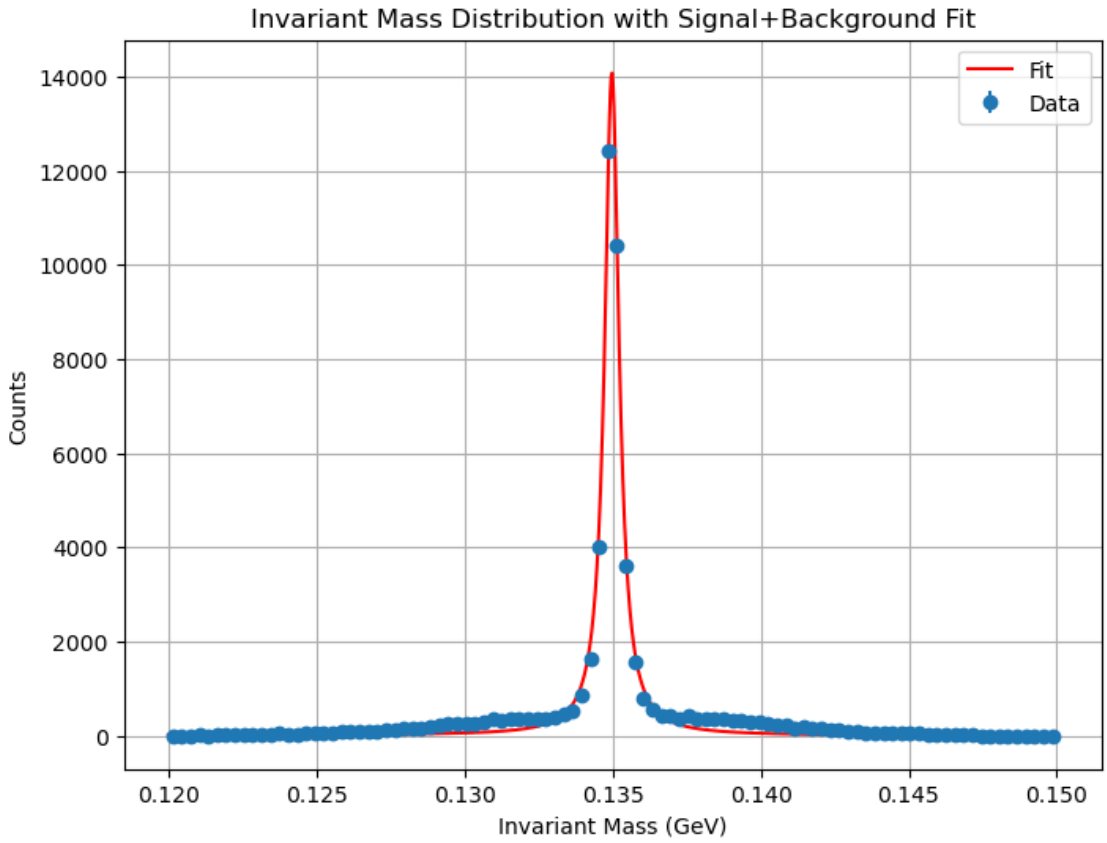
Fitted parameters:

Migrad	
FCN = 6503	Nfcn = 380
EDM = 4.72e-05 (Goal: 0.0002)	
Valid Minimum	Below EDM threshold (goal x 10)
No parameters at limit	Below call limit
Hesse ok	Covariance accurate

	Name	Value	Hesse Err	Minos Err-	Minos Err+	Limit-	Limit+	Fixed
0	A	3.520e3	0.034e3					
1	m0	134.9770e-3	0.0018e-3					

```
  2   Gamma    588e-6       5e-6

  3   B0         92           5

  4   B1        -627         32
```

```
                      A              m0          Gamma              B0
B1

     A         1.13e+03    -61.6288e-9 -140.160049e-6              5
-0
    m0      -61.6288e-9       3.19e-12        0.1e-12     43.8000e-9
-333.6713e-9
 Gamma   -140.160049e-6        0.1e-12       2.37e-11   -1.378915e-6
7.136454e-6
    B0                5     43.8000e-9   -1.378915e-6           21.1
-146
    B1               -0   -333.6713e-9    7.136454e-6           -146
1.02e+03
```



N=1, S=49872.78, B=0.22, S/B=224364.61, uncertainty=0.4478%
Fitted parameters:

Migrad

FCN = 5551 Nfcn = 341
EDM = 2.19e-06 (Goal: 0.0002)

Valid Minimum Below EDM threshold (goal x 10)

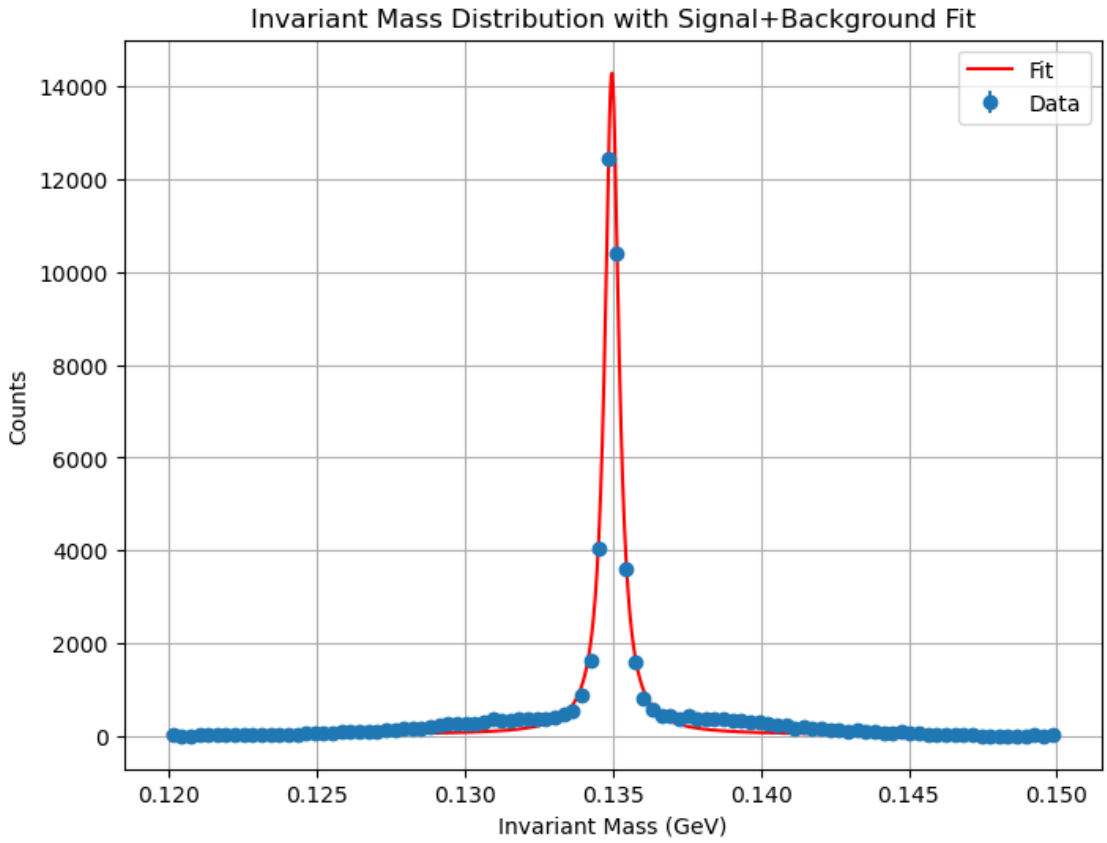
No parameters at limit Below call limit

Hesse ok Covariance accurate

	Name	Value	Hesse Err	Minos Err-	Minos Err+	Limit-	Limit+	Fixed
0	A	3.566e3	0.034e3					

```
  1   m0     134.9768e-3  0.0018e-3

  2   Gamma    574e-6       5e-6

  3   B0         81           8

  4   B1        -380         60
```

```
                      A             m0          Gamma             B0
B1

     A         1.17e+03    655.6716e-9 -141.212857e-6              0
-0
    m0      655.6716e-9       3.13e-12             -0     91.8729e-9
-684.7712e-9
 Gamma   -141.212857e-6             -0       2.33e-11   -1.181270e-6
3.940588e-6
    B0                0     91.8729e-9   -1.181270e-6           59.7
-430
    B1               -0   -684.7712e-9    3.940588e-6           -430
3.17e+03
```



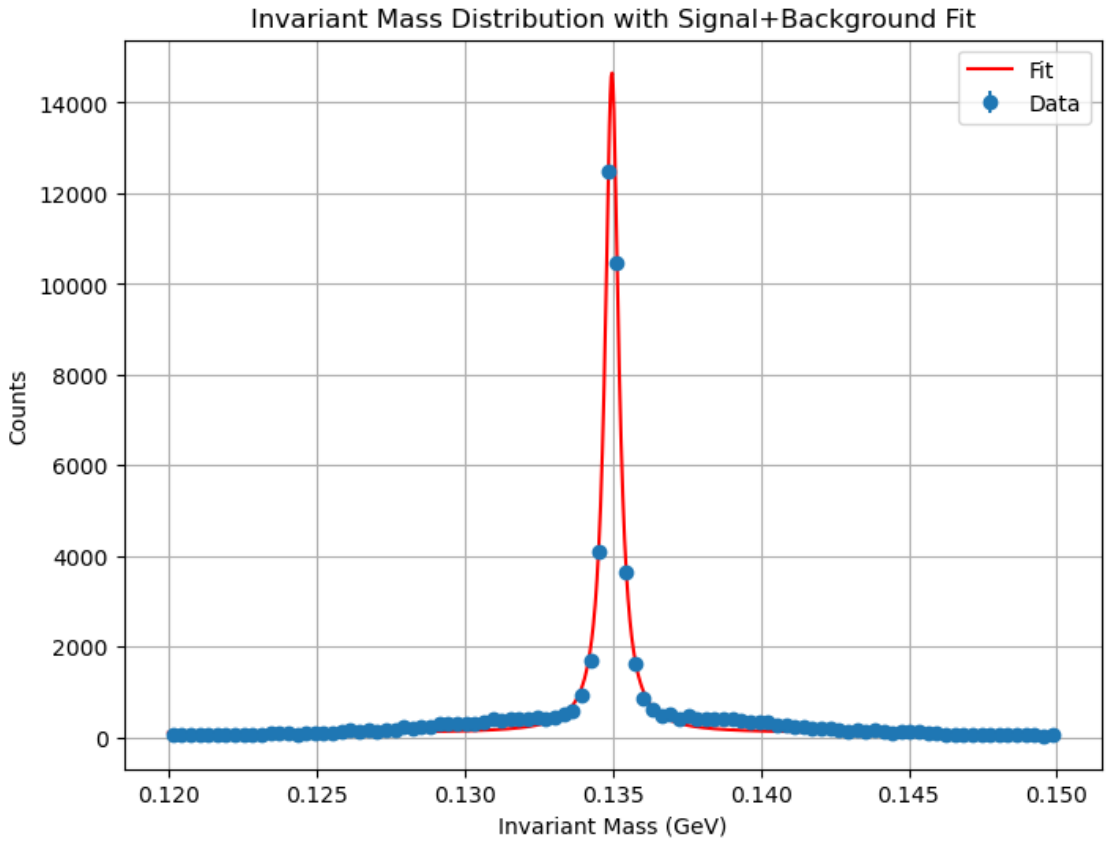
N=5, S=50768.09, B=0.91, S/B=55614.96, uncertainty=0.4438%
Fitted parameters:

Migrad	
FCN = 4040	Nfcn = 333
EDM = 5.56e-08 (Goal: 0.0002)	
Valid Minimum	Below EDM threshold (goal x 10)
No parameters at limit	Below call limit
Hesse ok	Covariance accurate

	Name	Value	Hesse Err	Minos Err-	Minos Err+	Limit-	Limit+	Fixed
0	A	3.64e3	0.04e3					

```
  1   m0     134.9765e-3  0.0017e-3

  2   Gamma    553e-6       5e-6

  3   B0         118         14

  4   B1       -0.14e3     0.10e3
```

```
                        A              m0           Gamma              B0
B1


     A         1.26e+03   2.0277180e-6 -145.995387e-6         0.01e3
-0
    m0     2.0277180e-6       3.04e-12       -0.2e-12    203.4796e-9
-1.4570135e-6
 Gamma   -145.995387e-6       -0.2e-12       2.33e-11   -1.460660e-6
2.048094e-6
    B0           0.01e3    203.4796e-9   -1.460660e-6            194
-1.43e3
    B1               -0  -1.4570135e-6    2.048094e-6        -1.43e3
1.06e+04
```




```
N=25, S=55448.01, B=2.99, S/B=18517.90, uncertainty=0.4247%
```

1.6 Part 3

Find the number of pions in each of the histograms (making sure to remove the background). How does the precision of the number of pions vary with the signal to background? Explain what you find.

The number of pions in each histogram is just the number of signals, obtained by subtracting the backgrounds from the total count:

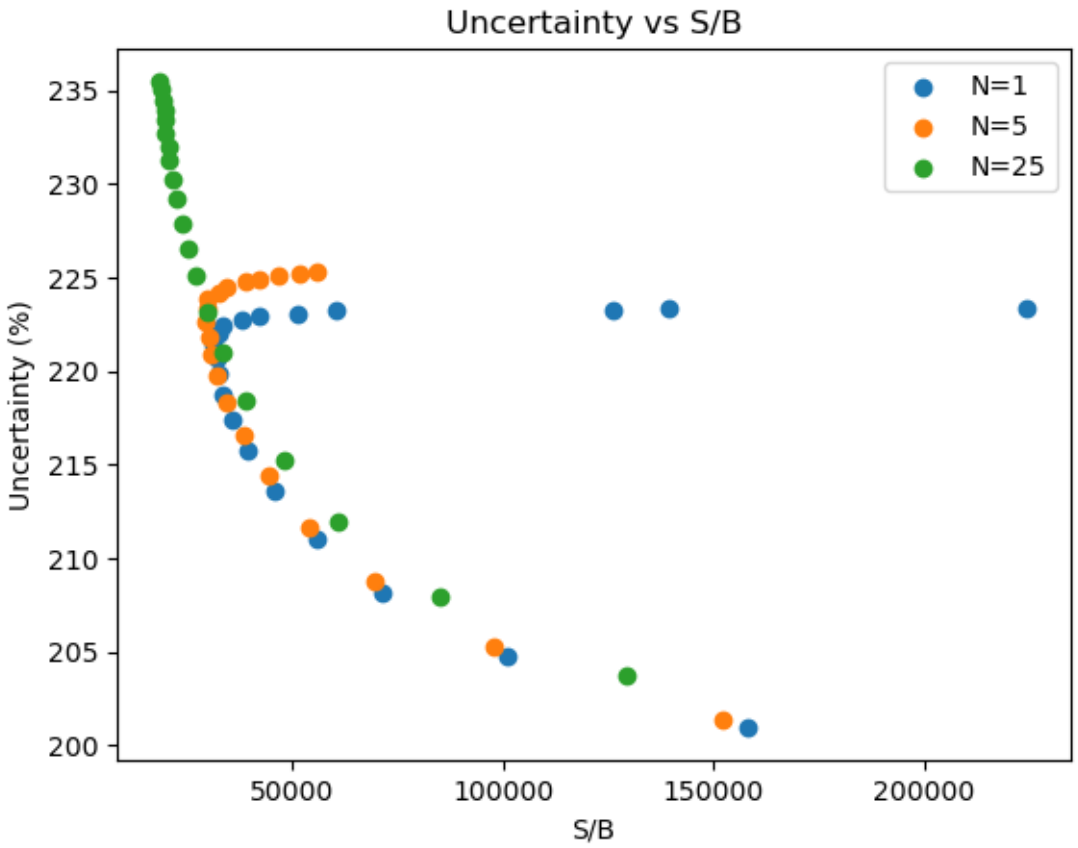
[15]:
```
for key in results_dict:
    S = results_dict[key][0]
    sigma = results_dict[key][1]
    print(f"For {key}, the number of pions is {S} with uncertainty {sigma}")
```

```
For N=1, the number of pions is 49872.77771548693 with uncertainty
223.32263655975405
For N=5, the number of pions is 50768.08715057417 with uncertainty
225.31977276750482
For N=25, the number of pions is 55448.00570822416 with uncertainty
235.4803601152334
```

1.6.1 To investigate how precision changes with signal to background

[18]:
```
# Generate 20 values for the lower edge from 0.132 to 0.12
low_edges = np.linspace(0.132, 0.12, 20)
# Generate 20 values for the upper edge from 0.138 to 0.15
high_edges = np.linspace(0.138, 0.15, 20)

# Zip them into a list of tuples
ranges = list(zip(low_edges, high_edges))

results_dict = {'N=1': [],
                'N=5': [],
                'N=25': []
               }
# loop over the ranges, for each data file, store a list of signal precisions␣
 ↪(percent error of S from the true value 50000), and a list of corresponding␣
 ↪s/b. Then plot precision over s/b.
true_S = 50000
for key in mass_dict.keys():
    for low, high in tqdm(ranges):
        S, B = fit_signal_background(mass_dict[key], low, high, 100,␣
 ↪make_plot=False)
        precision = np.sqrt(S+B)
        s_b = S / B
        results_dict[key].append((precision, s_b))

# plot precision over s/b
for key in results_dict.keys():
    precisions, s_bs = zip(*results_dict[key])
    plt.scatter(s_bs, precisions, label=key)
plt.xlabel('S/B')
plt.ylabel('Uncertainty (%)')
plt.title('Uncertainty vs S/B')
plt.legend()
plt.savefig('Uncertainty_vs_sb.png')
plt.show()
```

```
100%|
                                  | 20/20 [00:00<00:00,
129.71it/s]
100%|
                                   | 20/20
[00:00<00:00, 45.83it/s]
100%|
                                   | 20/20
[00:01<00:00, 11.72it/s]
```



1.6.2 Conclusion

From the plot above, we observe that the uncertainty decreases as the signal-to-background ratio (S/B) increases. In other words, the precision of the signal measurement increases when the S/B ratio is higher.

This effect is especially prominent for the N=25 sample.

With a dominant signal, the fitting procedure can more accurately distinguish the signal shape from the background. In contrast, a large background can mask the signal, making it harder to separate the two and increasing the uncertainty in the extracted signal parameters.

[ ]: