Problem Set 3

Question 1: Relativistic Expressions for Accelerator Performance (10 points)

Learning objectives

In this question you will:

- Derive the expressions for the center-of-mass energy at colliders and in fixed target experiments
- Show that when the masses of the beam and target can be neglected, the expressions reduce to a simple form

In the pdf file for Lecture 6, page 6 defines the configuration of the beams for colliding beam and fixed target experiments and gives expressions for the center-of-mass energy $\sqrt{s}$ in the limit where the mass of the beam particle(s) is small compared to its energy.

1a.

Explicitly derive the expressions for the center-of-mass energy for the collider and fixed target case NOT assuming that the masses of the colliding particles are small.

In general,

$$s = (p_1 + p_2)^2 = (\sum_{i=1}^{2} E_i)^2 - (\sum_{i=1}^{2} p_i)^2$$

Note: p is four vector, $\vec{p}$ is 3-vector
Fixed target case:

$$s_{fixed} = (E_{beam} + m_{targ})^2 - \overrightarrow{p_{tot}}^2$$
$$s_{fixed} = (E_{beam}^2 + 2m_{targ}E_{beam} + m_{targ}^2) - (E_{beam}^2 - m_{targ}^2)$$
$$s_{fixed} = 2m_{targ}^2 + 2m_{targ}E_{beam}$$
$$E_{cm}^{fixed} = \sqrt{s_{fixed}} = \sqrt{2m_{targ}^2 + 2m_{targ}E_{beam}}$$

Collider Case:

$$s_c = (p_1 + p_2)^2$$
$$s_c = p_1^2 + p_2^2 + 2p_1p_2$$
$$s_c = m_1^2 + m_2^2 + 2(E_1E_2 + 2\overrightarrow{p_1} \cdot \overrightarrow{p_2})$$
$$s_c = m_1^2 + m_2^2 + 2(E_1E_2 + 2(E_1 - m_1)(E_2 - m_2))$$
$$s_c = m_1^2 + m_2^2 + 2E_1E_2 + 2E_1E_2 - 2m_2E_1 - 2m_1E_2 + 2m_1m_2$$
$$E_{cm}^{collider} = \sqrt{s_c} = \sqrt{m_1^2 + m_2^2 + 4E_1E_2 - 2m_2E_1 - 2m_1E_2 + 2m_1m_2}$$

1b.

Using your result from 1a and appropriate approximations, derive the expressions quoted in class:

$$E_{cm}^{collider} = \sqrt{4E_1E_2} \text{ and } E_{cm}^{fixed\ target} = \sqrt{2m_{target}E_{beam}}$$

Using the results from 1a, approximate the masses of the particles as small $(m << 1)$. Then:
Fixed: $m^2 << m$

$$E_{cm}^{fixed} = \sqrt{2m_{targ}^2 + 2m_{targ}E_{beam}} \approx \sqrt{2m_{targ}E_{beam}}$$

Colldier: $m_i << E_i$

$$E_{cm}^{collider} = \sqrt{m_1^2 + m_2^2 + 4E_1E_2 - 2m_2E_1 - 2m_1E_2 + 2m_1m_2} \approx \sqrt{4E_1E_2}$$

Question 2: Cherenkov Imaging (10 points)

Learning objectives

In this question you will:

- Review the expression that determines when Cherenkov light is emitted and how the angle of the Cherenkov cone is related to a particle's speed and the index of refraction of the medium
- Apply the concepts of resolution and of confidence level to the case of a Cherenkov detector

In BaBar, the DIRC detector identifies charged particles (eg distinguishes $\pi^{\pm}$ from $K^{\pm}$) by reconstructing Cherenkov rings produced in a thin quartz bar. The index of refraction of quartz is n = 1.2.

2a.

What is the minimum momentum below which K^{+} do not produce Cherenkov light?

In [1]:
```
import numpy as np
```

In [4]:
```
# according to PDG:
# mK == mass of charged K
mK = 0.493677 #GeV/c^2
# minimum velocity for Cherenkov light: vp >= c/n
n = 1.2
c = 1 # units of c established
vp = c/1.2
# p = gamma m v
beta = vp/c
gamma = 1/(np.sqrt(1 - (beta)**2))

pmin = gamma*mK*vp # units: GeV/c^2 * c = GeV/c
print('Minimum momentum =', pmin,'GeV/c')
```

```
Minimum momentum = 0.7442460802855831 GeV/c
```

2b

A charged particle is observed to have a momentum of 2.5 GeV. What are the predicted angles of the Cherenkov cone produced if the particle is a $\pi^{\pm}$ and if it is a $K^{\pm}$?

In [17]:
```
# 2.5 GeV/c = p = gamma m v
# gamma is a function of v; rearranging to solve for v:
def get_v(p, m):
    c = 1
    factor = 1 + (p**2)/((m**2)*(c**2)) # unitless factor to be divided
    return np.sqrt(((p**2)/(m**2))/factor) # checked units: returns in
units of c, which is perfect

pcharged = 2.5 # GeV/c

# K+/-

vK = get_v(pcharged, mK)
betaK = vK/c # which is the same number, but accounts for units properl
y (beta = unitless)

# pi+/-

mpi = 0.139570 # GeV/c^2
vpi = get_v(pcharged, mpi)
betapi = vpi/c

# emission angle formula: cos(theta) = 1/(n*beta)
angleK = np.arccos(1/(n*betaK))
anglepi = np.arccos(1/(n*betapi))

print('If pion:', anglepi, 'rad')
print('If Kaon:', angleK, 'rad')
```

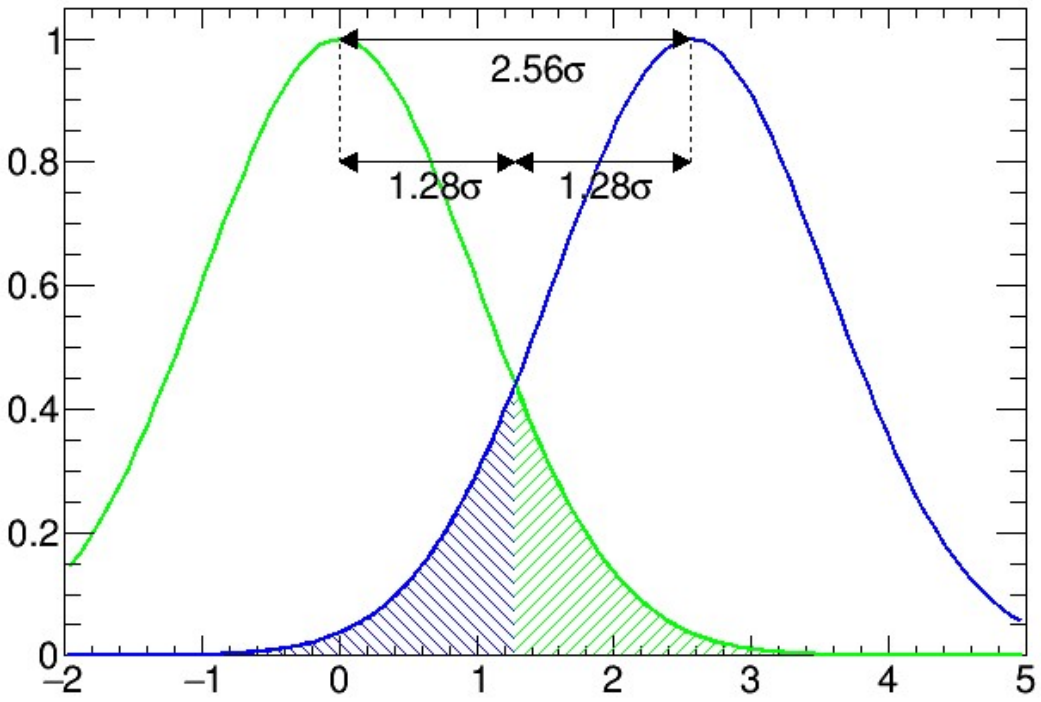
```
If pion: 0.5833338498396993 rad
If Kaon: 0.5559001794626739 rad
```

2c

The angular resolution for the Cherenkov cone of the DIRC detector is approximately $\sigma(\theta_c) = 2.5$~mrad. This means that the measured Cherenkov angle distributed about the true value such that

$$\theta_c^{meas} - \theta_c^{true} = \frac{1}{\sigma_c\sqrt{2\pi}}\exp\left[-\frac{1}{2\sigma^2}\right]$$

As the charged particle momentum increases, the Cherenkov angles for $\pi^{\pm}$ and $K^{\pm}$ of the same momentum converge towards the same value. The largest momentum for which the two hypothesis can be distinguished depends on σ_c. Scientists use the term confidence level to describe how well we can tell two hypothesis apart A 90\%\ confidence level separation means that it we start with equal populations in the two possibilities, we will choose the wrong hypothesis 10\%\ of the time. See, for example, the PDG review of statistics Figure 40.4 and Table 40.1. From that table, you learn that a 90\%\ confidence level separation corresponds to 1.64σ for a two-sided cut. Here, however, we are doing something slightly different. If we have two Gaussians where one is centered at $\theta = 0$ and the other is centered at $\theta = n\sigma$ then the natural way to assign particles by species is to say all the measurements with $\theta > n\sigma/2$ belong to the Gaussian centered at $\theta = n\sigma$ and the others belong to the Gaussian centered at $\theta = 0$. The probability of getting the wrong assignment with this strategty is called a one-sided confidence limit. In our case, we must ask that 10% of the events be in one tail (so 20% in both tails). That corresponds to having the distance be 1.28σ from each peak, or 2.56 σ between the peaks, as shown here:



Here, the hatched regions correspond to the cases where the species are misidentified.

Using this definition of confidence level, what is the maximum momentum for which π^+ and K^+ can be separated at the 90\%\ confidence level in the BaBar DIRC?

In [28]:
```
sigmac = 2.5/1000 # units of rad
diff = sigmac*2.56 # minimum difference for 90% confidence interval

# scan through possible p values to find the best one that matches the
requirements
def get_max_p(scanmin, scanmax, difference,m1,m2,n):
    c = 1
    ps = np.linspace(scanmin, scanmax,1000)
    success = []
    for p in ps:
        v1 = get_v(p, m1)
        beta1 = v1/c
        v2 = get_v(p, m2)
        beta2 = v2/c

        angle1 = np.arccos(1/(n*beta1))
        angle2 = np.arccos(1/(n*beta2))
        current_diff = np.abs(angle1-angle2)
        #print(current_diff)

        if current_diff >= difference:
            success.append(p)

    success = np.array(success)
    #print(success)
    maxp = np.max(success)

    return maxp

maxvalue = get_max_p(1,10,diff,mK,mpi,n)

print('Maximum momentum that maintains 90% confidence level:',maxvalu
e,'GeV/c')
```

```
Maximum momentum that maintains 90% confidence level: 5.1441441441441
444 GeV/c
```

Question 3: Luminosity of a Collider (10 points)

(Thomson problem 1.11) At the LEP e^+e^- collider, which had a circumference of 27~km, the electron and positron beam currents were each 1.0 mAmps. Each beam consisted of four equally spaced bunches of electrons/positrons. The bunches had an effective area of 1.8×10^4 μm. Calculate the instantaneous luminosity on the assumption that the beams collided head-on.

In [44]:
```
nbunches = 4
Aeff = 1.8 * (10**4) # units: micrometers^2
I = 0.001 #amps
# current density
j = I/Aeff # amps/um^2 = C/s*um^2
# but also j = q*(rho)*v (where v = c in this case)
# we're looking for rho, the numberdensity, so rho = I/A*c*q
q = 1.602*(10**(-19)) # Coulombs
actualc = 3*(10**8) # m/s
# converting to meters^2
Aeffm2 = Aeff * (10**(-12))

rho = I/(Aeffm2*actualc*q)
# N = rho * volume, volume = Aeff * length (27km)
V = Aeffm2*27000
N = V*rho

# frequency = 1 collision/ time, where time
time = 27*1000/actualc # seconds
freq = 1/time # 1/seconds, or Hz

# finally, for L:
L = (freq * nbunches * N * N)/(4*np.pi*Aeffm2)
print("Luminosity:", L, "m^-2 s^-1")
```

```
Luminosity: 6.201476583261849e+34 m^-2 s^-1
```

Question 4: Central Limit Theorem (30 points)

Learning objectives

In this question you will:

- See that you can use σ as a measure of resolution even for distributions that are not Gaussian
- Learn how to write a simple Monte Carlo and use it to reproduce an analytical result

The *Central Limit Theorem* (https://en.wikipedia.org/wiki/Central_limit_theorem) tells us that the distribution of the sum (or average) of a large number of independent, identically distributed measurements will be approximately normal, regardless of the underlying distribution (subject to the condition that mean and variance of the underlying distribution are not infinite). We'll see how this works for the simplest pdf (probability density function (https://www.khanacademy.org/math/statistics-probability/random-variables-stats-library/random-variables-continuous/v/probability-density-functions)), a random variable x uniformly distributed:

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

4a.

Show that the mean $\mu \equiv \int_{\infty}^{\infty} x f(x) dx$ is $\frac{b+a}{2}$ and the variance $\sigma^2 \equiv \int_{\infty}^{\infty} x^2 f(x) dx - \mu^2$ is $\frac{(b-a)^2}{12}$ for the above distribution.

The bounds can be simplified by the "f = 0 if otherwise" constraint. Therefore, given b > a,

$$\mu = \int_{\infty}^{\infty} x f(x) dx = \int_{a}^{b} x \frac{1}{b-a} dx = \frac{x^2}{2} \frac{1}{b-a}$$

Evaluating at $x = b$ and $x = a$ and then taking the difference, we have:

$$\mu = \frac{b^2 - a^2}{2(b-a)} = \frac{b+a}{2}$$

Using similar logic for variance:

$$\sigma^2 = \int_{\infty}^{\infty} x^2 f(x) dx - \mu^2 = \int_{a}^{b} x^2 \frac{1}{b-a} dx - \frac{(b+a)^2}{4}$$

$$\sigma^2 = \frac{(b^3 - a^3)}{3(b-a)} - \frac{(b+a)^2}{4} = \frac{a^2 + ab + b^2}{3} - \frac{a^2 + 2ab + b^2}{4}$$

$$\sigma^2 = \frac{a^2 - 2ab + b^2}{12}$$

$$\therefore \sigma^2 = \frac{(b-a)^2}{12}$$

4b.

Let $a = 0$ and $b = 1$. Using your favorite random number generator, generate 1000 random numbers from the uniform distribution, $f(x)$. Calculate the mean and variance of the numbers you generate. Hint: You can use the NumPy (Numerical Python) library to generate random numbers from the (0,1) uniform distribution:

In [45]:
```
#Import the NumPy library as "np"
import numpy as np

#Use NumPy to generate a list (it's actually a numpy.ndarray) of 1000 r
andom numbers
samples = np.random.rand(1000)

#Print the first 10 random numbers
print(samples[0:10])
```

```
[0.70398486 0.52604354 0.07593052 0.94666912 0.38158277 0.73864901
 0.56360295 0.25830431 0.88836987 0.24791697]
```

Write your own functions to find the mean and variance of a list of numbers:

In [53]:
```
# I assume I was supposed to write my own rather than just slot in "np.
mean" and "np.var" here, so

def find_mean(num_list):

    length = len(num_list)
    total = np.sum(num_list)
    mean = total/length

    return mean

def find_variance(num_list):

    length = len(num_list)
    mean = find_mean(num_list)
    diffs = []
    for item in num_list:
        diffs.append((item - mean)**2)

    numerator = np.sum(diffs)

    return numerator/(length - 1)
```

In [54]:
```
np.var(samples)
```

Out[54]:
```
0.07975816222110041
```

In [55]:
```
# Compare between my function and numpy's:

print("Numpy result for mean:",np.mean(samples))
print("My function result for mean:",find_mean(samples))

print("Numpy variance:", np.var(samples))
print('My function result for variance:', find_variance(samples))
```

```
Numpy result for mean: 0.5084939900094709
My function result for mean: 0.5084939900094709
Numpy variance: 0.07975816222110041
My function result for variance: 0.07983800022132173
```

4c.

Do these numerical results agree with the analytical results you found above?

In [56]:
```
# Analytical results:
a = 0
b = 1
Amean = (b+a)/2
Avar = (b-a)/12

print("Analytical mean:", Amean)
print("Analytical var:", Avar)

print("The numerical results agree with the analytical results well; th
ey will get closer to the analytical result as the number of samples in
creases")
```

```
Analytical mean: 0.5
Analytical var: 0.08333333333333333
The numerical results agree with the analytical results well; they wi
ll get closer to the analytical result as the number of samples incre
ases
```

4d.

Make a histogram with 100 bins where the lower edge of the first bin is at $x = 0$ and the upper edge of the last is at $x = 1$. Fill your histogram with the random numbers you generated above.

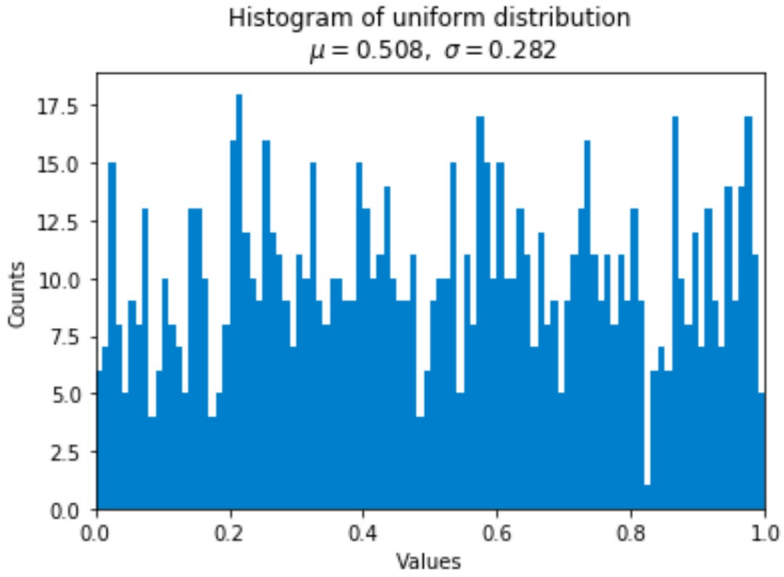
We'll use the **matplotlib.pyplot** module to make several histograms throughout the assignment, so we should go ahead and import it now. Making a histogram from a list of numbers is as simple as calling plt.hist() (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html), with the list as the input parameter. There are many optional parameters, like the number of bins and the color of the bars.

Since our histograms will share similar formatting, it's also useful to define a function rather than typing the same thing over and over. For this homework, we'll supply the histogramming routine. In the future, you will write this code yourself.

In [63]:
```
#Import the pyplot module of matplotlib as "plt"
import matplotlib.pyplot as plt


#Makes a histogram filled with the random numbers we generate
def plot_histogram(samples,xtitle,ytitle, title, limits):

    #It would be nice to have the mean and standard deviation in the ti
tle, so let's get these
    mean, sigma = np.mean(samples), np.sqrt(np.var(samples))
    #Plot the histogram of the sampled data with 100 bins and a nice co
lor
    plt.hist(samples, bins=100, range=limits, color=(0,0.5,0.8))  #Set
the color using (r,g,b) values or
                                                                  #  us
e a built-in matplotlib color"""

    #Add some axis labels and a descriptive title
    plt.xlabel(xtitle)
    plt.ylabel(ytitle)
    plt.title(title+'\n $\mu={0:.3f},\ \sigma={1:.3f}$'.format(mean,sig
ma) )

    #Get rid of the extra white space on the left/right edges (you can
delete these two lines without a problem)
    xmin, xmax, ymin, ymax = plt.axis()
    plt.axis([limits[0],limits[1],ymin,ymax])

    #Not necessarily needed in a Jupyter notebook, but it doesn't hurt
    plt.show()

    return mean, sigma
```

Take some time to play with the plot formatting and choose a color (https://matplotlib.org/2.0.2/api/colors_api.html) you like for the bars. Then call the function with your random samples.

In [64]:
```
plot_histogram(samples, xtitle = 'Values', ytitle = 'Counts', title='Histogram of uniform distribution',
               limits = [0,1])
```



Out[64]: (0.5084939900094709, 0.28241487606197446)

4e.

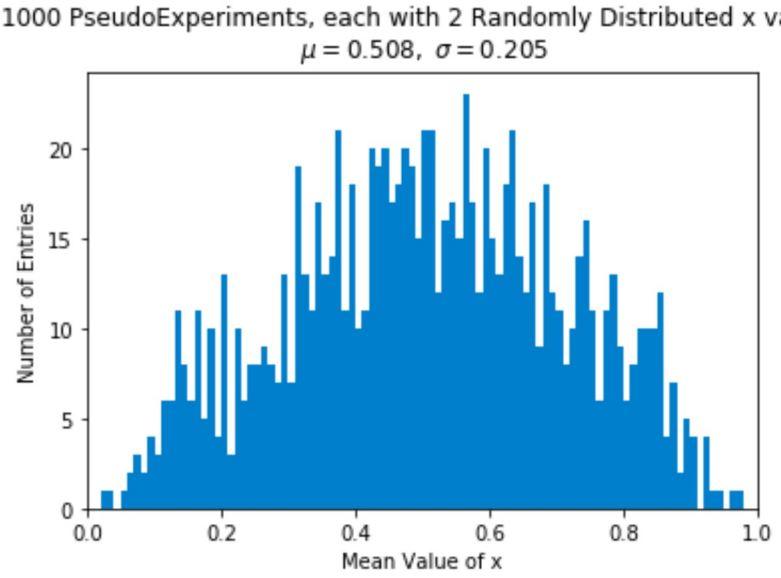
Now suppose you make an ensemble of 1000 pseudoexperiments where each pseduoexperiment consists of N uniformly distributed random numbers. For each pseudoexperiment, define the measurement S to be

$$S \equiv \frac{1}{N}\sum_{1}^{N} x_i$$

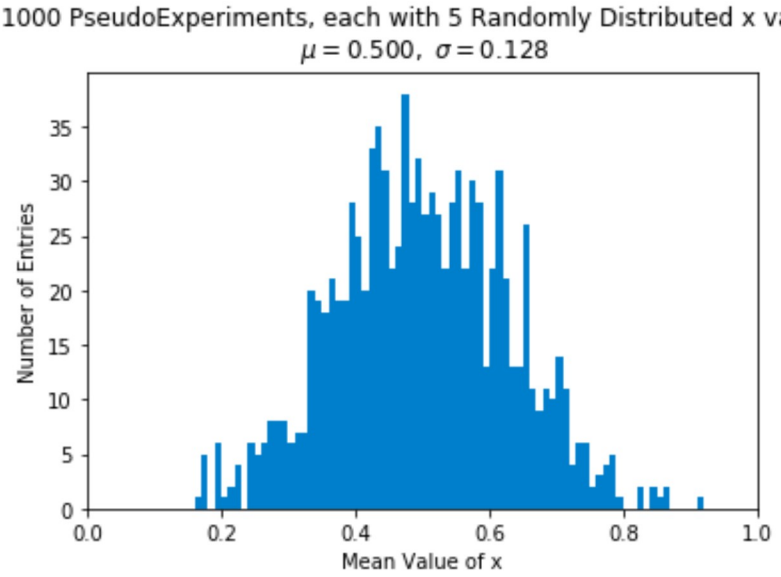
Make histograms of S with the same $x-$axis as above for the cases $N = 2$, $N = 5$ and $N = 10$. Determine the mean and the σ of the distributions displayed in these histograms.

In [66]:
```
#Calculates and returns S as defined above
def pseudoexperiment(N):
    samples = np.random.rand(N)  #samples = [x_1, x_2, ... x_N]

    # s = 0  replace this with your calculation of s

    s = np.sum(samples)/N

    return s

#Performs each pseudoexperiment 1000 times and plot a histogram of the
results
def run_pseudoexperiments(N):
    s_list = []
    #run the ensemble of 1000 pseudoexperiments and store measurements
    for i in range(1000):
        s_list.append(pseudoexperiment(N))
    #plot a histogram of these 1000 mesa
    mean, sigma = plot_histogram(s_list,"Mean Value of x","Number of En
tries","1000 PseudoExperiments, each with "+str(N)+" Randomly Distribut
ed x values",[0.0,1.0])

    return mean,sigma
```

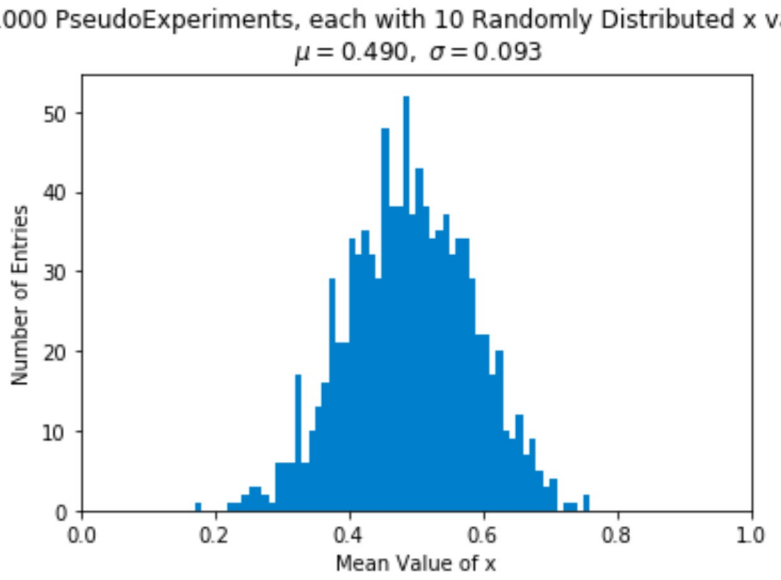
In [69]:
```
# not sure if it was required to print out the mean and sigma since plo
t_histogram does that already
# but I added it in the functions just in case
means =[]
stdevs = []
for N in [2,5,10]:
    m,s = run_pseudoexperiments(N)
    print('Mean:', m)
    print("Sigma:", s)
    means.append(m)
    stdevs.append(s)
```




```
Mean: 0.5077287318570084
Sigma: 0.2048964751493592
```




```
Mean: 0.49997242480791126
Sigma: 0.12827138046413356
```




```
Mean: 0.4903760529830898
Sigma: 0.0930375328186929
```

In each case, compare the σ you obtain to what you would predict if you assumed the experiments followed a normal distribution.

If we assumed that these experiments followed a normal distribution, they would follow exactly these $sigma$ values. This is the main result of the Central limit theorem, which we just demonstrated above.

In [74]:
```
n = [2,4,5]
for i in range(len(n)):
    print('Sample size:', n[i])
    print('Corresponding sigma:', stdevs[i])
```

```
Sample size: 2
Corresponding sigma: 0.2048964751493592
Sample size: 4
Corresponding sigma: 0.12827138046413356
Sample size: 5
Corresponding sigma: 0.0930375328186929
```

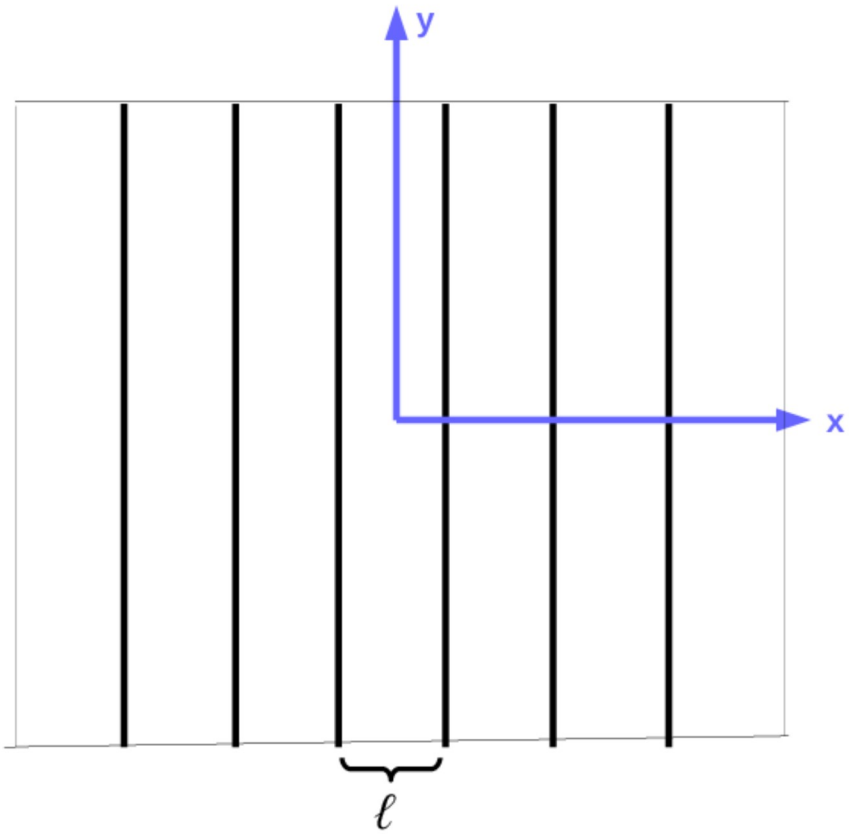
Question 5: Detector Resolution and Monte Carlo methods (40 points)

Learning objectives

In this question you will:

- Understand what position resolution means using a silicon strip detector as an example
- Demonstrate that you can use simulation to solve problems that are more complicated than what can be done analytically
- Learn how noise affects measurements

In this problem we will study how the position resolution of a detector depends upon the properties of that detector. For our example, we will consider a silicon strip detector. We will describe our detector as an x-y plane segmented into strips, each of width ℓ in the y-direction. When a track moving in the z-direction passes through the plane, it deposits energy in the detector and that energy is collected using charge sensitive amplifiers (one per strip). You many assume that the incident track is normal to the silicon plane. Looking down on the strip detector (so that the incident tracks are traveling into the page), the detector looks like this:



The position $x = 0$, $y = 0$ is taken to be the center of the middle strip.

5a.

Suppose all the energy is deposited in a single strip (the strip the track passes through). Find an expression for the position resolution of the detector as a function of ℓ. The position resolution is defined to be $\sigma_x = \sqrt{var[(x_{meas} - x_{true})]}$ where x_{true} is the position where the track actually hit the detector. Because we only know which strip is hit, in this example x_{meas} is the center of the strip that is hit.

Assuming x_{meas} and x_{true} refer to the same strip per independent measurement, then we can think about this as a uniform distribution over a single strip of length ℓ. Using notation from the previous problem, we can use $b = \ell/2$ and $a = -\ell/2$. Thus:

$$\sigma^2 = \frac{(b-a)^2}{12} \rightarrow \sigma = \sqrt{\frac{\ell^2}{12}}$$

Or that

$$\sigma_x = \frac{\ell}{\sqrt{12}} \approx 0.289\ell$$

5b.

Suppose that the charge deposited in our detector spreads out due to physical effects such as diffusion. It is possible for more than one strip to register a signal. Assume in this part that our electronics is binary (i.e. registers a 1 if the deposited energy on the strip is above a specified threshold and 0 otherwise). Assume the threshold on the electronics is such that particles hitting within a distance of $\pm\ell/3$ of the center of the strip only register on a single strip while all particles hitting further from the strip center register on two strips.

What is the position resolution now? (Here, if only one strip is hit, x_{meas} is the center of the strip. If two strips are hit, then x_{meas} is the common edge of the two hit strips). **Note:** this is *not* an unrealistic example. The ATLAS silicon strip detector has such binary readout.

For one strip hit, $b-a=\frac{2}{3}\ell$. For each of the "two-hit" regions, $b-a=\frac{1}{6}\ell$.
Actually, that's what I thought it would be based on the wording above, but that does not yield the correct value as the distribution in 5d. So I've come back to change the thought process, but I still think, based on the wording above and the given code from 5d, my initial model should work (but it doesn't for some reason). Instead, I'm going to say that the 3 regions are of equivalent length/spacing if we consider the whole bar (which is untrue actually. The middle region is 2/3, while each of the sides is 1/6 like I mentioned before). This way, we define separate variances, and can calculate the total variance by addition, and then find the standard deviation afterwards. Thus:

$$\sigma_1^2=(\frac{1}{3}\ell)^2\frac{1}{12}=\frac{1}{108}\ell^2$$

$$\sigma_1^2=\sigma_2^2=\sigma_3^2$$

$$\sigma_x^2=\sigma_1^2+\sigma_2^2+\sigma_3^2=\frac{3}{108}=\frac{1}{36}\ell^2$$
$$\therefore\sigma_x=\frac{\ell}{6}\approx 0.167\ell$$

As shown here, we've decreased σ_x without having to do any hardware improvements

5c.

So far, it has been possible to calculate the position resolution analytically. In cases where the detector response is more complicated, this may not be the case. Typically, physicists model detector performance using Monte Carlo simulations. In the remainder of this problem, you will write a simple simulation to determine the position resolution of a silicon detector.

Let's begin by reproducing the analytic results obtained above. Consider a silicon strip detector that consisting of several strips of width ℓ.
Assume that the incident particles have a uniform distribution in x with $-\ell/2<x<\ell/2$ and all have $y=0$. (We'll just focus on this "center strip", so x_{meas} can either be $-\ell/2$, 0, or $\ell/2$ depending on where the particle hits.) Generate 10,000 such particles for the case described in part **2(a)** and for the case described in part **2(b)**. For each case, make a histogram of ($x_{meas}-x_{true}$) and verify that the resolution is consistent with that obtained in problem **2**.

You can use np.random.uniform() to sample from a uniform distribution with arbitrary bounds. We'd like 10,000 samples for this problem, so we set size = 10000. Note that we take $\ell=1$ for simplicity.

In [98]:
```
import numpy as np
samples = np.random.uniform(-0.5, 0.5, size=10000)
```

For the case of **2(a)**, $x_{meas} = 0$ (the center of the strip), so we can simply find the variance of these samples and take the square root to find the position resolution. Note that this is really just a repetition of the first problem, but with our bounds shifted from (0,1) to (-0.5,0.5).

In [99]:
```
variance = np.var(samples)
onestripres = np.sqrt(variance)
print("The position resolution of the sample, using the 5a model, is",
onestripres)
```

```
The position resolution of the sample, using the 5a model, is 0.28911
046853464173
```

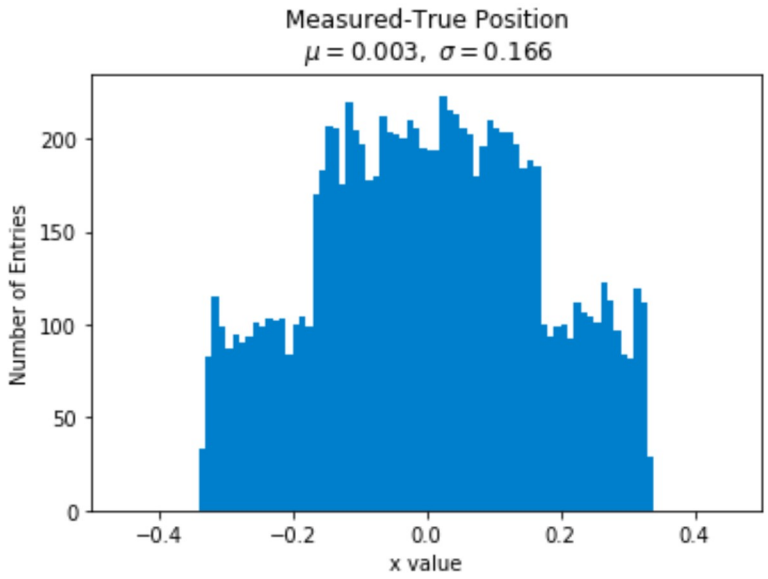
In [100]:
```
print("Compare with the analytical result of 1/sqrt(12) or", 1/np.sqrt
(12))
```

```
Compare with the analytical result of 1/sqrt(12) or 0.288675134594812
9
```

5d.

For the case where two strips are sometimes hit, we need to consider the three different cases for x_{meas}.

In [96]:
```
#We'll call the quantity (x_meas - x_true) the "error" of the measureme
nt
errors = []

#Go through each of our random samples and append (x_meas - x_true) to
the error list
for x_true in samples:
    #particle hits the left third of strip
    if x_true < -1/3:
        x_meas =  -0.5
    #particle hits the right third of strip
    elif x_true > 1/3:
        x_meas = 0.5
    # in this case, else would mean the middle 2/3s of the strip
    else:
        x_meas = 0

    errors.append(x_meas - x_true)

meanx, sigmax = plot_histogram(errors,"x value","Number of Entries","Me
asured-True Position",[-0.5,0.5])
```



In [101]:
```
print("The overall position resolution is", sigmax)
print("Compare with the analytical result 1/6 or", 1/6)
```

```
The overall position resolution is 0.16627822869207717
Compare with the analytical result 1/6 or 0.16666666666666666
```

In [ ]: