Your name here.
Your Woskshop section here.

Homework 4: File I/O, Statistics

Comments

Each problem 10 points

Submit this notebook to bCourses to receive a grade for this Homework.

Please complete homework activities in code cells in this iPython notebook. Be sure to comment your code well so that anyone who reads it can follow it and use it. Enter your name in the cell at the top of the notebook. When you are ready to submit it, you should download it as a python notebook (click "File", "Download as", "Notebook (.ipynb)") and upload it on bCourses under the Assignments tab.

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".

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 (https://en.wikipedia.org/wiki/Central_limit_theorem#/media/File:Dice_sum_central_limit_theorem.svg))

Comments

Part 1: 1 pt

Parts 2/3: 7 pts

- generating correct sets of random numbers: 2 pts
- summing the results of each throw for N = 2,3,4,5,10: 1 pt
- histograms: 2 pts
- sample μs and σs: 1 pt
- analytic μ and σ and comparison for N = 1: 1 pt

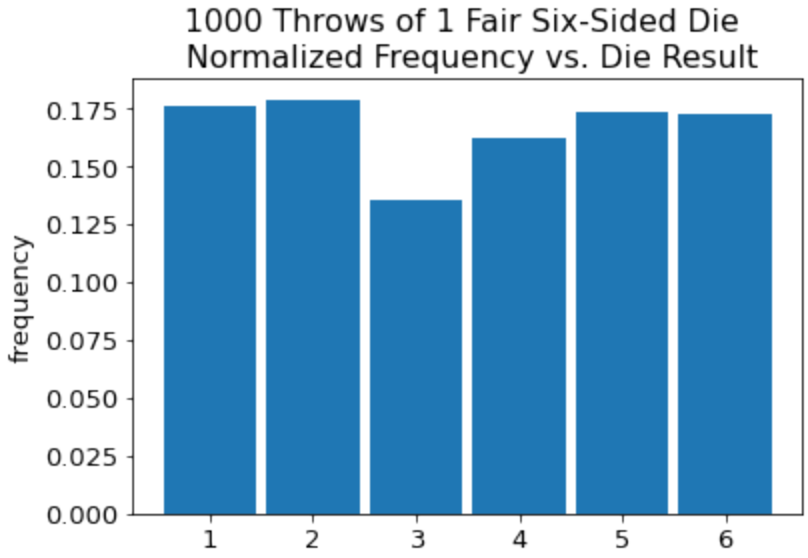
Part 4: 2 pts

- 1 pt: plot
- 1 pt: does it follow the CLT?

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 1000 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 1000 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?

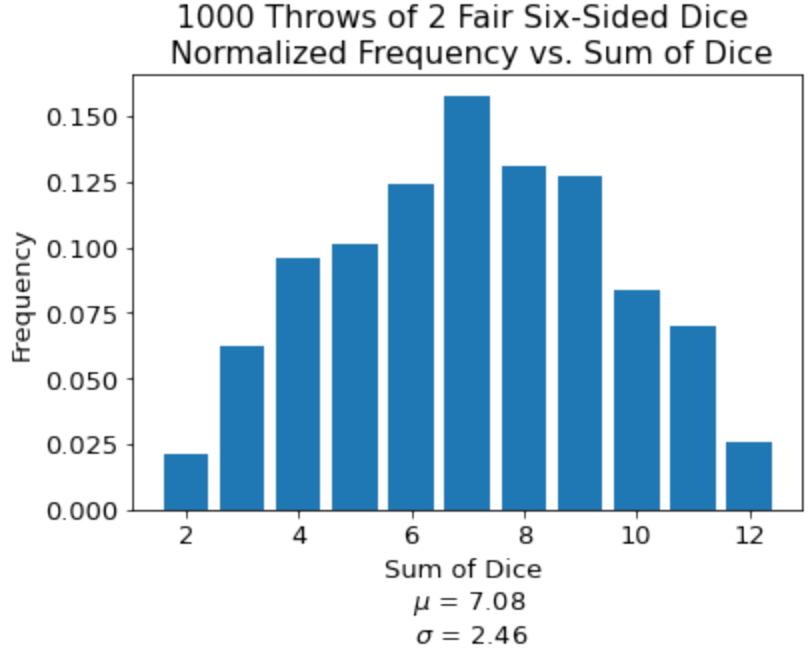
In [4]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = [6,4]
plt.rcParams['font.size'] = 13

def roll_dice(n):
    return np.random.randint(1,7,n)

thousand_rolls = roll_dice(1000)
plt.hist(thousand_rolls, bins = np.arange(.5,6.6,1), rwidth = .9, density = True)
plt.xticks(np.arange(1, 7, 1))
plt.title('1000 Throws of 1 Fair Six-Sided Die \n Normalized Frequency vs. Die Re
plt.ylabel('frequency')
plt.text(3.5, -.1, 'Die result \n \
            $\mu$ = {:.2f}, expected $\mu$ = 3.5 \n \
            $\sigma$ = {:.2f}, expected $\sigma = \sqrt{{\dfrac{{1}}{{6}}\sum_{{i

Ns = [2, 3, 4, 5, 10]
standard_deviations_of_sums = np.zeros(len(Ns))
i = 0
for N in Ns:
    sums_of_dice = np.sum(np.array([roll_dice(1000) for i in range(N)]), axis = 0
    plt.figure()
    _, bins, _ = plt.hist(sums_of_dice, density = True, bins = np.arange(N - .5,
    plt.title('1000 Throws of {:d} Fair Six-Sided Dice \n Normalized Frequency vs
    plt.ylabel('Frequency')
    plt.xlabel('Sum of Dice \n $\mu$ = {:.2f} \n $\sigma$ = {:.2f}'.format(np.mea

    standard_deviations_of_sums[i] = np.std(sums_of_dice)
    i = i + 1


plt.figure()
plt.plot(Ns, standard_deviations_of_sums, 'r*')
plt.ylabel(r'$\sigma_N$, standard deviation of sums')
plt.xlabel('N, number of dice thrown')
plt.title('1000 Throws of N Fair Six-Sided Dice \n standard deviation of 1000 thr
plt.text(0, 0.1, 'The standard deviations of the throw sums go as $\sqrt{{N}}$ (t
                  meaning that the standard deviations of the throw means go as $1
                  in accordance with the central limit theorem')
```

Out[4]:
```
Text(0, 0.1, 'The standard deviations of the throw sums go as $\\sqrt{{N}}$ (try
N = 20) \n                  meaning that the standard deviations of the throw me
ans go as $1/\\sqrt{{N}}$,\n                  in accordance with the central lim
it theorem')
```



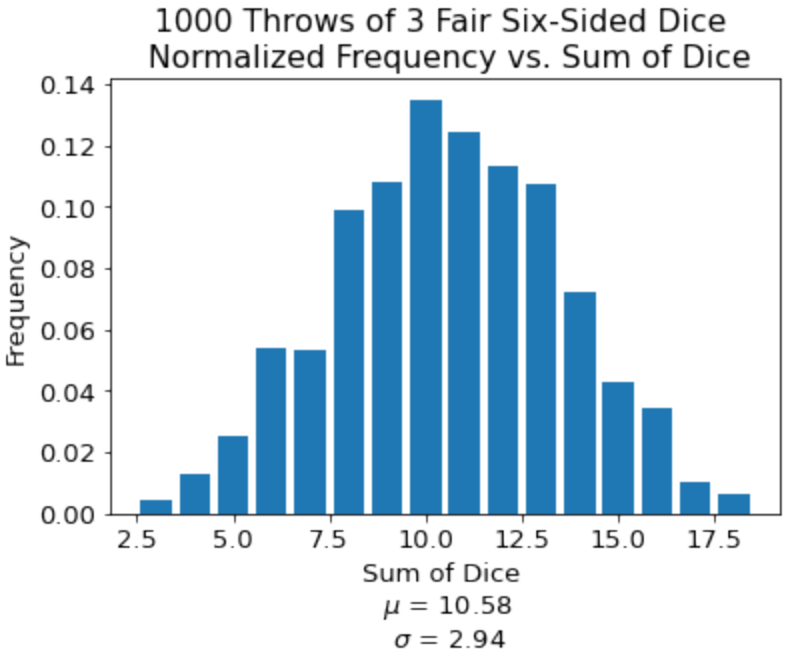
$\mu = 3.50$, expected $\mu = 3.5$

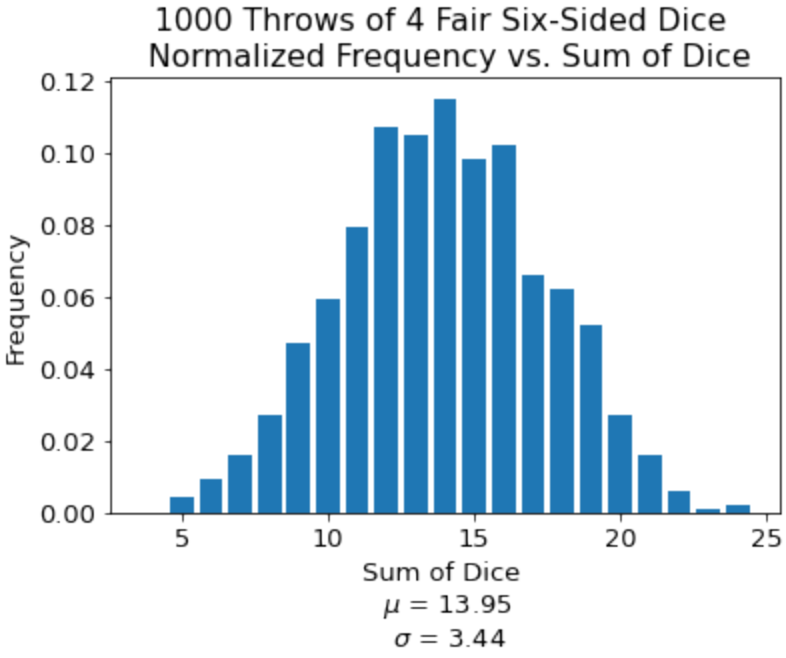
$\sigma = 1.75$, expected $\sigma = \sqrt{\frac{1}{6}\sum_{i=1}^{6}(i-3.5)^2} = 1.71$

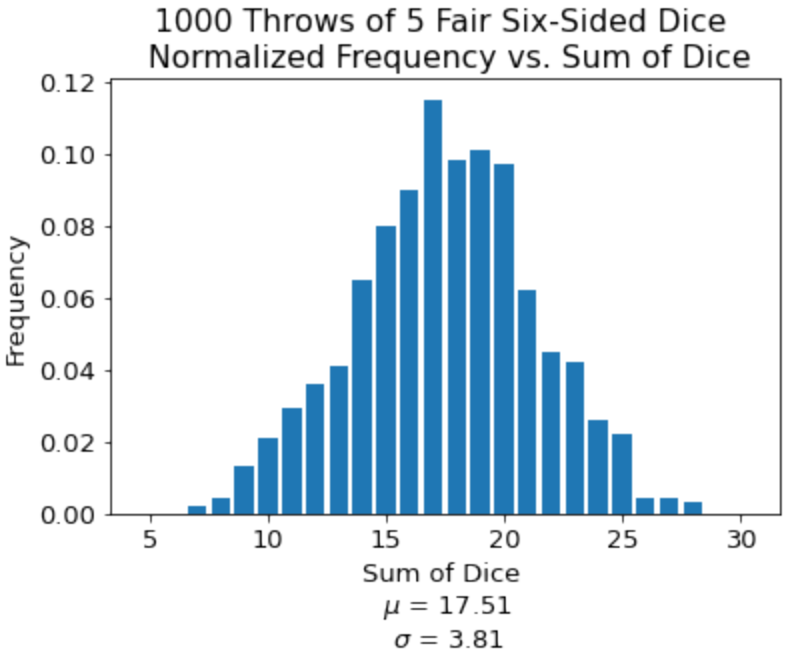


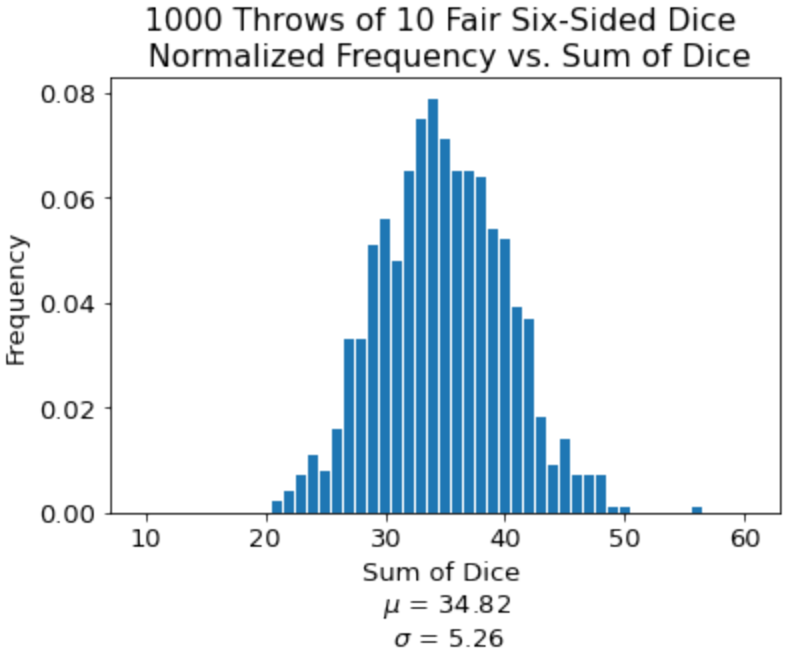
$\mu = 7.08$

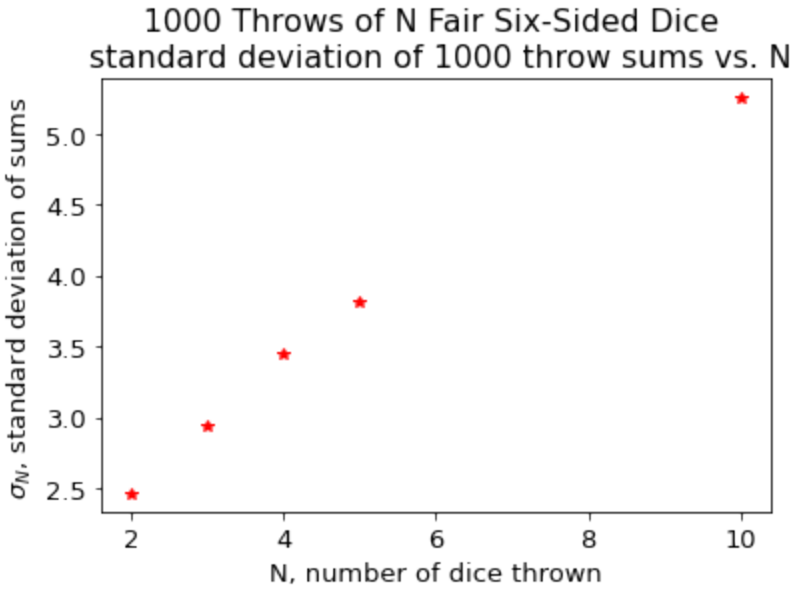
$\sigma = 2.46$


1000 Throws of 3 Fair Six-Sided Dice
Normalized Frequency vs. Sum of Dice
Frequency
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Sum of Dice
$\mu = 10.58$
$\sigma = 2.94$


1000 Throws of 4 Fair Six-Sided Dice
Normalized Frequency vs. Sum of Dice
Frequency
0.12
0.10
0.08
0.06
0.04
0.02
0.00
5
10
15
20
25
Sum of Dice
$\mu = 13.95$
$\sigma = 3.44$


1000 Throws of 5 Fair Six-Sided Dice
Normalized Frequency vs. Sum of Dice
Frequency
0.12
0.10
0.08
0.06
0.04
0.02
0.00
5
10
15
20
25
30
Sum of Dice
$\mu = 17.51$
$\sigma = 3.81$


1000 Throws of 10 Fair Six-Sided Dice
Normalized Frequency vs. Sum of Dice
Frequency
0.08
0.06
0.04
0.02
0.00
10
20
30
40
50
60
Sum of Dice
$\mu = 34.82$
$\sigma = 5.26$



The standard deviations of the throw sums go as $\sqrt{N}$ (try N = 20)
meaning that the standard deviations of the throw means go as $1/\sqrt{N}$,
in accordance with the central limit theorem

Problem 2: Parity-Violating Asymmetry

The data sample for this problem comes from the E158 (http://www.slac.stanford.edu/exp/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 (https://en.wikipedia.org/wiki/Pearson_correlation_coefficient) for additional help understanding correlation coefficients. Which variables are approximately independent of each other ?

Comments

Part 1: 3 pts

- reading in data: 1 pt
- histograms: 2 pts

Part 3: 1 pt

- computing and displaying the mean with uncertainty: 1 pt

Part 4: 1 pt

- computing and displaying the fraction of events within the given sigma deviations: 1 pt

---

Part 5: 3 pts

- each plot 1 pt

Part 6: 2 pts

- correlation coefficients: 1 pt
- which variables are independent?: 1 pt

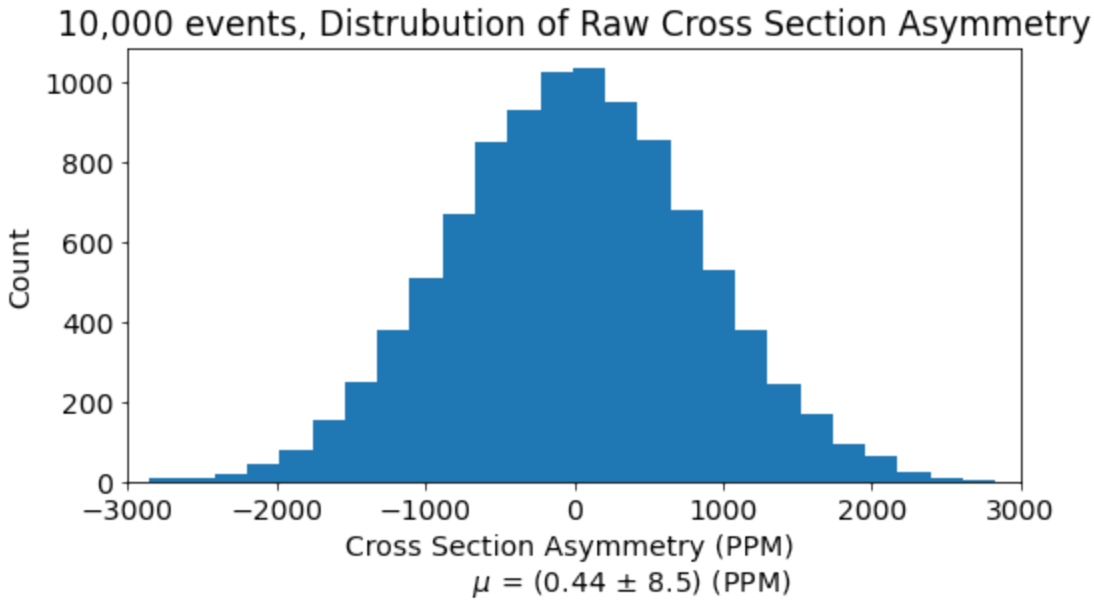
In [5]:

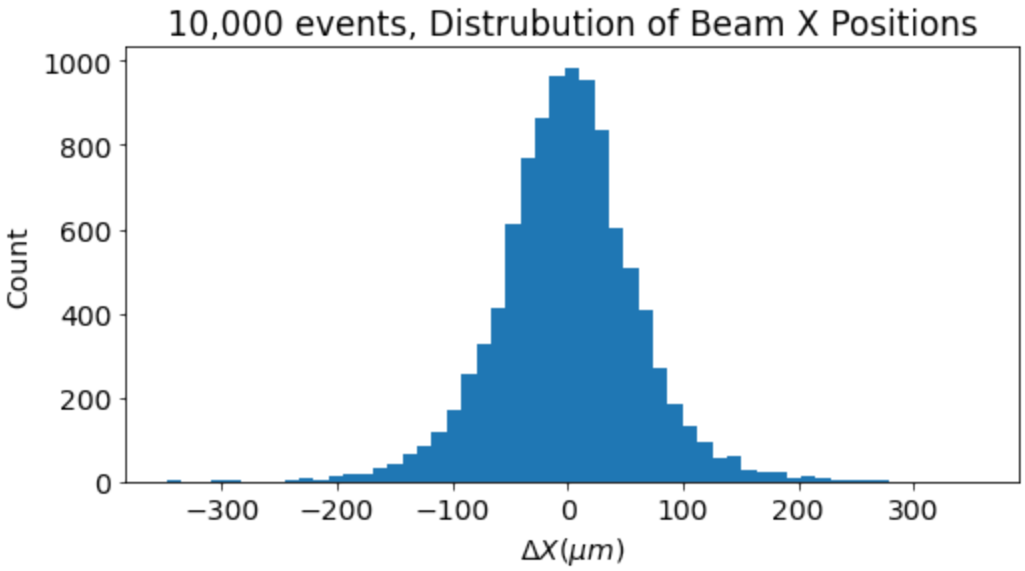
```
import matplotlib.pyplot as plt
import numpy as np

plt.rcParams['figure.figsize'] = 8,4
plt.rcParams['font.size'] = 14

#                       PPM           microns  microns
event_index, cross_section_asymmetry, deltaX, deltaY = np.loadtxt('asymdata.txt',

cross_section_asymmetry_mean = np.mean(cross_section_asymmetry)
cross_section_asymmetry_mean_err = np.std(cross_section_asymmetry)/(cross_section

plt.hist(cross_section_asymmetry, 40)
plt.title('10,000 events, Distrubution of Raw Cross Section Asymmetry')
plt.xlabel('Cross Section Asymmetry (PPM) \n \
            $\mu$ = ({:.2f} $\pm$ {:.2g}) (PPM)'.format(cross_section_asymmetry_m
plt.ylabel('Count')
plt.xlim(-3000, 3000)
plt.figure()
plt.hist(deltaX, 55)
plt.title('10,000 events, Distrubution of Beam X Positions')
plt.xlabel('$\Delta X (\mu m)$')
plt.ylabel('Count')
plt.figure()
plt.hist(deltaY, 50)
plt.title('10,000 events, Distrubution of Beam Y Positions')
plt.xlabel('$\Delta Y (\mu m)$')
plt.xlim(-150, 150)
plt.ylabel('Count')
print('Standard deviation: {:.2f}'.format(np.std(cross_section_asymmetry)))

#part 4
#event_value_list: cross_section_asymmetry
#mean_value: cross_section_asymmetry_mean
#number_of_sigma: 1, 2, or 3
#sigma_value: np.std(cross_section_asymmetry)
def get_frac_within_sigma_dev(event_value_list, mean_value, sigma_value, number_o
    counted_event_num = 0
    for i in range(len(event_value_list)):
        if (event_value_list[i] > mean_value - (number_of_sigma*sigma_value)) and
            #within the given number of sigma deviation
            counted_event_num = counted_event_num+1
    event_frac = counted_event_num/len(event_value_list)
    return event_frac

print(get_frac_within_sigma_dev(cross_section_asymmetry,cross_section_asymmetry_m
print(get_frac_within_sigma_dev(cross_section_asymmetry,cross_section_asymmetry_m
print(get_frac_within_sigma_dev(cross_section_asymmetry,cross_section_asymmetry_m


def calc_pearson_correlation_R(x, y):
    x = np.array(x)
    y = np.array(y)
    x_mean = np.mean(x)
    y_mean = np.mean(y)
    covariance_sum = 0
    for i in range(x.size):
        covariance_sum += (x[i] - x_mean)*(y[i] - y_mean)
```

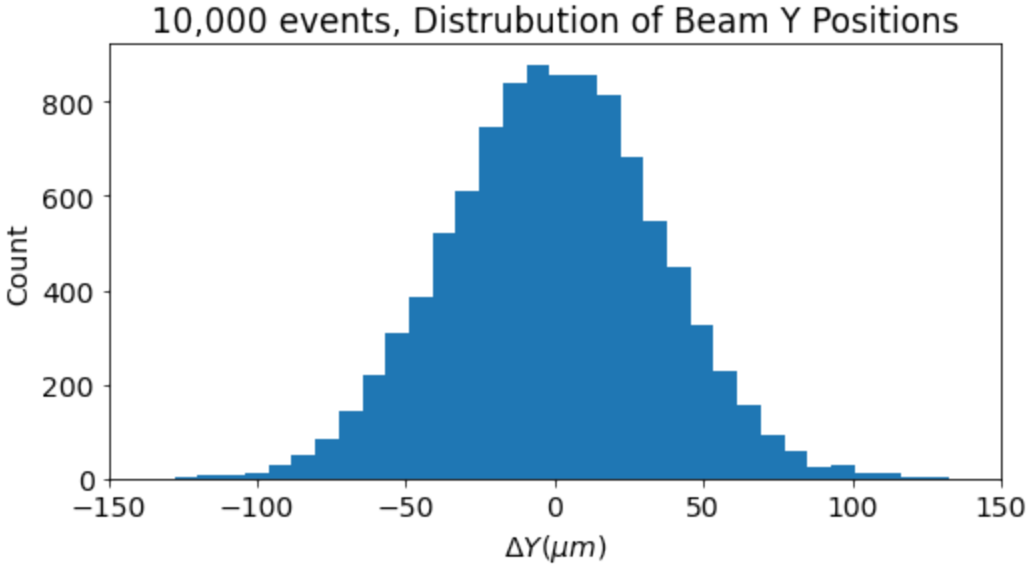
```
    return covariance_sum/(x.size * np.std(x) * np.std(y))

corr_asymm_deltaX = calc_pearson_correlation_R(cross_section_asymmetry, deltaX)
corr_asymm_deltaY = calc_pearson_correlation_R(cross_section_asymmetry, deltaY)
corr_deltaX_deltaY = calc_pearson_correlation_R(deltaX, deltaY)


plt.figure()
plt.scatter(deltaX, cross_section_asymmetry)
plt.xlabel('$\Delta X (\mu m)$')
plt.ylabel('$A_{{raw}}$ (PPM)')
plt.title('R = {:.2f}: independent'.format(corr_asymm_deltaX))
plt.figure()
plt.scatter(deltaY, cross_section_asymmetry)
plt.xlabel('$\Delta Y (\mu m)$')
plt.ylabel('$A_{{raw}}$ (PPM)')
plt.title('R = {:.2f}: correlated'.format(corr_asymm_deltaY))
plt.figure()
plt.scatter(deltaY, deltaX)
plt.xlabel('$\Delta Y (\mu m)$')
plt.ylabel('$\Delta X (\mu m)$')
plt.title('R = {:.2f}: correlated'.format(corr_deltaX_deltaY))
```

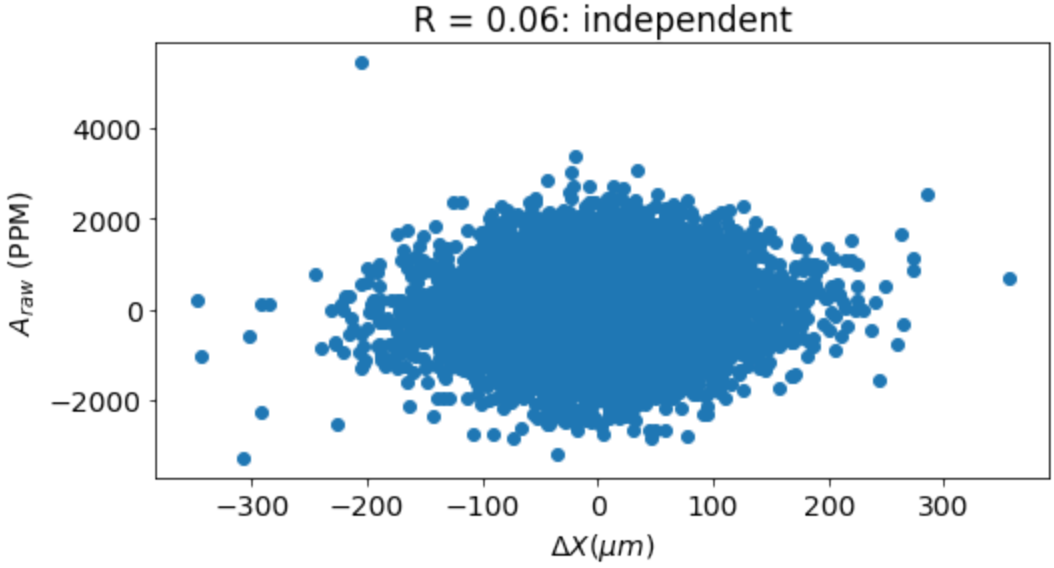
```
Standard deviation: 848.85
0.6854
0.9555
0.9974
```

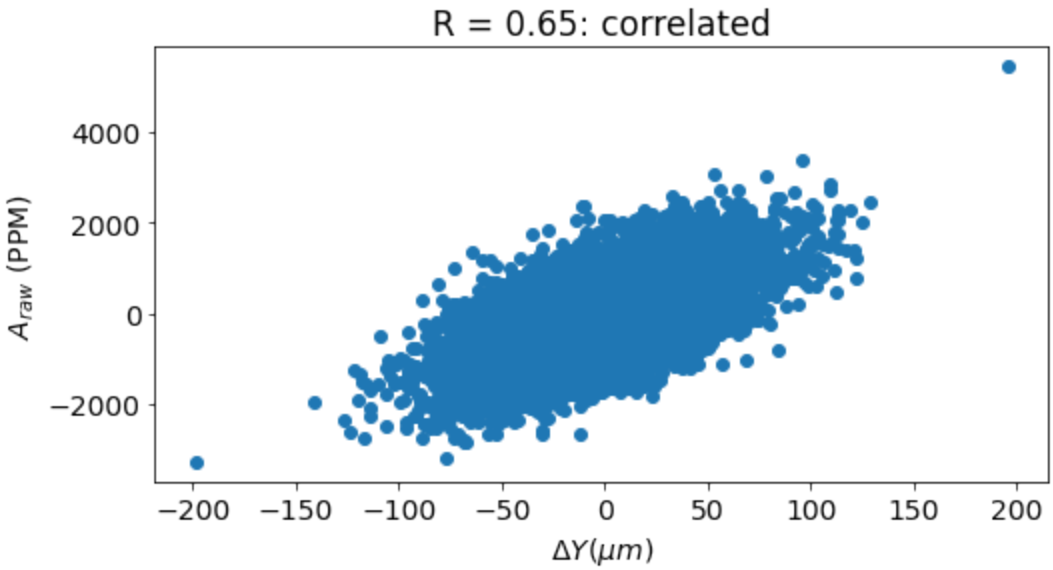
Out[5]: Text(0.5, 1.0, 'R = 0.40: correlated')

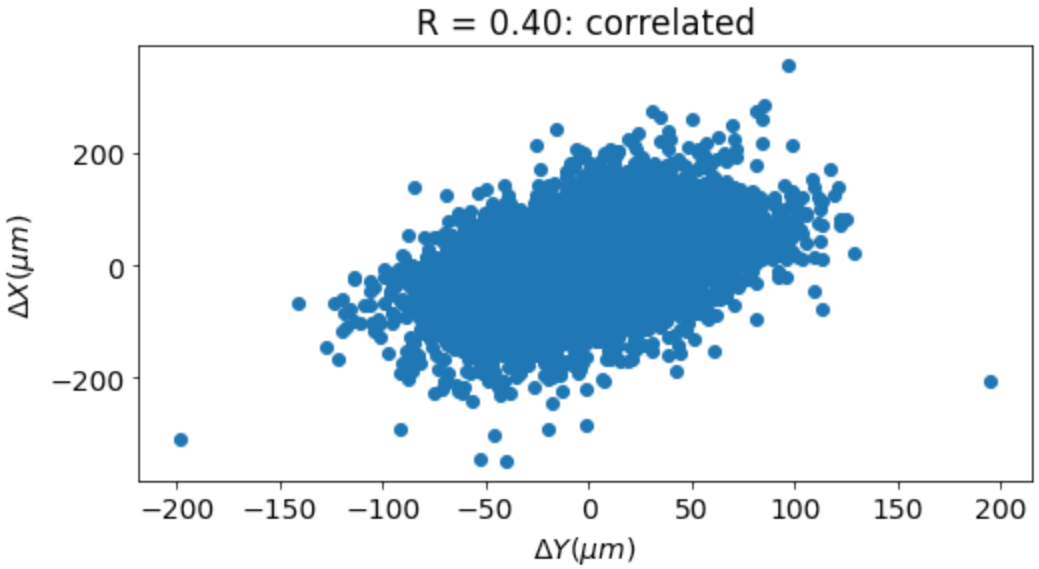



10,000 events, Distrubution of Beam X Positions
Count
1000
800
600
400
200
0
−300
−200
−100
0
100
200
300
ΔX(μm)


10,000 events, Distrubution of Beam Y Positions
Count
800
600
400
200
0
−150
−100
−50
0
50
100
150
ΔY(μm)


R = 0.06: independent
A_{raw} (PPM)
4000
2000
0
−2000
−300
−200
−100
0
100
200
300
ΔX(μm)





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 are given a dataset (`peak.dat`) from a gamma-ray experiment consisting of ~1000 gamma-ray hits. 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.
2. 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.
3. 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.

Comments

Part 1: 4 pts

- basic histogram: 2 pts
- error on count = sqrt(count): 1 pt
- plotting the count errors: 1 pt

Part 2: 3 pts

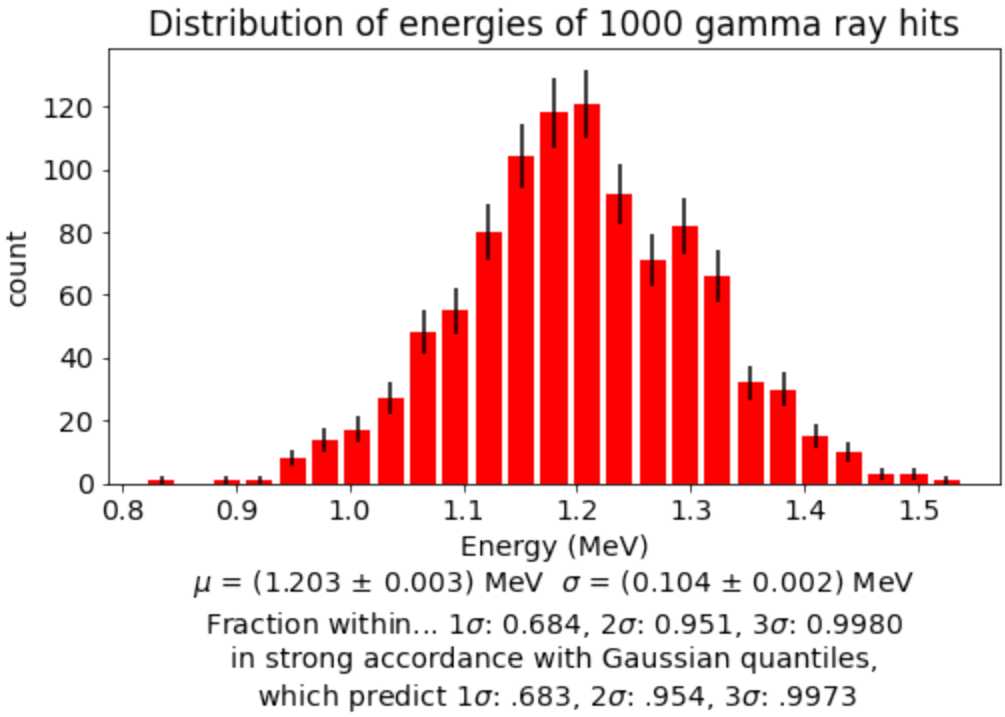
- mean and error on mean: 2 pts
- stddev and error on stddev: 1 pts

Part 3: 3 pts

- fit distribution to unbinned Gaussian function: 1 pts
- comparison with Gaussian quantiles: 2 pts

In [6]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = 8,4
plt.rcParams['font.size'] = 14


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


# use numpy.histogram to get counts and bins without actually plotting
counts, bins1 = np.histogram(energies, bins = 25)


bincenters = 0.5*(bins1[1:]+bins1[:-1])

# assume Poisson errors on the counts — errors go as the square root of the count
counts_errors = np.sqrt(counts)
width = (bins1[1] - bins1[0]) * .8
plt.bar(bincenters, counts, width=width, color='r', yerr=counts_errors)

mean = np.mean(energies)
stddev = np.std(energies)

N = energies.size
err_on_mean = stddev/N**(1/2)

# see slide 43 lec05_stats
err_on_stddev = stddev/(2 * N)**(1/2)

frac_within_one_stddev = np.sum(np.all([energies > mean - stddev, energies < mean
frac_within_two_stddev = np.sum(np.all([energies > mean - 2 * stddev, energies <
frac_within_three_stddev = np.sum(np.all([energies > mean - 3 * stddev, energies

plt.xlabel('Energy (MeV)\n$\mu$ = ({:.3f} $\pm$ {:.3f}) MeV\t $\sigma$ = ({:.3f}
           'Fraction within... 1$\sigma$: {:.3f}, 2$\sigma$: {:.3f}, 3$\sigma$: {
           'in strong accordance with Gaussian quantiles,\n which predict 1$\sigm

plt.title('Distribution of energies of 1000 gamma ray hits')
plt.ylabel('count')
```

Out[6]:
```
Text(0, 0.5, 'count')
```


Distribution of energies of 1000 gamma ray hits
count
Energy (MeV)
μ = (1.203 ± 0.003) MeV σ = (0.104 ± 0.002) MeV
Fraction within... 1σ: 0.684, 2σ: 0.951, 3σ: 0.9980
in strong accordance with Gaussian quantiles,
which predict 1σ: .683, 2σ: .954, 3σ: .9973

In [ ]: