Problem Set 3 solutions

Question 1: Relativistic Expressions for Accelerator Performance

Learning objectives

In this question you will:

```
Derive the expressions for the center-of-mass energy at colliders and in fixed target experim
ents
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 4, 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.

Collider case:

$$\begin{aligned} s &= (p_1+p_2)^2 \\ &= (E_1+E_2)^2 - (\vec{p}_1+\vec{p}_2)^2 \\ &= E_1^2 + 2E_1E_2 + E_2^2 - \vec{p}_1^2 - 2\vec{p}_1\cdot\vec{p}_2 - \vec{p}_2^2 \\ &= E_1^2 - \vec{p}_1^2 + E_2^2 - \vec{p}_2^2 + 2E_1E_2 - 2\vec{p}_1\cdot\vec{p}_2 \\ &= m_1^2 + m_2^2 + 2E_1E_2 - 2\vec{p}_1\cdot\vec{p}_2 \\ &= m_1^2 + m_2^2 + 2E_1E_2 + 2\left|\vec{p}_1\right|\ \left|\vec{p}_2\right| \end{aligned}$$

where the final line uses the fact that the two collider beams are colliding head-on. Note, we could write $|\vec{p}| = \sqrt{E^2-m^2}$ to remove the momentum from the equation. Then, in the limit $m_1 \to 0$ and $m_2 \to 0$ we get:

$$s = 2E_1E_2 + 2E_1E_2 = eE_1E_2$$

which is the expression quoted in class

Fixed Target case The target is at rest so its four vector is (m*{target},0,0,0). Therefore: \begin{eqnarray*} s & = & \left (E*{beam}+m*{target} \right)^2 - \left (\vec p*{beam} + 0 \right)^2\ & = & E*{beam}^2 + 2 m*{target} E*{beam} + m*{target}^2 - \vec p*{beam}^2 \ & = & m*{beam}^2 + 2 m*{target} E*{beam} \ \end{eqnarray*} In the limit where $m_{beam} << E_{beam}$, this reduces to

$$s = 2m_{target}E_{beam}$$

which is the expression quoted in class

Question 2: Cherenkov Imaging

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?

The expression for the angle of the Cherenkov radiation is

$$\cos\theta = \frac{1}{n\beta}$$

Since $\cos\theta \le 1$,

$$\beta \ge \frac{1}{n}$$

to give Cherenkov light. Using the relativistic expression for β in natural units:

$$\begin{aligned} \beta &= \frac{p}{E} \\ &= \frac{p}{\sqrt{p^2+m^2}} \\ \frac{p}{\sqrt{p^2+m^2}} &\ge \frac{1}{n} \end{aligned}$$

The smallest momentum is then we have the equal sign above

$$\begin{aligned} \frac{p}{\sqrt{p^2+m^2}} &= \frac{1}{n} \\ p^2 &= \frac{1}{n^2}\left(p^2+m^2\right) \\ p^2\left(1-\frac{1}{n^2}\right) &= \frac{m^2}{n^2} \\ p^2 &= \frac{m^2}{n^2\left(1-\frac{1}{n^2}\right)} \\ p &= \frac{m}{n\sqrt{1-\frac{1}{n^2}}} \end{aligned}$$

The mass of the $K^\pm$ is 0.493 GeV so the minimum p for Cherenkov light is

$$p = 0.493/\left(1.2\sqrt{1-1.2^2}\right) = 0.74\ \text{GeV}$$

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 $K^\pm$?

For the kaon:

$$\begin{aligned} \cos\theta &= \frac{1}{n\beta} \\ &= \frac{E}{1.2p} \\ &= \frac{\sqrt{p^2+m_K^2}}{1.2\times p} \\ \cos\theta &= 0.849 \\ \theta &= 0.556 \end{aligned}$$

For the pion with mass = 0.1396 GeV. we find:

$$\begin{aligned} \cos\theta &= 0.8346 \\ \theta &= 0.583 \end{aligned}$$

So the difference in angle is $0.583 - 0.556 = 0.0273$ radians, or 27.3 mradians

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

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

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:

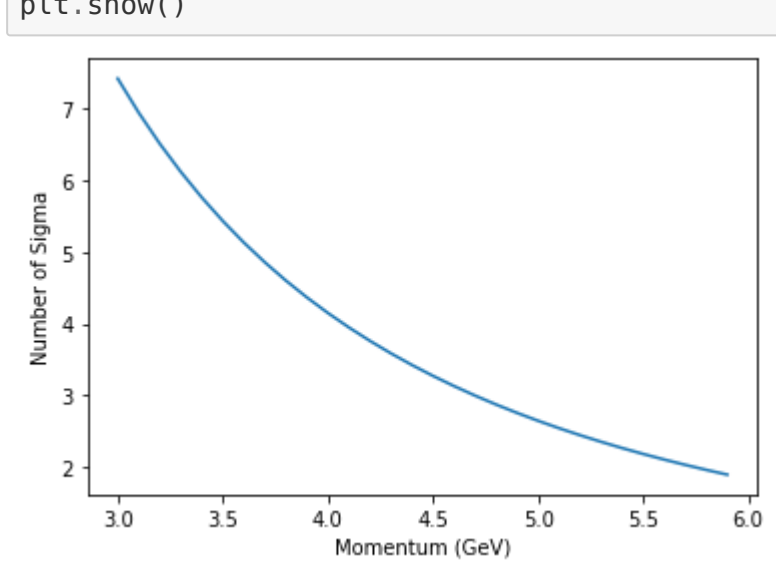
Drawing

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?

Since the resolution is 2.5 mrad and we want a 90\%\ confidence limit separation, we want $2.56 \times 2.5 = 6.4$ mrad separation. The easiest way to find this momentum is graphically.

In [4]:
```
import matplotlib.pyplot as plt
import math
import numpy as np
def numSig(p):
    betaPi = p/math.sqrt(p*p+0.13957*0.13957)
    betaK = p/math.sqrt(p*p+0.493677*0.493677)
    cosThetaPi = 1.0/(1.2*betaPi)
    cosThetaK = 1.0/(1.2*betaK)
    ns = (math.acos(cosThetaPi)-math.acos(cosThetaK))/0.00256
    return ns

pin = np.arange(3.0, 6.0, 0.1)
ns = []
for x in pin:
    ns.append(numSig(x))

plt.plot(pin,ns)
plt.xlabel("Momentum (GeV)")
plt.ylabel("Number of Sigma")
plt.show()
```



So, the π and K can be separated at 90\% confidence level up to about 5 GeV

Question 3: Luminosity of a Collider

(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^2. Calculate the instantaneous luminosity on the assumption that the beams collided head-on.

The instantaneous luminosity at a collider is

$$l = f\frac{n_1n_2}{4\pi\sigma_x\sigma_y}$$

There is some ambiguity as to how these σ's relate to the area of the beam. We'll assume here that σ_x and σ_y correspond to the semi-major and semi-minor axes of the beam so that the area of the beam is

$$A_{beam} = \pi\sigma_x\sigma_y$$

However, we will give full credit for any reasonable assumption for how the area is related to the σ's.

If the number of electrons/positrons in each bunch is n_e and each bunch circulates the ring at a frequency of $f_b = c/(27\times10^3) = 11.1$ kHZ, the bunch current I is given by

$$I = f_b \times n_e e$$

A current of 1 mA therefore corresponds to $n_e = 5.6\times10^{11}$ 10^3 \; {\rm kper bunch,

$$n_1 = n_2 = 5.6\times10^{11}$$

With four bunches per beam, the bunch-crossing frequency of

$$f = 4\cdot c/27000 = 4\ \text{fb} = 44.4\ \text{kHz}$$

The instantaneous luminosity is

$$\mathcal{L} = 44.4\times10^3\ \text{kHz}\frac{\left(5.6\times10^{11}\right)^2}{4\times1.8\times10^4\ \ \mu\text{m}^2} = 6\times10^{32} = 2.4\times10^{31}\ \text{cm}^{-2}\text{s}^{-1}$$

Looking at https://pdg.lbl.gov/2020/reviews/rpp2020-rev-hep-collider-params.pdf Table 32.2, you will see that this result is in good agreement with the actual luminosity of LEP when running at the energy of the Z boson

Question 4: Central Limit Theorem

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* 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), a random variable x uniformly distributed:

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

4a.

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

Solution:

For the mean:

$$\begin{aligned} \mu &\equiv \int_\infty^\infty xf(x)dx \\ &= \int_0^1 \frac{x}{b-a}dx \\ &= \frac{1}{2}\frac{x^2}{b-a}\Big|_a^b \\ &= \frac{1}{2}\left(\frac{b^2-a^2}{b-a}\right) \\ &= \frac{a+b}{2} \end{aligned}$$

For the variance:

$$\begin{aligned} \sigma^2 &\equiv \int_\infty^\infty x^2f(x)dx - \mu^2 \\ &= \int_0^1 x^2dx - \frac{1}{4}\left(\frac{b^2-a^2}{b-a}\right)^2 \\ &= \frac{1}{3}x^3\Big|_a^b - \frac{1}{4}(a+b)^2 \\ &= \frac{1}{3}\left(\frac{b^3-a^3}{b-a}\right) - \frac{1}{4}(a+b)^2 \\ &= \frac{a^2}{3} + \frac{ab}{3} + \frac{b^2}{3} - \frac{a^2}{4} + \frac{ab}{2} + \frac{b^2}{4} \\ &= \frac{1}{12}\left(a^2 - 2ab + b^2\right) \\ &= \frac{1}{12}(a-b)^2 \end{aligned}$$

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 [99]:
```
#Import the NumPy library as "np"
import numpy as np

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

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

```
[0.50530506 0.18439254 0.51488023 0.81991371 0.89529383 0.78487876
 0.71686778 0.98827103 0.22221948 0.95705558]
```

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

In [ ]:
```
def find_mean(num_list):

    """Your code here to calculate mean"""

    return mean

def find_variance(num_list):

    """Your code here to calculate variance"""

    return variance
```

Solution:

In [100]:
```
def find_mean(num_list):
    return sum(num_list)/len(num_list)

def find_variance(num_list):
    average = find_mean(num_list)
    variance = sum((average - value) ** 2 for value in num_list) / len(num_list)
    return variance
```

Feel free to compare against NumPy's built-in functions:

In [101]:
```
#NumPy's functions
print("np values of mean: ",np.mean(samples)," and variance: ",np.var(samples))

#Your functions
print("My values of mean: ",find_mean(samples)," and variance: ",find_variance(samples))
```

```
np values of mean:  0.49860792871841114  and variance:  0.08437073153209712
My values of mean:  0.4986079287184117  and variance:  0.08437073153209715
```

4c.

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

Solution:

The predicted mean is 0.5 and the predicted variance is \frac{1}{12}=0.833, in good agreement with our numeric results

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(), with the list as the input parameter. There are many optional parameters, like the number of bins and the color of the bars.

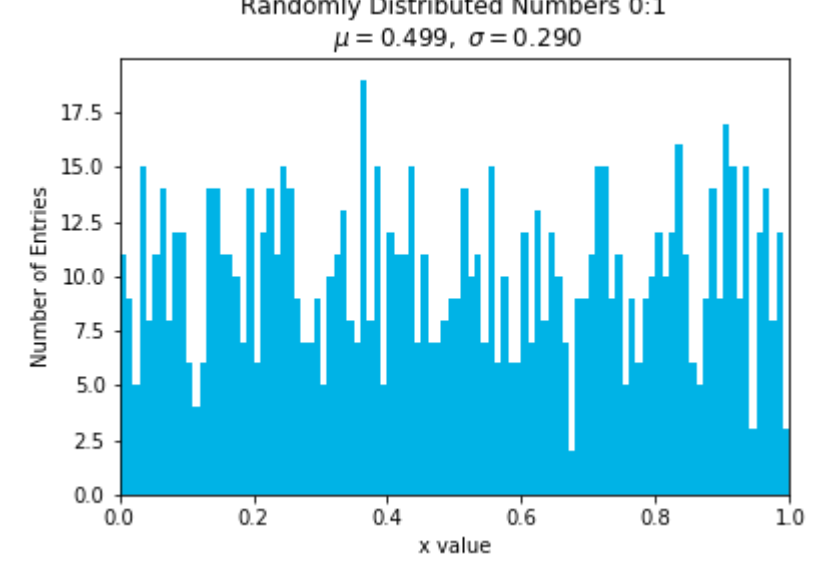
Since our histograms will share similar formatting, it's also useful to define a function rather than typing the same thing over and over. For this homework, we'll supply the histogramming routine. In the future, you will write this code yourself.

In [102]:
```
#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 title, 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 color
    plt.hist(samples, bins=100, range=limits, color=(0,0.7,0.9))  #Set the color using (r,g,b) valu
es or
                                                                  #  use a built-in matplotlib colo
r"""

    #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,sigma) )

    #Get rid of the extra white space on the left/right edges (you can delete these two lines witho
ut 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()
```

Take some time to play with the plot formatting and choose a color you like for the bars. Then call the function with your random samples.

Solution:

In [103]:
```
plot_histogram(samples,"x value","Number of Entries","Randomly Distributed Numbers 0:1",[0.0,1.0])
```



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.

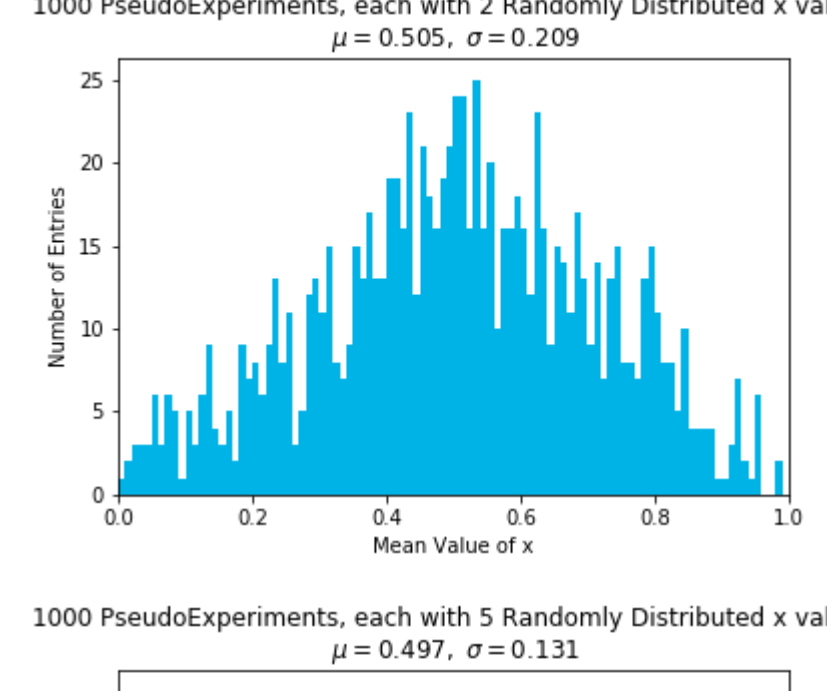
In [42]:
```
#Calculates and returns S as defined above
def pseudoexperiment(N):
    samples = np.random.rand(N)  #samples = [x_1, x_2, ... x_N]
    s = 0  #replace this with your calculation of s

    """Your code here"""

    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
    plot_histogram(s_list,"Mean Value of x","Number of Entries","1000 PseudoExperiments, each with
 "+str(N)+" Randomly Distributed x values",[0.0,1.0])
```

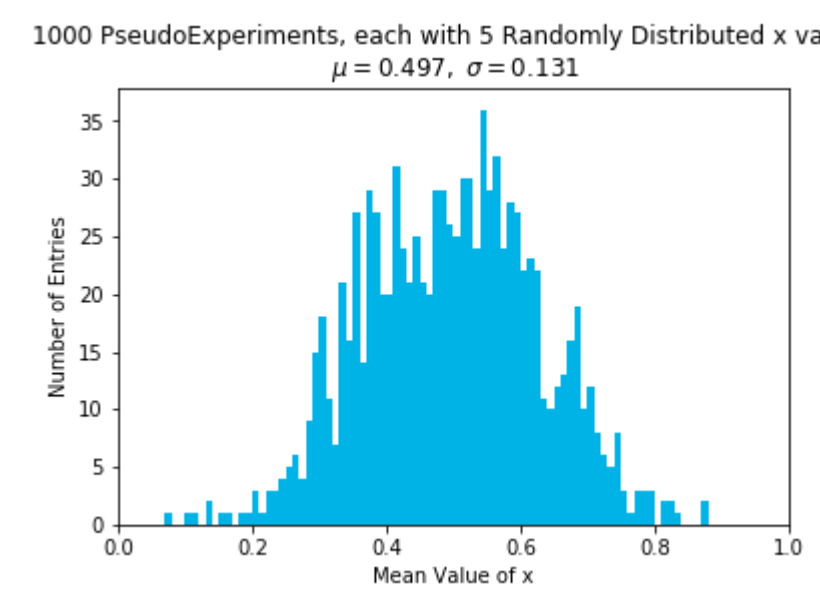
In [43]:
```
# Uncomment this and run it after you have completed the code above
#for N in [2,5,10]:
#    run_pseudoexperiments(N)
```

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

Solution:

In [104]:
```
#Calculates and returns S as defined above
def pseudoexperiment(N):
    samples = np.random.rand(N)  #samples = [x_1, x_2, ... x_N]
    s = sum(samples)/len(samples)
    return s

for N in [2,5,10]:
    run_pseudoexperiments(N)
```





Question 5: Detector Resolution and Monte Carlo methods

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:

Drawing

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.

Solution:

This is essentially the same as the Central Limit Theorem problem. The expression for the variance is

$$\sigma^2 = \int (x - x_{meas})^2 f(x)dx$$

where $f(x)$ is the probability distribution function for x. If we let $x_{true} = x$ range from $-\ell/2$ to $\ell/2$ then x_{meas} always is 0 and $f(x) = 1/\ell$. Thus

$$\sigma^2 = \int_{-\ell/2}^{\ell/2} x^2(1/\ell)dx = \frac{\ell^2}{12}$$

Thus $\sigma = \ell/\sqrt{12}$.

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.

Solution:

The only difference from part (a) is that $x_{meas} = 0$ for the inner two thirds of the strip and $x_{meas} = \ell/2$ for the left sixth and $-\ell/2$ for the right sixth. So

$$\sigma^2 = \int_{-\ell/2}^{-\ell/3}(x-(-\ell/2))^2(1/\ell)dx + \int_{-\ell/3}^{\ell/3}x^2(1/\ell)dx + \int_{\ell/3}^{\ell/2}(x-\ell/2)^2(1/\ell)dx = \frac{\ell^2}{36}$$

Thus $\sigma = \ell/6$

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.

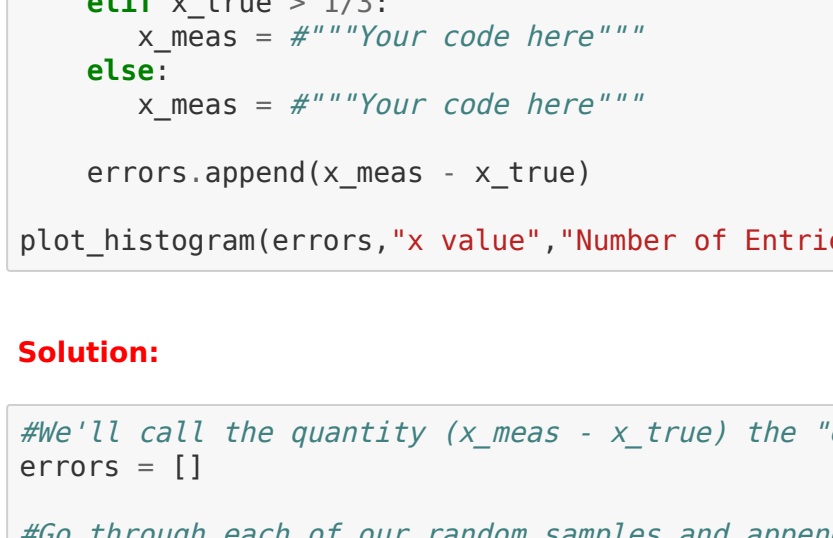
In [2]:
```
import numpy as np
samples = np.random.uniform(-0.5, 0.5, size=10000)
```

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).

Solution:

In [8]:
```
#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 title, 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 color
    plt.hist(samples, bins=100, range=limits, color=(0,0.7,0.9))  #Set the color using (r,g,b) valu
es or
                                                                  #  use a built-in matplotlib colo
r"""

    #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,sigma) )

    #Get rid of the extra white space on the left/right edges (you can delete these two lines witho
ut 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()
```

In [4]:
```
plot_histogram(samples,"x value","Number of Entries","True minus Measured Position",[-0.5,0.5])
```



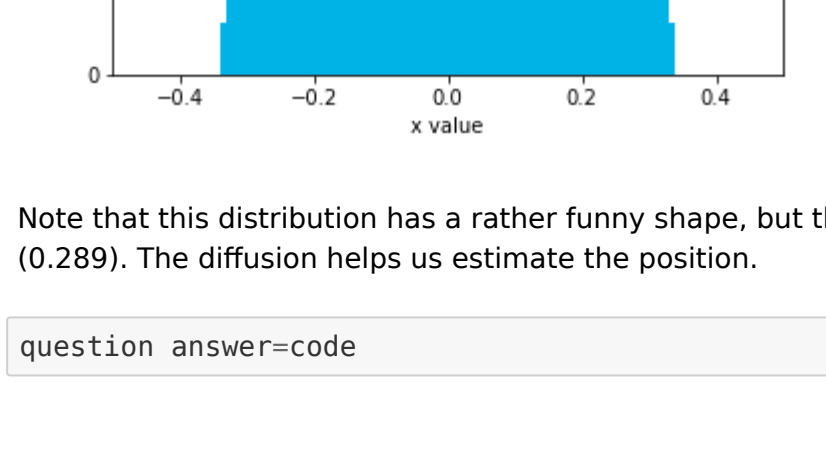
5d.

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

In [11]:
```
#We'll call the quantity (x_meas - x_true) the "error" of the measurement
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 =  # """Your code here"""
    #particle hits the right third of strip
    elif x_true > 1/3:
       x_meas = #"""Your code here"""
    else:
       x_meas = #"""Your code here"""

    errors.append(x_meas - x_true)

plot_histogram(errors,"x value","Number of Entries","Measured-True Position",[-0.5,0.5])
```

Solution:

In [107]:
```
#We'll call the quantity (x_meas - x_true) the "error" of the measurement
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
    else:
        x_meas = 0.0

    errors.append(x_meas - x_true)

plot_histogram(errors,"x value","Number of Entries","Randomly Distributed Numbers 0:1",[-0.5,0.5])
```



Note that this distribution has a rather funny shape, but the variance of 0.166 is much less than in the previous case (0.289). The diffusion helps us estimate the position.

In [ ]:
```
question answer=code
```

Now, let us replace our binary electronics from part 2(b) with analog electronics (so that the magnitude of the charge deposited on the strip is recorded). We will model the transverse spreading of the charge from our incident track using a Gaussian distribution with width σ_M:

$$f(x)\,dx = \frac{1}{\sigma_M\sqrt{2\pi}} \exp(-(x-x_0)^2/2\sigma_M^2)\,dx$$

where $f(x)$ is the charge deposited between position x and $x+dx$ and x_0 is the point where the track hits the detector. Assume that the total energy deposited by each track is 1 MIP (a MIP is the energy deposited by a single minimum ionizing particle), that our analog electronics has a threshold of 0.2 MIP and that $\sigma_M = \ell$. Also assume that the electronics has an intrinsic noise contribution $\sigma_N = 0.1$ MIP. (This means that the measurement of the charge on each strip is modified by adding a noise contribution that is distributed according to a Gaussian with mean 0 and variance σ_N^2. Assume that the noise on neighboring strips is uncorrelated.)

Generate 10,000 particles and simulate the response of this silcon strip detector (**using 7 strips now**) to these particles. From this simulation determine the position resolution of the silicon detector. Assume that in the analysis of these data the measured position of the particle is:

$$x_{meas} = \sum_{i=strips} q_i x_i$$

where the index i is the strip number, q_i is the measured charge on the strip (set to zero for strips with charge below threshold) and x_i is the position of the center of strip i.

Once again, assume that the incident particles have a uniform distribution in x with $-\ell/2 < x < \ell/2$.

We'll take $\sigma_M = \ell = 1$ for simplicity. To keep track of our detector geometry, we'll first create a list of each strip's centers along with a list of its left/right bounds.

In [108]:
```
num_strips = 7  #Keep this odd so the problem is symmetric
centers = []
bounds = []

for i in range(-int(num_strips/2),int(num_strips/2)+1):
    centers.append(i)
    bounds.append([i-0.5, i+0.5])

print(centers, bounds)
```

```
[-3, -2, -1, 0, 1, 2, 3] [[-3.5, -2.5], [-2.5, -1.5], [-1.5, -0.5], [-0.5, 0.5], [0.5, 1.5], [1.5, 2.5], [2.5, 3.5]]
```

Next, we'll use the error function to integrate over the Gaussian distribution while finding the charge deposited on each strip. We'll also implement the weighted sum in finding the measured position.

In [ ]:
```
#Import the error function to help integrate the gaussian distribution
from math import erf

#Finds the charge deposited on strip i with a hit at location x
def get_charge(i,x):
    charge=0 # replace this with your code below
    """Your code here.  Using the error function erf() is convenient/fast, but
       feel free to use an integration package like scipy.integrate or write your
       own numerical integration function.  The left and right bound of each strip
       is contained in bounds[i][0] and bounds[i][1], respectively. """

    return charge

#Finds the measured particle position
def find_x_meas(charges, cutoff):
    xmeas = 0 # replace this with your code below
    """Your code here.  Use the weighted sum definition given in the problem statement.
       Note that x_i = centers[i]."""

    return xmeas
```

Finally, we'll write a function that finds the position resolution for any intrinsic noise and charge threshold.

In [109]:
```
#Noise, cutoff corresponds to sigma_N, threshold described in the problem statement
def test_analog_electronics(noise, cutoff):
    samples = np.random.uniform(-0.5,0.5, size=10000)
    errors = []
    for x_true in samples:
        #Calculate the charge deposited on each detector strip
        charges = []
        for i in range(num_strips):
            charges.append(get_charge(i,x_true))

        #Add intrinsic noise to the electronics
        charges += np.random.normal(0, noise, num_strips)

        #Find the measured position of the particle
        x_meas = find_x_meas(charges, cutoff)

        errors.append(x_meas - x_true)

    return errors
```

First run the test_analog_electronics(σ_N, cutoff) function with $\sigma_N = 0.05$ and cutoff $= 0.2$ MIP. How does the position resolution of the detector change as you increase/decrease these two parameters? (Note: physicists often characterize the the threshold in terms of how units of σ_N. For example, the parameters above correspond to a threshold of $4\simga_N$. Consider when exploring these parameters, describing the theshold in this way)

flags: solutions

In [141]:
```
# Import the error function to help integrate the gaussian distribution
# erfc(x) is 1-erf(x)
from math import erf, erfc
import math

#Finds the charge deposited on strip i with a hit at location x
def get_charge(i,x):
    # We need to find the strip boundaries relative to the x_true
    lowEdge = bounds[i][0]-x
    highEdge = bounds[i][1]-x
    # Find the integrated charge to the right of the lower edge of strip i
    # lowEdge negative means x  is to the right of the lower boundary of the strip
    if lowEdge < 0:
        chargeBelow = 0.5*erfc(abs(lowEdge))
    else:
        chargeBelow = 0.5+0.5*erf(lowEdge)
    # Find the integrated charge to the left of the upper edge of strip i
    # highEdge positive means x is to the right of the upper boundary of the strip
    if highEdge > 0:
        chargeAbove = 0.5*erfc(highEdge)
    else:
        chargeAbove = 0.5+0.5*erf(abs(highEdge))
    charge = 1.0-chargeBelow-chargeAbove
    return charge

#Finds the measured particle position
def find_x_meas(charges, cutoff):
    xsum = 0.0
    chargeSeen = 0.0
    for i in range(7):
        if charges[i]>cutoff:
            xsum += charges[i]*centers[i]
            chargeSeen += charges[i]
    if chargeSeen > 0:
        xmeas = xsum/chargeSeen
    else:
        xmeas = 0.0

    return xmeas

#Noise, cutoff corresponds to sigma_N, threshold described in the problem statement
def test_analog_electronics(noise, cutoff):
    samples = np.random.uniform(-0.5,0.5, size=10000)
    errors = []
    for x_true in samples:
        #Calculate the charge deposited on each detector strip
        charges = []
        for i in range(num_strips):
            charges.append(get_charge(i,x_true))

        #Add intrinsic noise to the electronics
        charges += np.random.normal(0, noise, num_strips)

        #Find the measured position of the particle
        x_meas = find_x_meas(charges, cutoff)
        errors.append(x_meas - x_true)

    return errors
sigmaNoise = 0.05
threshold = 0.20
errors = test_analog_electronics(sigmaNoise,threshold)
plot_histogram(errors,"Measured x - True x","Number of Entries","sigmaNoise"+format(sigmaNoise,"5.
4")+" threshold: "+format(threshold,"5.4"),[-0.5,0.5])

scanNoise = []
nSigMultipliers = np.arange(5,1,-0.2)
#for i in np.arange(4,1,-0.1):
#    nSigMultipliers.append(i)
for sigmaNoise in [0.10, 0.05, 0.025]:
    scanThresh = []
    for nSig in nSigMultipliers:
        threshold = nSig*sigmaNoise
        errors = test_analog_electronics(sigmaNoise,threshold)
        scanThresh.append(math.sqrt(np.var(errors)))
    scanNoise.append(scanThresh)

plt.plot(nSigMultipliers,scanNoise[0],'ro')
plt.plot(nSigMultipliers,scanNoise[1],'bo')
plt.plot(nSigMultipliers,scanNoise[2],'go')
plt.xlabel("Threshold (units: # sigmaNoise)")
plt.ylabel("Resolution")
plt.title("SigmaNoise = 0.10 (red), 0.05 (blue), 0.025 (green)")
plt.ylim((0.0,1.2*max(scanNoise[0])))
plt.show()
#        plot_histogram(errors,"Measured x - True x","Number of Entries","sigmaNoise: "+format(sigm
aNoise,"5.4")+" threshold: "+format(threshold,"5.4"),[-0.5,0.5])
```


From the plot above, we see that as expected, the resolution degrades with increasing σ_N. For a given value of σ_N the best resolution occurs for thresholds between 2.5 and 3 times σ_N. For lower thresholds, we include signals that come largely from noise. For higher thresholds, we start to loose real signal. However, if the noise is low enough, the change of reolution with threshold becomes less important.