3038741162_Homework04

March 20, 2023

1 Homework 4: Statistics

Please complete this homework assignment in code cells in the iPython notebook. Include comments in your code when necessary. Please rename the notebook as SIS ID_HW04.ipynb (your student ID number) and save the notebook once you have executed it as a PDF (note, that when saving as PDF you don't want to use the option with latex because it crashes, but rather the one to save it directly as a PDF).

For questions that ask you to make an interpretation, please write a sentence or two explaining your response. You can use "Markdown" language to create a cell (like this one) which is ordinary text instead of using code. To do this, create a new cell, and then look at the bar above which has a picture of a floppy disk. On the right side is a drop down menu that allows you change whether a cell is "Markdown" or "Code".

The homework should be submitted on bCourses under the Assignments tab (both the .ipynb and .pdf files). Please label it by your student ID number (SIS ID)

1.1 Problem 1: Central Limit Theorem

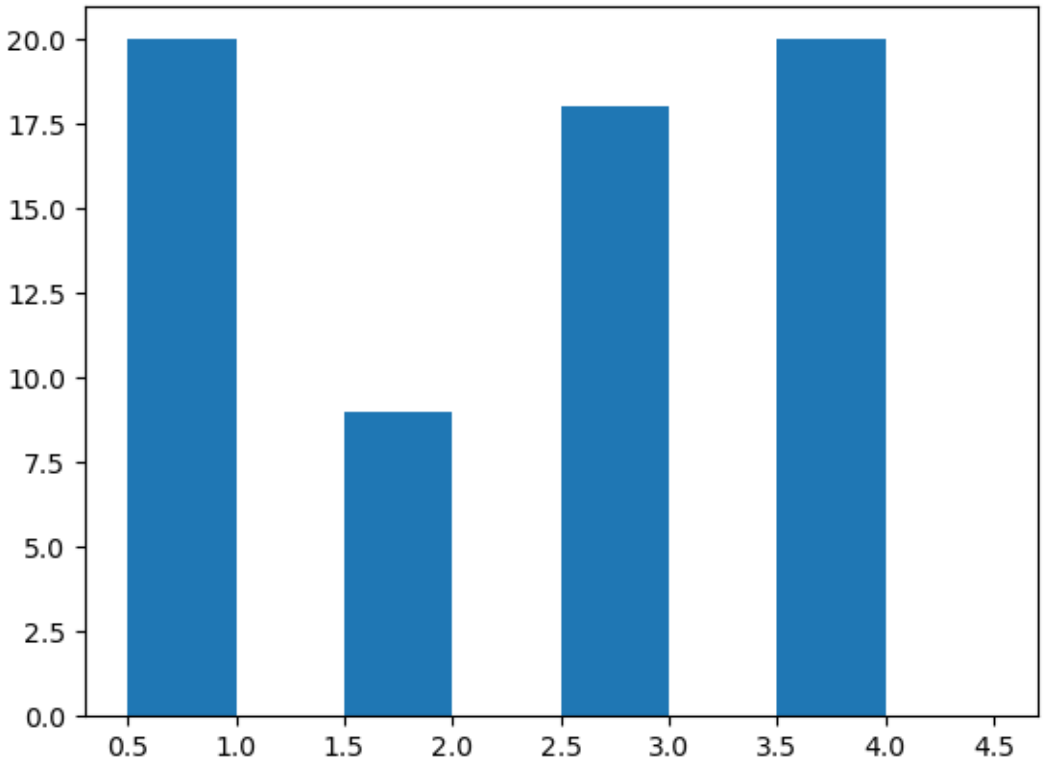
Here we will verify the Central Limit Theorem and reproduced a plot I showed in class (https://en.wikipedia.org/wiki/Central_limit_theorem#/media/File:Dice_sum_central_limit_theorem.svg)

1. Write a function that returns n integer random numbers, uniformly disributed between 1 and 6, inclusively. This represents n throws of a fair 6-sided die. The value that comes up at each throw will be called the "score".
2. Generate a distribution of 100 dice throws and plot it as a histogram normalized to unit area. Compute the mean μ_1 and standard deviation σ_1 of this distribution. Compare your numerical result to the analytical calculation.
3. Generate 10,000 sets of throws of $N = 2, 3, 4, 5, 10$ dice, computing the total sum of dice scores for each set. For each value of N, plot the distribution of total scores, and compute the mean μ_N and standard deviation σ_N of each distribution. This should be similar to the plot at the link above.
4. Plot the standard deviation σ_N as a function of N. Does it follow the Central Limit Theorem?

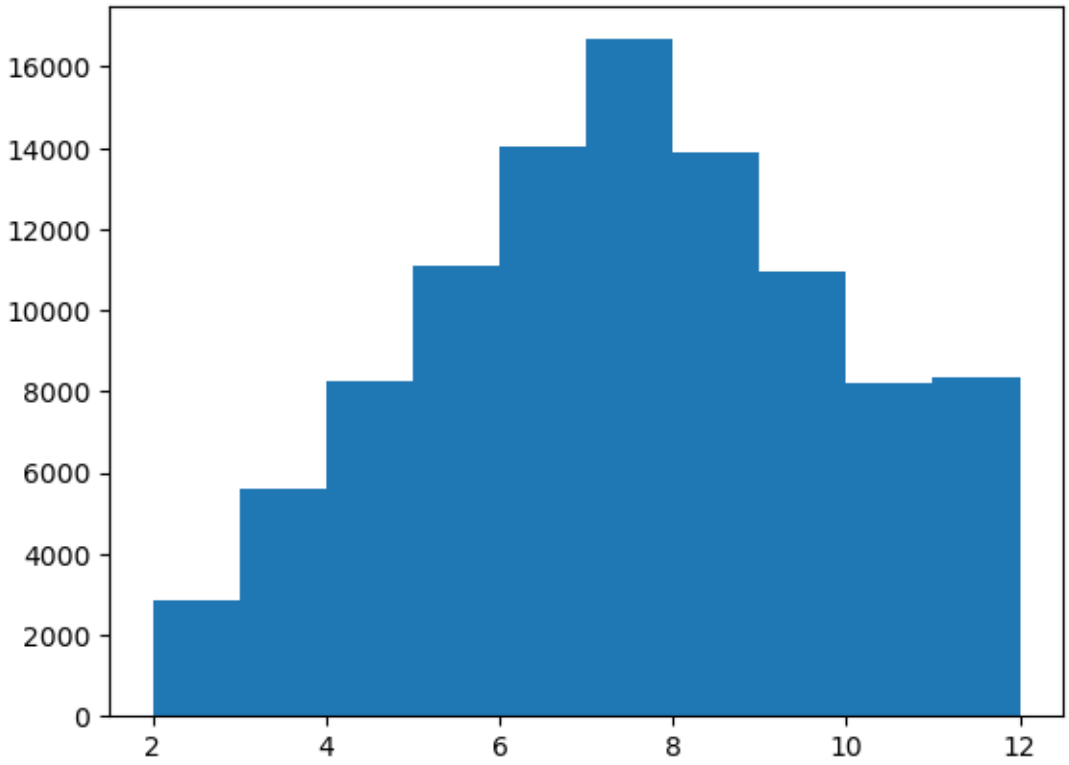
[2]:
```
#1.
import random
import numpy as np
import matplotlib.pyplot as plt

def roll_dice(n):
```

```
    #random.uniform : return float
    #returns  integer random numbers, uniformly disributed between 1 and 6,␣
 ↪inclusively.
    scores = [random.randint(1, 6) for i in range(n)]
    return scores

#2.
N = 100
roll_result = roll_dice(N)
bins = np.arange(1, 6) - 0.5

plt.hist(roll_result, bins=bins, width=0.5)
plt.show()

mu1 = 3.5
sigma1 = np.sqrt(35/12)
mean = np.mean(roll_result)
std_deviation = np.std(roll_result)
standard_error = std_deviation / np.sqrt(N)

print('Mean = {0:5.5f}'.format(mean))
print('Standard deviation = {0:5.5f}'.format(std_deviation))
print('---------------------------------------')
#  = (1+2+3+4+5+6)/6 = 3.5
print('Analytical mean = {0:5.5f}'.format(mu1))
print('Analytical standard deviation = {0:5.5f}'.format(sigma1))

#3. Generate 10,000 sets of throws of  =2,3,4,5,10 dice, computing the total␣
 ↪sum of
#dice scores for each set. For each value of  , plot the distribution of total␣
 ↪scores,
#and compute the mean  and standard deviation  of each distribution.
#This should be similar to the plot at the link above.
sigama_N_list = []
for N in [2, 3, 4, 5, 10]:
    set_of_total_scores = [sum(roll_dice(N)) for i in range(100000)]
    plt.hist(set_of_total_scores)
    plt.show()
    mu_N = np.mean(set_of_total_scores)
    sigma_N = np.std(set_of_total_scores)
    sigama_N_list.append(sigma_N)
    print(f"N = {N}: Mean = {mu_N:.5f}, Standard deviation = {sigma_N:.5f}")

#4. Plot the standard deviation  as a function of  .
#Does it follow the Central Limit Theorem?
#Central Limit Theorem:              (=   )
N_list = [2, 3, 4, 5, 10]
```

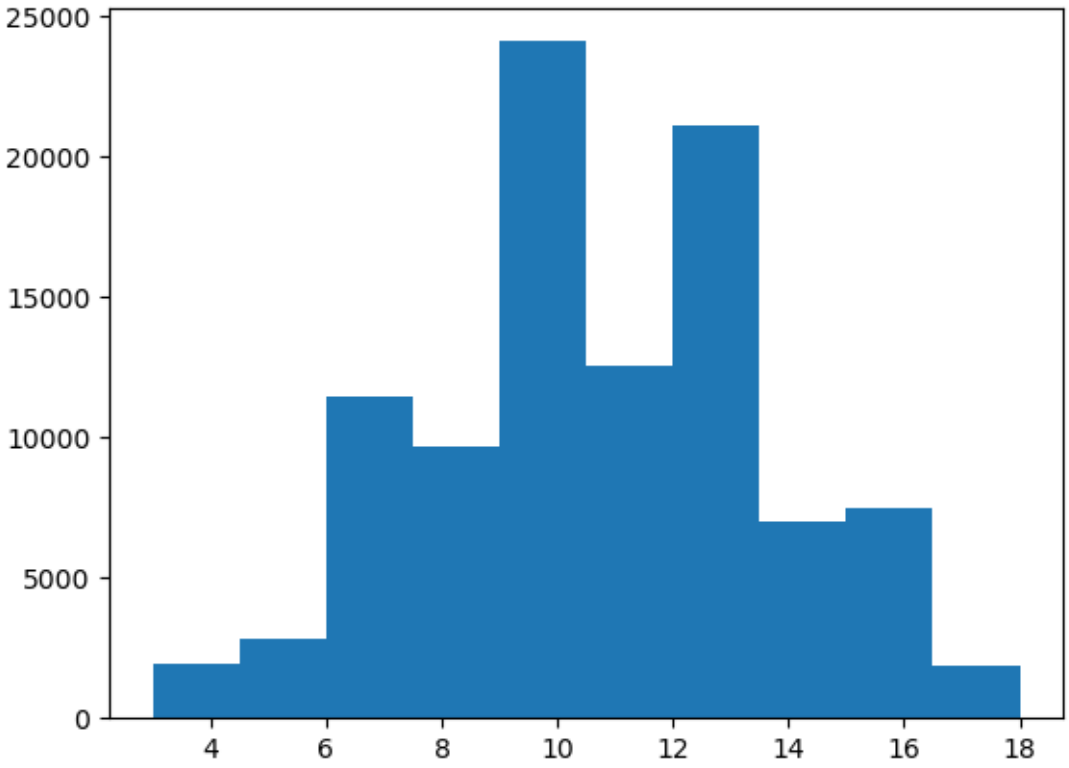
```
plt.plot(N_list, sigama_N_list, 'o-')
plt.xlabel('N')
plt.ylabel('Standard deviation')
plt.show()
```



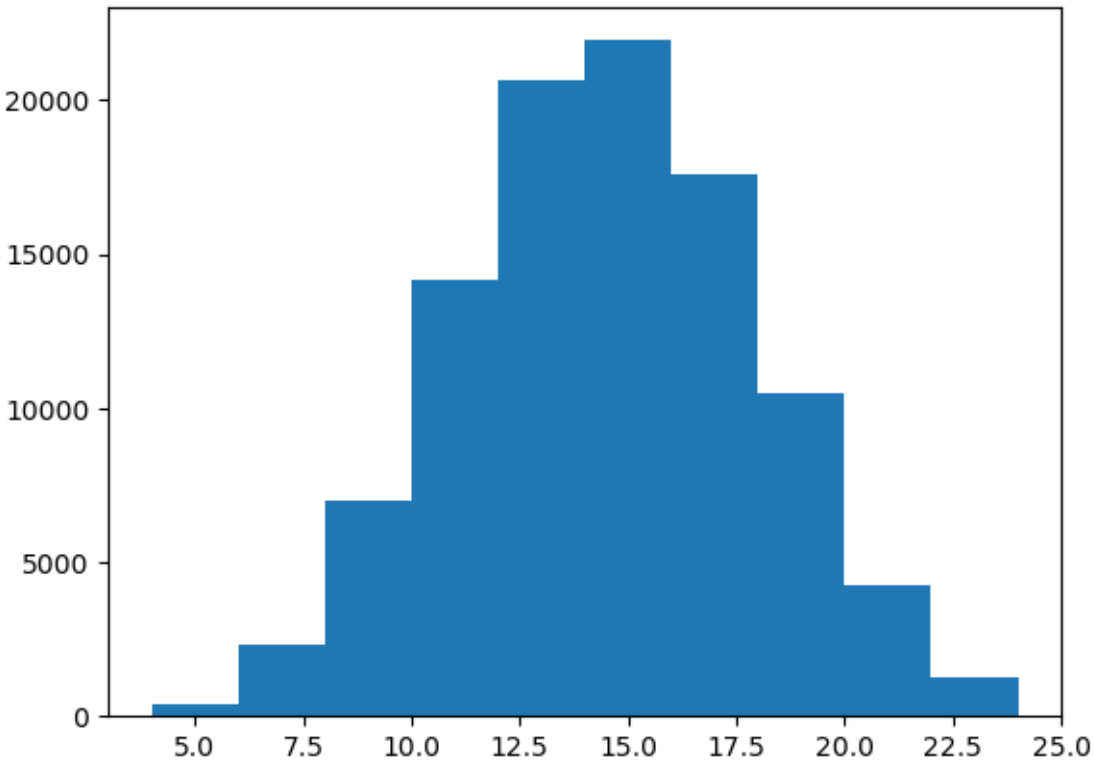

```
Mean = 3.52000
Standard deviation = 1.69988
----------------------------------------
Analytical mean = 3.50000
Analytical standard deviation = 1.70783
```



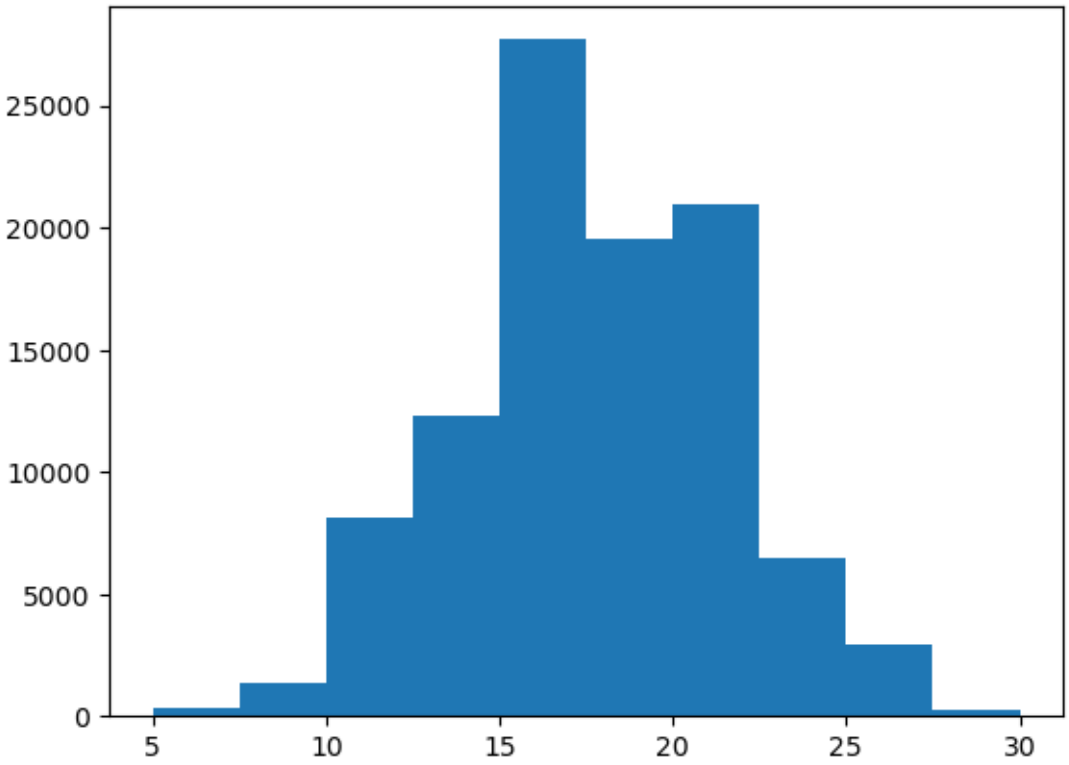

```
N = 2: Mean = 6.98608, Standard deviation = 2.41683
```



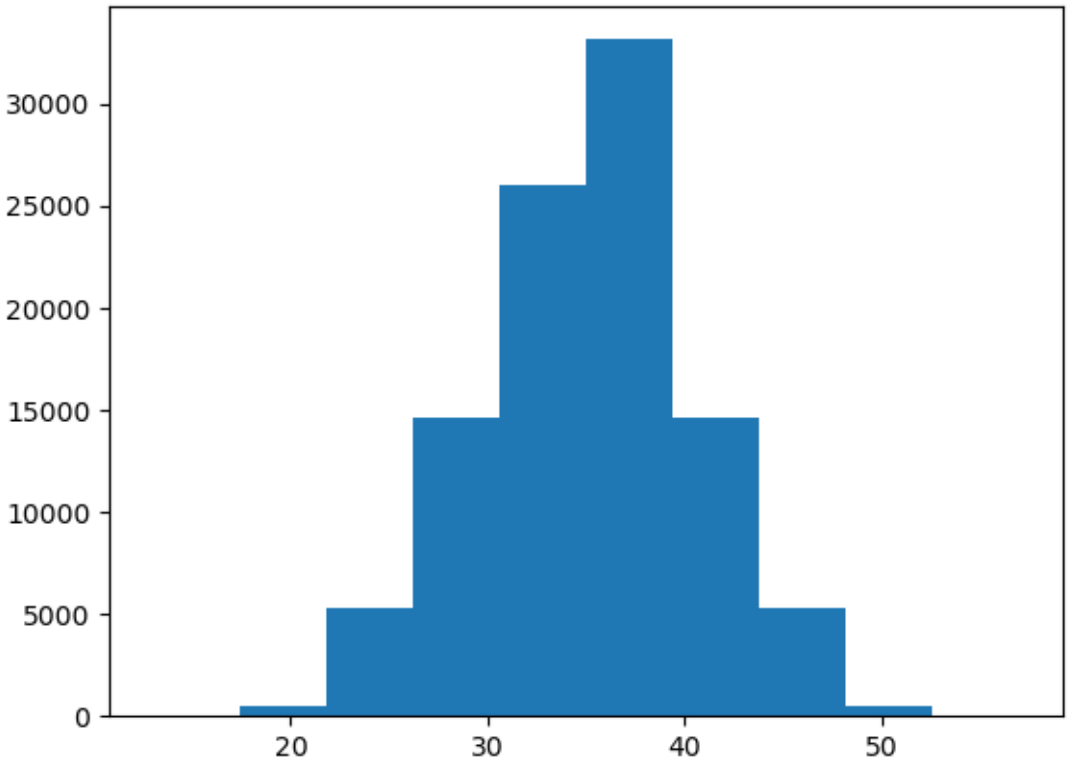

```
N = 3: Mean = 10.49656, Standard deviation = 2.96746
```



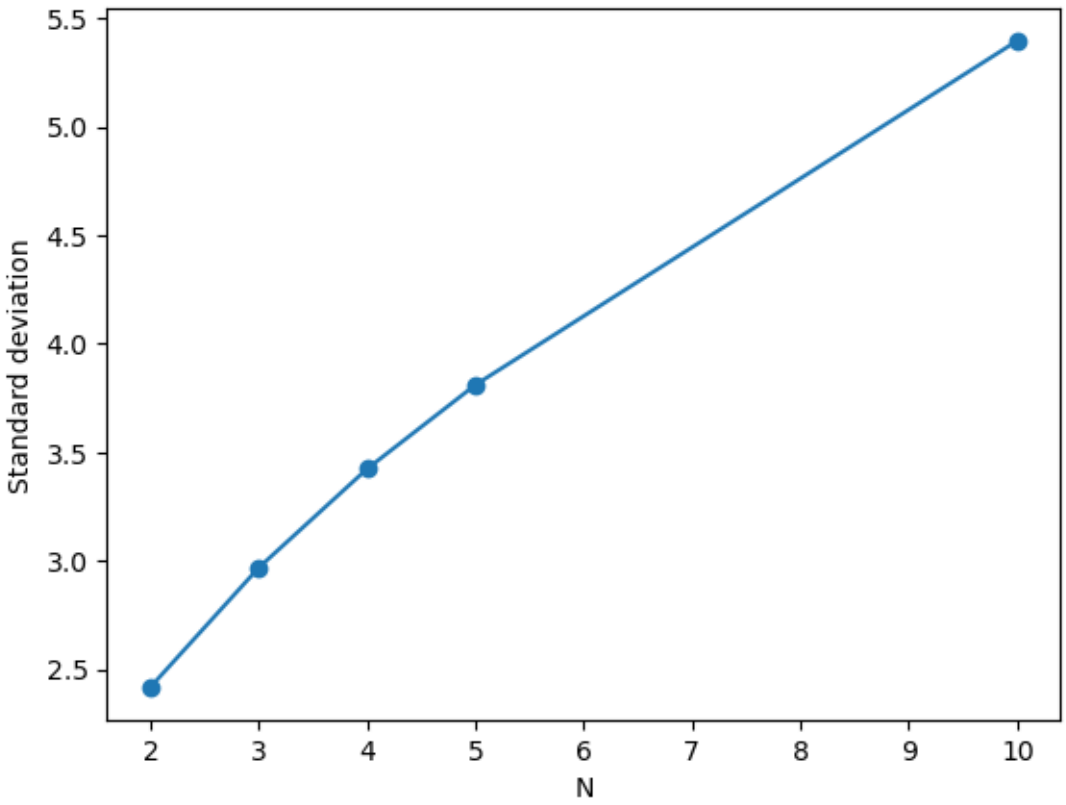

```
N = 4: Mean = 13.99791, Standard deviation = 3.42297
```




```
N = 5: Mean = 17.49820, Standard deviation = 3.81011
```




```
N = 10: Mean = 35.01072, Standard deviation = 5.39582
```



1.2 Problem 2: Parity-violating asymmetry

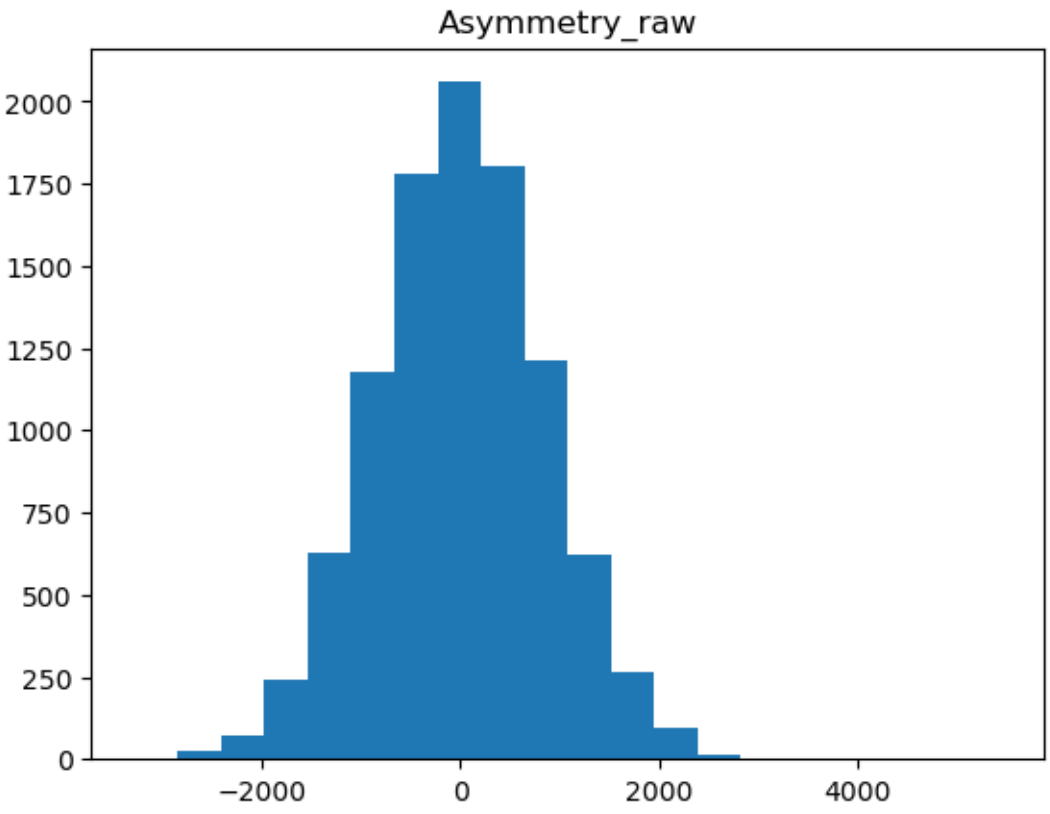
The data sample for this problem comes from the E158 experiment at SLAC (a national lab near that Junior university across the Bay). E158 measured a parity-violating asymmetry in Møller (electron-electron) scattering. This was a fixed-target experiment, which scattered longitudinally-polarized electrons off atomic (unpolarized) electrons in the 1.5m liquid hydrogen target. The data below contains a snapshot of 10,000 "events" from this experiment (overall, the experiment collected almost 400 million such events over the course of about 4 months). Each event actually records a pair of pulses: one for the right-handed electron (spin pointing along momentum) and one for the left-handed electron. For each event, we record 4 variables:

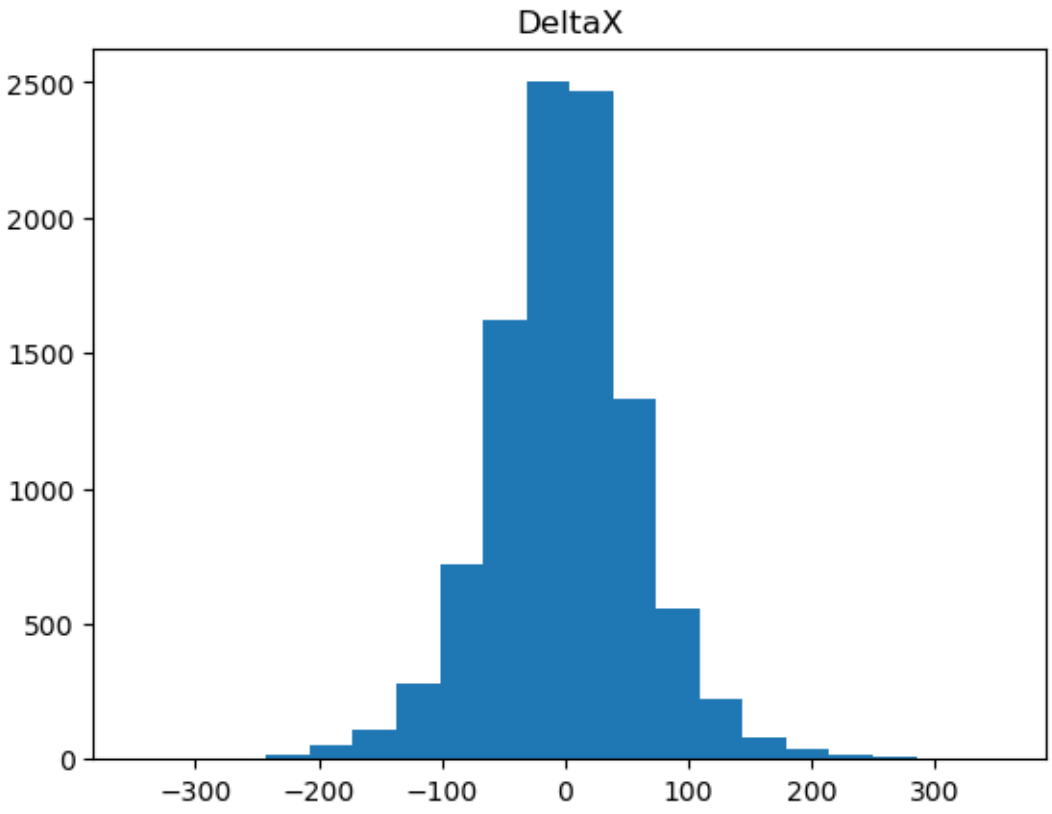
- Counter: event index
- Asymmetry: "raw" cross section asymmetry A_{raw} from one of the detector channels (there are 50 of these overall). The cross section asymmetry is defined as $A_{raw} = \frac{\sigma_R - \sigma_L}{\sigma_R + \sigma_L}$ The asymmetry is recorded in units of PPM (parts per million). It is called "raw" because corrections due to the difference in beam properties at the target are not yet applied.
- DeltaX: difference in beam position $\Delta X = X_R - X_L$ at the target in X direction in microns (with the convention that the beam is traveling along Z)
- DeltaY: difference in beam position $\Delta Y = Y_R - Y_L$ at the target in Y direction in microns

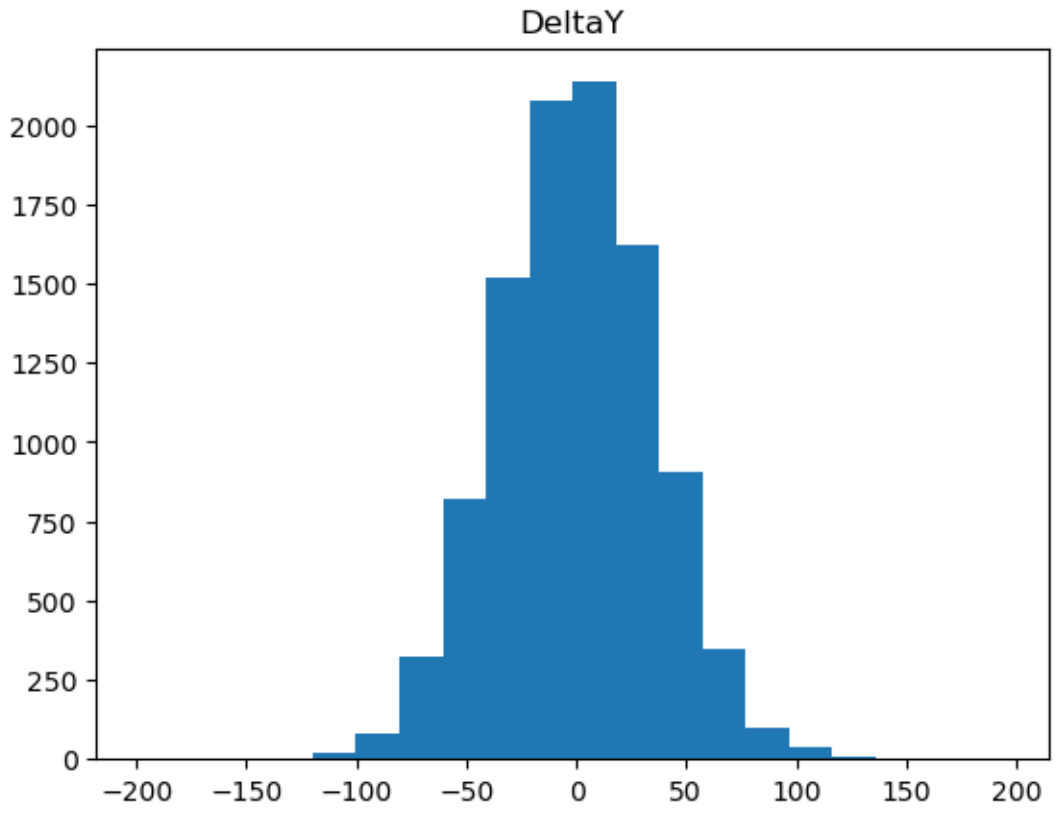
The data sample is provided in plain text format as the file asymdata.txt. Questions for this analysis:

1. Read the data from the file, and plot distributions of A_{raw}, ΔX, and ΔY.
2. Compute the mean of the raw asymmetry distribution and its statistical uncertainty.
3. Compute the standard deviation of the raw asymmetry distribution and its statistical uncertainty.
4. Compute the fraction of events contained within $\pm 1\sigma$ of the mean, $\pm 2\sigma$ of the mean, and $\pm 3\sigma$ of the mean (where σ is the standard deviation you computed in Part 3). Compare these fractions with the quantiles of the Gaussian distribution (see lecture notes) ?
5. Plot A_{raw} vs ΔX, A_{raw} vs ΔY, and ΔX vs ΔY as scatter plots.
6. Compute the correlation coefficients Corr(Asym,DeltaX), Corr(Asym,DeltaY), and Corr(DeltaX,DeltaY). See lecture notes, Workshop04.ipynb, Workshop05_optional.ipynb, or https://en.wikipedia.org/wiki/Pearson_correlation_coefficient for additional help understanding correlation coefficients. Which variables are approximately independent of each other ?

[15]:
```
#1. Read the data from the file, and plot distributions of   , Δ , and Δ .
f = open( 'asymdata.txt', 'r' )

next(f)

Asymmetry_raw = []
DeltaX = []
DeltaY = []
count = 0

for line in f:
    contents = line.split()
    Asymmetry_raw.append(float(contents[1]))
    DeltaX.append(float(contents[2]))
    DeltaY.append(float(contents[3]))
    count += 1

f.close()

import matplotlib.pyplot as plt

plt.hist(Asymmetry_raw, bins=20)
plt.title('Asymmetry_raw')
plt.show()

plt.hist(DeltaX, bins=20)
plt.title('DeltaX')
plt.show()

plt.hist(DeltaY, bins=20)
plt.title('DeltaY')
plt.show()
```


Asymmetry_raw
2000
1750
1500
1250
1000
750
500
250
0
−2000
0
2000
4000


DeltaX
2500
2000
1500
1000
500
0
−300
−200
−100
0
100
200
300



[19]:
```
#2. Compute the mean of the raw asymmetry distribution and its statistical␣
 ↪uncertainty.
print("Mean: {0:5.3f}".format(np.mean(Asymmetry_raw)))
print("Statistical Uncertainty: {0:5.3f}".format(np.std(Asymmetry_raw) / np.
 ↪sqrt(count)))
```

```
Mean: 0.443
Statistical Uncertainty: 8.489
```

[20]:
```
#3. Compute the standard deviation of the raw asymmetry distribution
#and its statistical uncertainty.
print("Standard Deviation: {0:5.3f}".format(np.std(Asymmetry_raw)))
print("Statistical Uncertainty: {0:5.3f}".format(np.std(Asymmetry_raw) / np.
 ↪sqrt(count)))
```

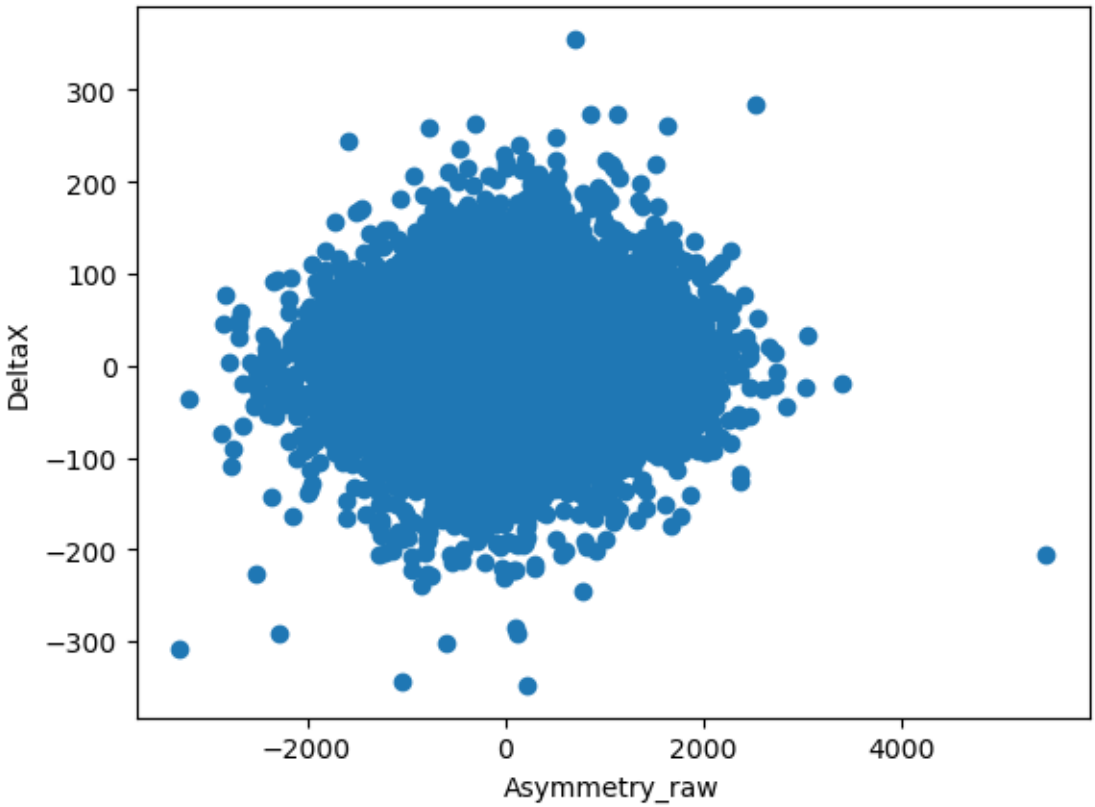
```
Standard Deviation: 848.853
Statistical Uncertainty: 8.489
```

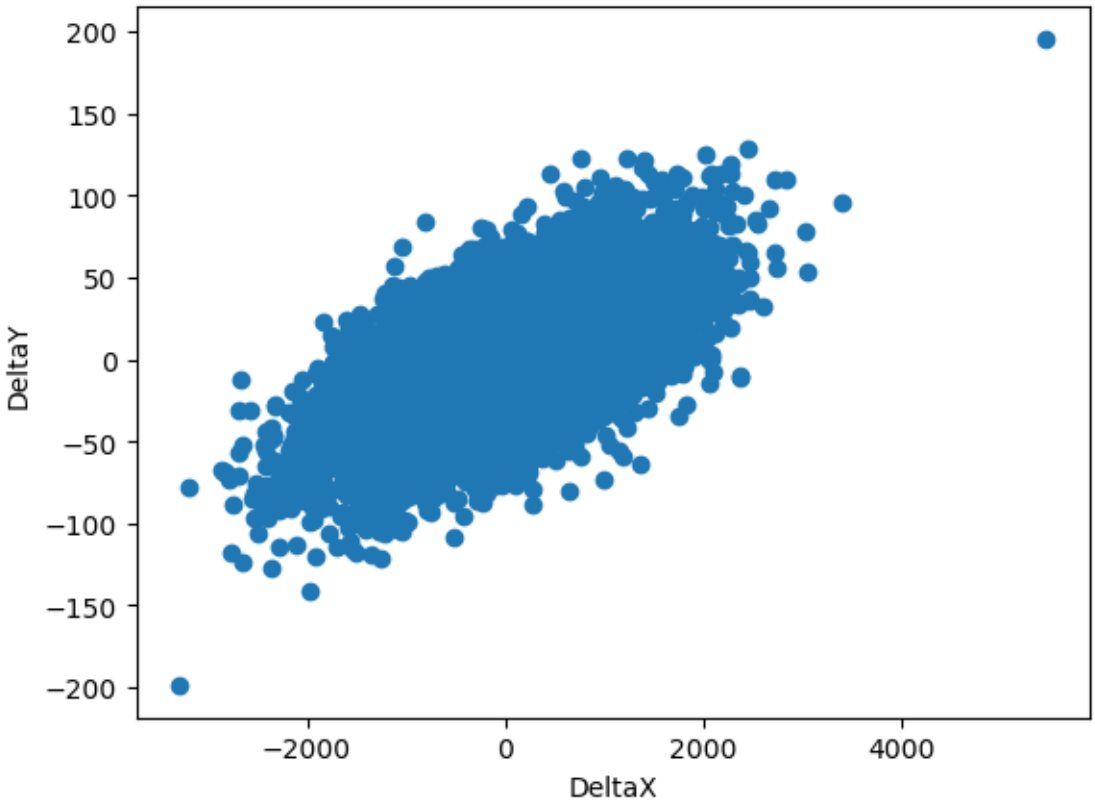
[25]:
```
#4. Compute the fraction of events contained within ±1  of the mean,
#±2  of the mean, and ±3  of the mean
#(where   is the standard deviation you computed in Part 3).
```

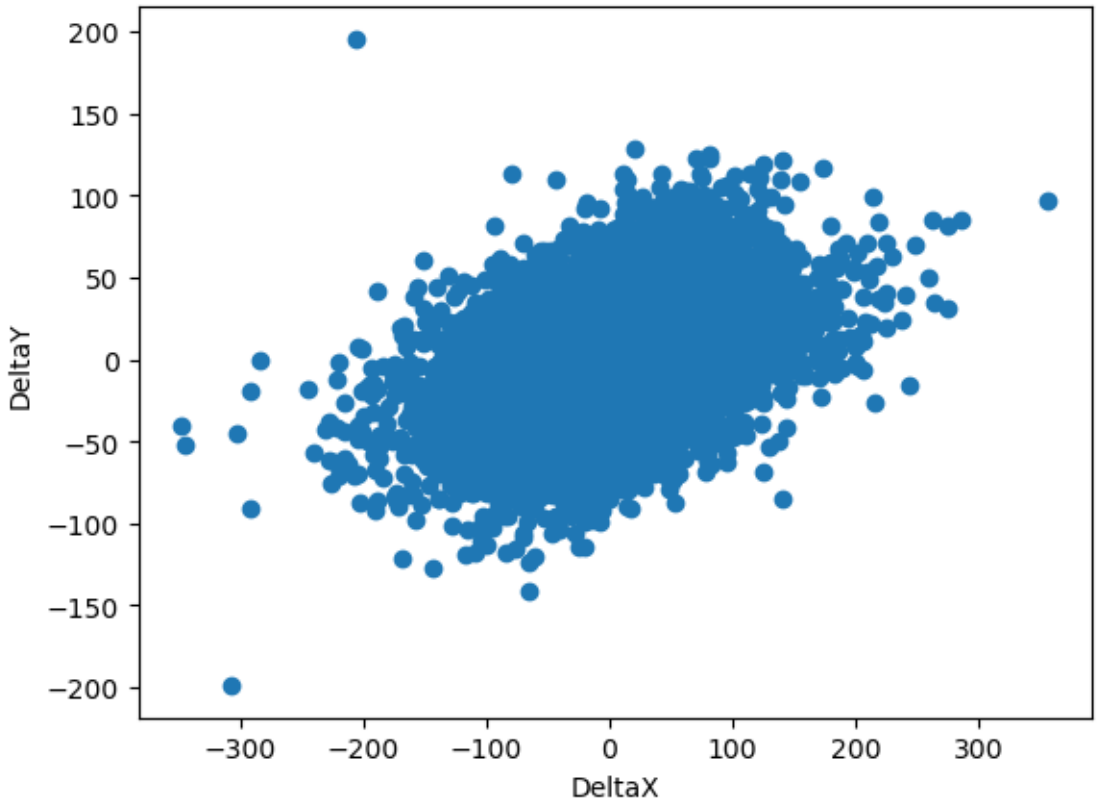
```
#Compare these fractions with the quantiles of the Gaussian distribution
sigma = np.std(Asymmetry_raw)

deviation = []
for x in Asymmetry_raw:
    abs_deviation = abs(x - np.mean(Asymmetry_raw))
    deviation.append(abs_deviation)

len_1sigma = len([x for x in deviation if x < sigma])
fraction_1sigma = len_1sigma / count
len_2sigma = len([x for x in deviation if x < 2*sigma])
fraction_2sigma = len_2sigma / count
len_3sigma = len([x for x in deviation if x < 3*sigma])
fraction_3sigma = len_3sigma / count

print("Fraction within ±1 : {:.2f}%".format(fraction_1sigma*100))
print("Fraction within ±2 : {:.2f}%".format(fraction_2sigma*100))
print("Fraction within ±3 : {:.2f}%".format(fraction_3sigma*100))
```

```
Fraction within ±1 : 68.54%
Fraction within ±2 : 95.55%
Fraction within ±3 : 99.74%
```

[26]:
```
#5. Plot    vs Δ ,    vs Δ , and Δ  vs Δ  as scatter plots.
plt.scatter(Asymmetry_raw, DeltaX)
plt.xlabel('Asymmetry_raw')
plt.ylabel('DeltaX')
plt.show()

plt.scatter(Asymmetry_raw, DeltaY)
plt.xlabel('DeltaX')
plt.ylabel('DeltaY')
plt.show()

plt.scatter(DeltaX, DeltaY)
plt.xlabel('DeltaX')
plt.ylabel('DeltaY')
plt.show()
```


300
200
100
0
−100
−200
−300
DeltaX
−2000
0
2000
4000
Asymmetry_raw


200
150
100
50
0
−50
−100
−150
−200
DeltaY
−2000
0
2000
4000
DeltaX



[35]:
```
#6. Compute the correlation coefficients Corr(Asym,DeltaX), Corr(Asym,DeltaY),
# and Corr(DeltaX,DeltaY). See lecture notes, Workshop04.ipynb,␣
 ↪Workshop05_optional.ipynb,
# or https://en.wikipedia.org/wiki/Pearson_correlation_coefficient
# for additional help understanding correlation coefficients.
# Which variables are approximately independent of each other ?
def correlation_coefficients(X, Y):
    sigma_x = np.std(X)
    sigma_y = np.std(Y)
    mu_x = np.mean(X)
    mu_y = np.mean(Y)
    R = 1 / (500 * sigma_x * sigma_y)
    R_sigma = sum([(X[i] - mu_x) * (Y[i] - mu_y) for i in range(500)])
    return R * R_sigma

print('Corr(Asym, DeltaX): {0:5.3f}'.
 ↪format(correlation_coefficients(Asymmetry_raw, DeltaX)))
print('Corr(Asym, DeltaY): {0:5.3f}'.
 ↪format(correlation_coefficients(Asymmetry_raw, DeltaY)))
print('Corr(DeltaX, DeltaY): {0:5.3f}'.format(correlation_coefficients(DeltaX,␣
 ↪DeltaY)))
```

```
Corr(Asym, DeltaX): 0.061
Corr(Asym, DeltaY): 0.677
Corr(DeltaX, DeltaY): 0.381
```

1.3 Problem 3: Gamma-ray peak

[Some of you may recognize this problem from Advanced Lab's Error Analysis Exercise. That's not an accident. You may also recognize this dataset in Homework05. That's not an accident either.]

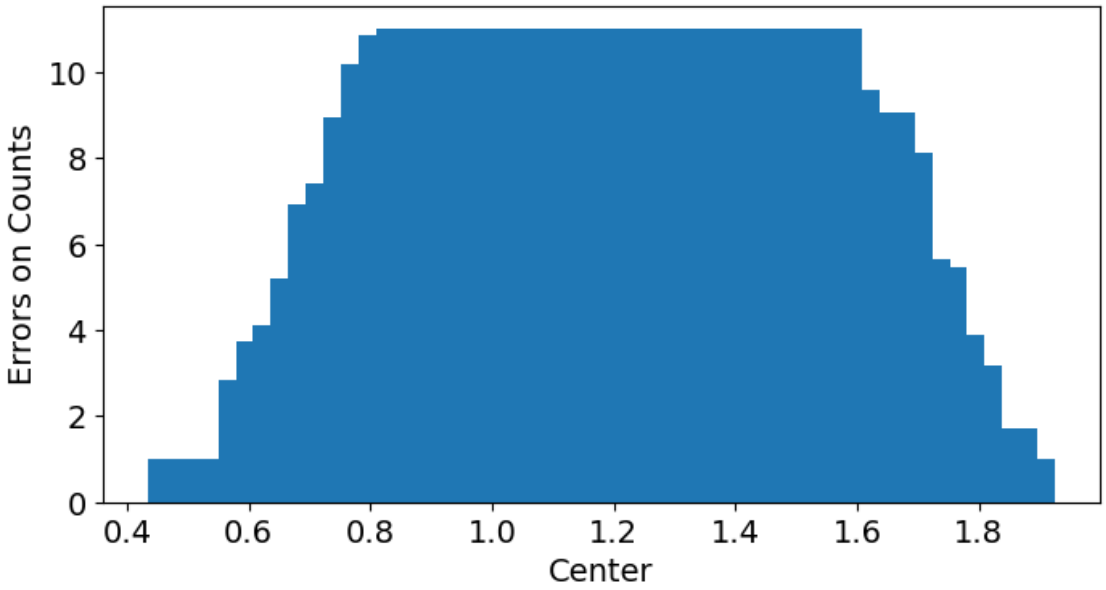
You are given a dataset (`peak.dat`) from a gamma-ray experiment consisting of ~1000 events. Each line in the file corresponds to one recorded gamma-ray event, and stores the the measured energy of the gamma-ray (in MeV). We will assume that the energies are randomly distributed about a common mean, and that each event is uncorrelated to others. Read the dataset from the enclosed file and: 1. Produce a histogram of the distribution of energies. Choose the number of bins wisely, i.e. so that the width of each bin is smaller than the width of the peak, and at the same time so that the number of entries in the most populated bin is relatively large. Since this plot represents randomly-collected data, plotting error bars would be appropriate. 1. Compute the mean and standard deviation of the distribution of energies and their statistical uncertainties. Assume the distribution is Gaussian and see the lecture notes for the formulas for the mean and variance of the sample and the formulas for the errors on these quantities. 1. Fit the distribution to a Gaussian function using an unbinned fit (Hint: use scipi.stats.norm.fit() function), and compare the parameters of the fitted Gaussian with the mean and standard deviation computed in Part 2

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats
import scipy.optimize as fitter


# Once again, feel free to play around with the matplotlib parameters
plt.rcParams['figure.figsize'] = 8,4
plt.rcParams['font.size'] = 14


energies = np.loadtxt('peak.dat') # MeV
```

Recall `plt.hist()` isn't great when you need error bars, so it's better to first use `np.histogram()` – which returns the counts in each bin, along with the edges of the bins (there are $n + 1$ edges for n bins). Once you find the bin centers and errors on the counts, you can make the actual plot with `plt.bar()`. Start with something close to `bins = 25` as the second input parameter to `np.histogram()`.

[61]:
```
# use numpy.histogram to get the counts and bin edges
counts, edges = np.histogram(energies, bins = 25)

# bin_centers = 0.5*(bin_edges[1:]+bin_edges[:-1]) works for finding the bin␣
 ↪centers
center = 0.5 * (edges[1:] + edges[:-1])
```

```
# assume Poisson errors on the counts - errors go as the square root of the count
errors = []
for c in counts:
    err = np.sqrt(c)
    errors.append(err)

# now use plt.bar() to make the histogram with error bars (remember to label the plot)
plt.bar(center, errors)
plt.xlabel('Center')
plt.ylabel('Errors on Counts')
plt.show()
```



[13]:
```
# Compute the mean and standard deviation of the list of `energies` and their uncertainties
print("Mean: {0:5.5f}".format(np.mean(energies)))
print("Standard Deviation: {0:5.5f}".format(np.std(energies)))
print("Statistical Uncertainty: {0:5.5f}".format(np.std(energies) / np.sqrt(len(energies))))
```

```
Mean: 1.20268
Standard Deviation: 0.10379
Statistical Uncertainty: 0.00328
```

You can use the list of `energies` directly as input to `scipy.stats.norm.fit()`; the returned values are the mean and standard deviation of a fit to the data.

[14]:
```
# Find the mean and standard deviation using scipy.stats.norm.fit()
# Compare these to those computed in the previous cell
```

[15]:
```
print('Mean: {0:5.5f}'.format(scipy.stats.norm.fit(energies)[0]))
print("Standard Deviation: {0:5.5f}".format(scipy.stats.norm.fit(energies)[1]))
```

```
Mean: 1.20268
Standard Deviation: 0.10379
```

[ ]:
```

```