Problem Set 10 problems

Question 1: Two Jet Production in pp collisions (30 points)

Learning objectives

In this question you will:

- Review the Feynman diagrams that represent possible vertices for the strong interactions
- Apply this knowledge to the example of proton proton collisions

On problem set 9, you looked at the process $pp \to \mu^+\mu^- X$. This process occurs via the electomagnetic interaction.bThe cross section (calculated to lowest order in perturbationtheory) is proportional to α^2 (a factor of e^2 in the matrix element leads to a factor of e^4 in the cross section).

Consider instead the process $pp \to jjX$ where j is a jet coming from the hadronization of a parton (a quark, an antiquark or a gluon). This process occurs through the strong interaction and the lowest order term for the cross section is therefore proportional to α_S^2. Draw all the Feynman diagrams that contribute to this lowest orderbcalculation.

Hints:

- The three vertices that describe the couplings of the strong interaction are shown on page 6 of the Lecture 19 notes.
- In many cases, both for s-channel and t-channel diagrams are possible
- There are quite a large number of such diagrams. Start by listing all the possible initial state combinations of partons and then for each initial state then draw all the relevant scattering diagrams for that initial state.

put your answer here

Question 2: Decays of the J/ψ (30 points)

Learning objectives

In this question you will:

- Review the reason why the J/ψ hadronic decays occur via 3 gluon annihilation and why these decays have a similar description to the decay of positronium
- Apply these concepts to estimate α_S
- Use Feynman diagrams to estimate the relative branching ratio for two decays

(adapted from Perkins)

Charmonium is the term used to describe any bound state of a charm quark and charm anti-quark. We can understand much about charmonium using insight gained from our understanding of positronium. The degeneracy of the $n = 1$ state of positronium is split due to spin-spin interactions to Spin-0 para-positronium and Spin-1 otho-positronium. The decay rates for these two states are quite different since para-positronium can decay to two photons while ortho-positronium cannot and therefore decays to three photons (due to charge conjugation invariance). The decay widths are:

$$\begin{aligned}\Gamma(2\gamma) &= \tfrac{\alpha^5 m}{2} \\ \Gamma(3\gamma) &= \tfrac{2\left(\pi^2-9\right)}{9\pi}\alpha^6 m\end{aligned}$$

where m is electron mass.
The same arguments that force ortho-positronium to decay to 3 photons require hadronic decays of the J/ψ to occur through 3 gluons.

2a.

The J/ψ gas a full width $\Gamma = 87$ keV and 88% of the decays are to a hadronic final state, Assuming that these 3 gluon decays follow the same formula as ortho-positronium with $\frac{4}{3}\alpha_S$ replacing α (the $\frac{4}{3}$ is a color factor) estimate the value of α_S from these data

write your answer here

2b.

Estimate the ratio of the branching ratios of $J/\psi \rightarrow \gamma + hadrons$ to $J/\psi \rightarrow hadrons$. How well does your estimate agree with measurements (as tabulated in the PDG)?

write your answer here

Question 3: Sphericity and the Discovery of the Gluon (40 points)

Learning objectives

In this question you will:

- learn how to calculate the sphericity tensor and understand how to interpret its eigenvalues
- reproduce (using simulated data) the analysis performed by Tasso to discover the gluon
- develop a simple event display as an aid to visualizing what two jet and three jet events look like in e^+e^- annihilation events

The first experimental evidence for the existence of the gluon came from the analysis of data collected at the Petra accelerator at DESY. Several different analysis strategies were used by the four collaborations (see arxiv:1012.2288 (https://arxiv.org/pdf/1012.2288.pdf) for a review). One such analysis, performed by the Tasso group, was based on studies of the sphericity tensor:

$$S_{\alpha\beta} = \frac{\sum_i p_{i\alpha} p_{i\beta}}{\sum_i \vec{p}_i^{\,2}}$$

where the sum is over all charged particles in the event (Tasso did not have good enough calorimetry to include neutrals in the analysis) and the α and β indices run from 1 to 3, representing the x, y and z components of the momentum vector. For each event, the Sphericity tensor can be diagonalized to obtain the principle axes $\hat{n}_1$ through $\hat{n}_3$ and eigenvalues Q_1 through Q_3. With the definition of S above, $Q_1 + Q_2 + Q_3 = 1$, so we only need two eigenvalues to specify each event. If the eigenvalues are ordered so that $Q_1 < Q_2 < Q_3$ then the sphericity is defined to be

$$S \equiv \frac{3}{2}(1 - Q_3) = \frac{3}{2}(Q_1 + Q_2)$$

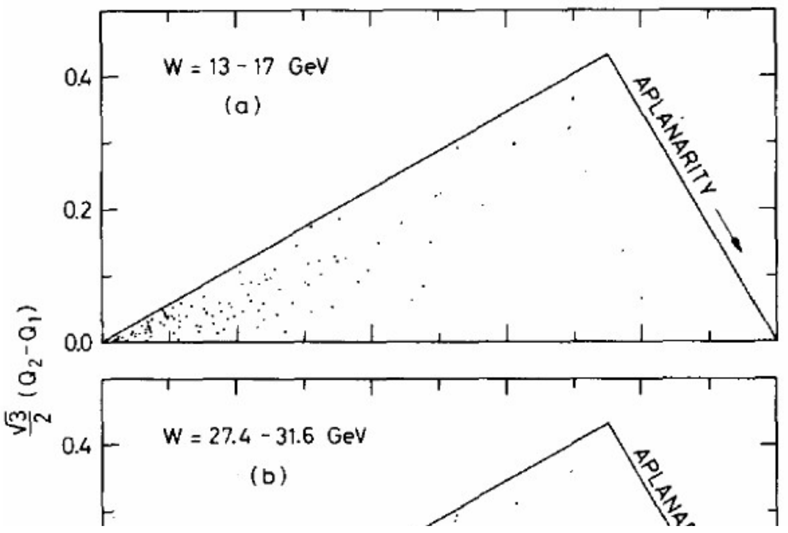
and the aplanarity is defined to be

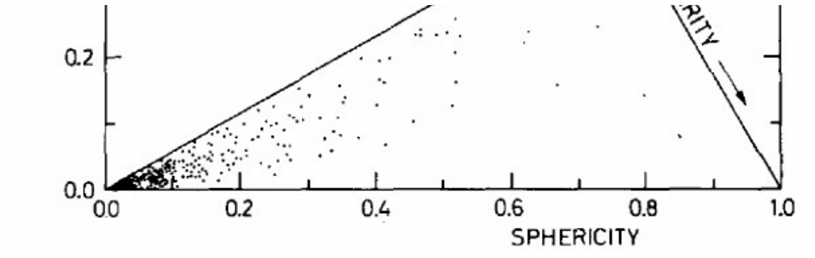
$$A \equiv \frac{3}{2} Q_1$$

Here $0 < S < 1$ and $0 < A < 0.5$.

** A note on nomenclature:The Q_1 through Q_3 described here are the same as the λ_1 throught λ_3 discussed in Lecture 19 except that they are ordered so that Q_1 is the smallest while the class definition has λ_3 as the smallest. The convention used in this problem matches the article cited here while the convention used in class is consistent with most modern textbooks)

The form of the sphericity tensor is the same as that of the moment of interia tensor (where the object's position is replaced by its momentum). We can therefore use our intuition from classical mechanics to interpret this tensor. Long, thin rods have one large eigenvalue and two small ones of rougly equal size, while a sphere has three eigenvalues of equal size. If we interpret the the process $e^+e^- \rightarrow hadrons$ at parton level as $e^+e^- \rightarrow q\bar{q}$ we would expect most of the momentum to flow along the original $q\bar{q}$ axis with the momentum transverse to this direction limited by to a scale set by Λ_{QCD}. Thus, such events have S close to zero. The axis n_3 is a good estimate of the initial direction of the back-to-back q and $\bar{q}$. If instead, a single hard gluon is radiated so that the hard scattering process is $e^+e^- \rightarrow q\bar{q}g$. The events would appear planar with only a small component of momentum outside the plane. Here is what Tasso observed:





Distribution of the sphericity and aplanarity of $e^+e^- \to hadron$ events measured by the TASSO collaboration in R. Brandelik et al., *Evidence for Planar Events in e$^+$e$^-$ Annihilation at High Energies*, Phys. Lett. B 86, 243 (1979).

You will now reproduce their analysis using simulated data created with the Pythia8 Monte Carlo generator. The file `Pythia8e+e-Toqqbar36GeV.dat` contains the charged particle information for 10000 events. The format of the file is specified in the metadata comments at the beginning of the file.

3a.

For each event in the file, calculate the sphericity tensor and find its eigenvalues. Plot the data using the same x and y axes as the Tasso plot above.

In [1]:
```
import numpy as np
from matplotlib import pyplot as plt

# Read in the data
with open("Pythia8e+e-Toqqbar36GeV.dat","r") as file:
    lines = file.readlines()

# This is the header
event_header = lines[5]




#### Had to redefine functions in the next boxes; the definition of the
sphericity tensor relies on being able
#### to use all momenta per event, not just one line.

# def momenta_from_events(line):
#     """Take provided event info and extract momemtum vectors

#     Parameters
#     ==========
#     lines : numpy array
#       2 dimensional array of event info

#     Returns
#     =======
#     mom : numpy array
#       n by 3 array of momentum vectors, one for each event given
#     """
#     mom = np.zeros(3)
#     string = np.array(line.split(",")[1:4])

#     if len(string) == 3:
#         mom[0] = float(string[0])
#         mom[1] = float(string[1])
#         mom[2] = float(string[2])

#     #print(mom)

#     return mom

# # def sphericity_tensor_from_mom(mom):
# #     """Calculate the sphericity of a momentum vector

# #     Parameters
# #     ==========
# #     mom : numpy array
# #       3 element array representing a particle momentum

# #     Returns
# #     =======
# #     sphericity : numpy array
# #       3 by 3 array of sphericity tensor values
# #     """
# # def find_eigenvalues(sphericity_tensor):
# #     """Calculate the eigenvalues of the sphericity tensor

# #     Parameters
# #     ==========
# #     sphericity_tensor : numpy array
# #       3 by 3 array representing a sphericity tensor

# #     Returns
# #     =======
# #     eigenvalues : numpy array
# #       sphericity tensor eigenvalues
# #     """

# #     '''Your code here'''

# # def compute_sphericity(eigenvalues):
# #     """Calculate the sphericity of the event given the sphericity t
ensor eigenvalues

# #     Parameters
# #     ==========
# #     eigenvalues : numpy array
# #       3 element array representing sphericity tensor eigenvalues

# #     Returns
# #     =======
# #     sphericity : float
# #       sphericity of the event
# #     """

# #     '''Your code here'''

# momenta = []

# # Start at the 6th element of the lines array, previous ones are head
er info
# i = 6

# '''Plotting code here'''
# print(len(lines))
# for line in lines[6:1000]:

#     #print(len(line))

#     momenta += [momenta_from_events(line)]

# ##print(momenta)
```

In [2]:
```
# To calculate the Sphericity tensor, we actually can't just use one mo
mentum. We need to sum over the momenta per
# event, which is a bit more complicated. Thus, I've written the code b
elow because the template code above
# doesn't really do that.

linesarr = np.array(lines)

eventstarts = np.argwhere(linesarr == '<event>\n')
eventends = np.argwhere(linesarr == '</event>\n')

mom_events = []
energies = []
masses = []

for i in range(len(linesarr)):

    temp_event = []

    if i in eventstarts:
        j = i + 1
        #print(linesarr[j])
        while j not in eventends:
            momentum = np.zeros(3)

            still_string = np.array(linesarr[j].split(',')[1:])

            momentum[0] = float(still_string[0])
            momentum[1] = float(still_string[1])
            momentum[2] = float(still_string[2])

            energies.append(float(still_string[3]))
            masses.append(float(still_string[4]))

            temp_event.append(momentum)
            j += 1

        mom_events.append(temp_event)
```

In [3]:
```
mom_events[0][0]
```

Out[3]:
```
array([ 0.230876, -1.06223 ,  0.265625])
```

In [4]:
```
# now that the momenta are separated via events, then we can perform th
e sphericity tensor calculations
def sphericity_tensor_from_events(mom_events):
    """Calculate the sphericity of a momentum vector

    Parameters
    ==========
    mom_events: momenta separated per event, in a m x n x 3 shape.
    - (ALL events)

    Returns
    =======

    Array of 3 by 3 array of sphericity tensor values, corresponding to
each event
    """

    tensor_list = []

    for event in mom_events:

        magnitude_sum = 0
        xx = 0
        xy = 0
        xz = 0
        yy = 0
        yz = 0
        zz = 0

        # px py = py px, etc.; so that's why its just those quantities


        for momentum in event:

            magnitude_sum += (momentum[0]**2 + momentum[1]**2 + momentu
m[2]**2)
            xx += momentum[0]**2
            xy += momentum[0] * momentum[1]
            xz += momentum[0] * momentum[2]
            yy += momentum[1]**2
            yz += momentum[1] * momentum[2]
            zz += momentum[2]**2

        tensor = np.zeros((3,3))
        # [row][column]
        tensor[0][0] = xx
        tensor[0][1] = xy
        tensor[0][2] = xz
        tensor[1][0] = xy
        tensor[1][1] = yy
        tensor[1][2] = yz
        tensor[2][0] = xz
        tensor[2][1] = yz
        tensor[2][2] = zz
```

```
            tensor /= magnitude_sum

            tensor_list.append(tensor)

            #print(magnitude_sum)

        return tensor_list
```

In [5]:
```
tensors = sphericity_tensor_from_events(mom_events)
```

In [6]:
```
len(tensors)
```

Out[6]:
```
10000
```

In [7]:
```
tensors[0]
```

Out[7]:
```
array([[ 0.04060652,  0.10334013, -0.05565653],
       [ 0.10334013,  0.71201415, -0.3565216 ],
       [-0.05565653, -0.3565216 ,  0.24737933]])
```

In [8]:
```
eigs = np.linalg.eig(tensors[0])[0]
print(eigs)
matrix = np.linalg.eig(tensors[0])[1]
print(matrix)
```

```
[0.92088106 0.02495286 0.05416608]
[[ 0.13216862  0.99117373  0.01030007]
 [ 0.87149748 -0.1211485   0.47520016]
 [-0.47225375  0.05383007  0.87981743]]
```

In [9]:
```
# Eigenvalues add up to 1, check.
np.sum(np.linalg.eig(tensors[0])[0])
```

Out[9]:
```
1.0000000000000002
```

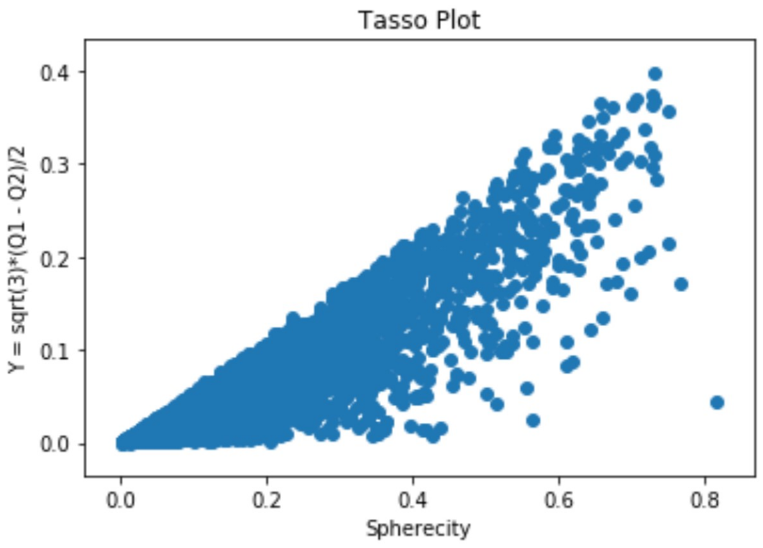
In [10]:
```
# sorts in ascending order, so use first two indices to find Sphericity
np.sort(eigs)
```

Out[10]:
```
array([0.02495286, 0.05416608, 0.92088106])
```

In [11]:
```
# computing sphericity for all tensors then:

def tasso_plot_from_tensors(t_list):

    S = []
    A = []
    y = []
    eigmatrices = []

    for tensor in t_list:
        eigvals = np.linalg.eig(tensor)[0]
        eigmatrix = np.linalg.eig(tensor)[1]
        sortedvals = np.sort(eigvals)
        q1 = sortedvals[0]
        q2 = sortedvals[1]
        current_S = (3/2)*(q1 + q2)
        current_A = (3/2)*q1
        current_y = (np.sqrt(3)/2)*(q2 - q1)

        S.append(current_S)
        A.append(current_A)
        y.append(current_y)
        eigmatrices.append(eigmatrix)



    return S, A, y, eigmatrices
```

In [12]:
```
S, A, y, eigmatrices = tasso_plot_from_tensors(tensors)
```

In [13]:
```
plt.figure()
plt.title("Tasso Plot")
plt.xlabel("Spherecity")
plt.ylabel("Y = sqrt(3)*(Q1 - Q2)/2")
plt.scatter(S, y)
plt.show()
```



3b.

As discussed above, the momentum along the $\hat{n}_3$ axis should be larger than in the other directions. The momentum along the $\hat{n}_2$ direction should be small for 2-jet events, with a tail extending to larger values of momentum for 3-jet events where a gluon is radiated. The momentum along the $\hat{n}_1$ direction should be small unless more than one gluon is radiated. Since $\alpha_S \approx 0.12$ at Petra energies, the probability of multiple hard gluon radiation is small. Make histograms of the components of momentum along each of the three principle axes for the events you have analyzed. What do these plots show?

In [14]:
```
# I went back to the previous problem to grab the matrices to transform
the momenta to the new basis
eigmatrices[0]
```

Out[14]:
```
array([[ 0.13216862,  0.99117373,  0.01030007],
       [ 0.87149748, -0.1211485 ,  0.47520016],
       [-0.47225375,  0.05383007,  0.87981743]])
```

In [15]:
```
len(eigmatrices)
```

Out[15]:
```
10000
```

In [16]:
```
len(mom_events)
```

Out[16]:
```
10000
```

In [17]:
```
len(mom_events[0])
```

Out[17]:
```
14
```

In [18]:
```
# organized into events
new_momenta_events = []
# not organized into events
new_momenta = []

for i in range(len(mom_events)):

    new_current_event = []

    event = mom_events[i]
    matrix = eigmatrices[i]

    for momentum in event:
        new_momentum = np.matmul(matrix, momentum)
        new_current_event.append(new_momentum)
        new_momenta.append(new_momentum)

    new_momenta_events.append(new_current_event)

new_momenta = np.array(new_momenta)
```

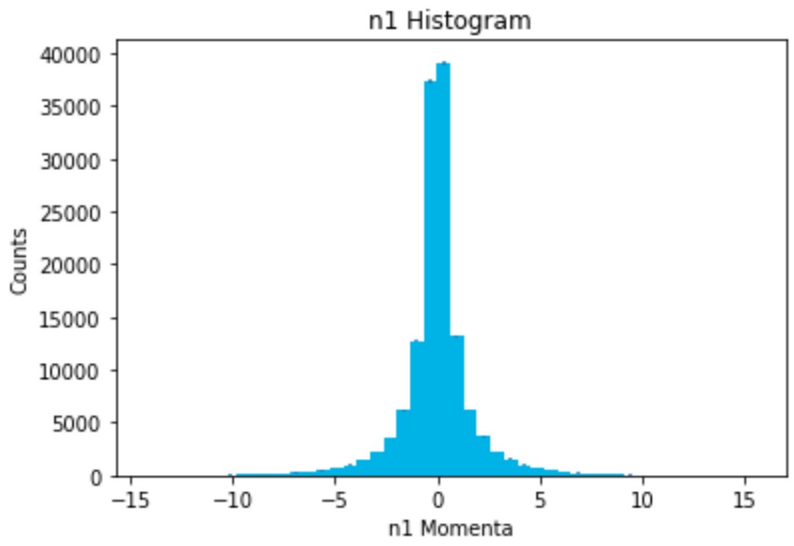
In [19]:
```
new_momenta
```

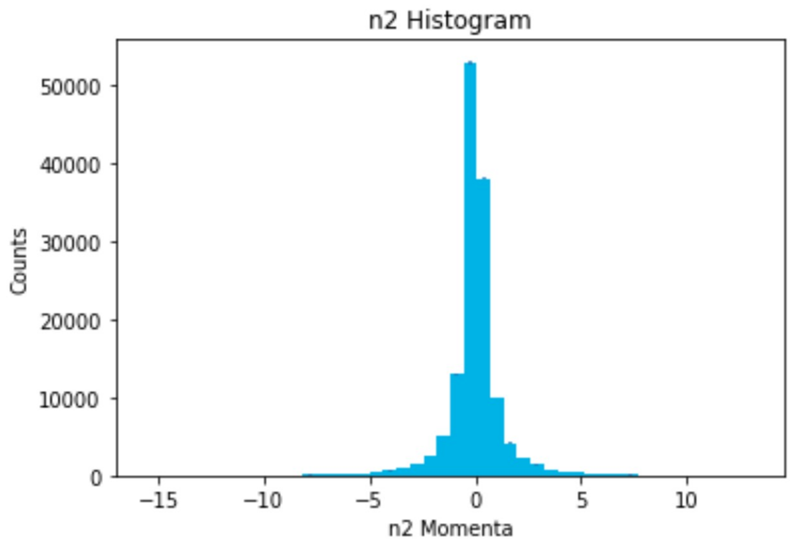
Out[19]:
```
array([[-1.01960396,  0.45612046,  0.06748954],
       [-0.31009369,  0.02580894,  0.17288538],
       [-0.61443908,  0.98340439,  1.3396016 ],
       ...,
       [-0.58730846, -0.32248348,  0.06114448],
       [ 0.71608594, -0.00163219,  0.00219011],
       [ 1.21741694,  0.46306833, -0.26294834]])
```

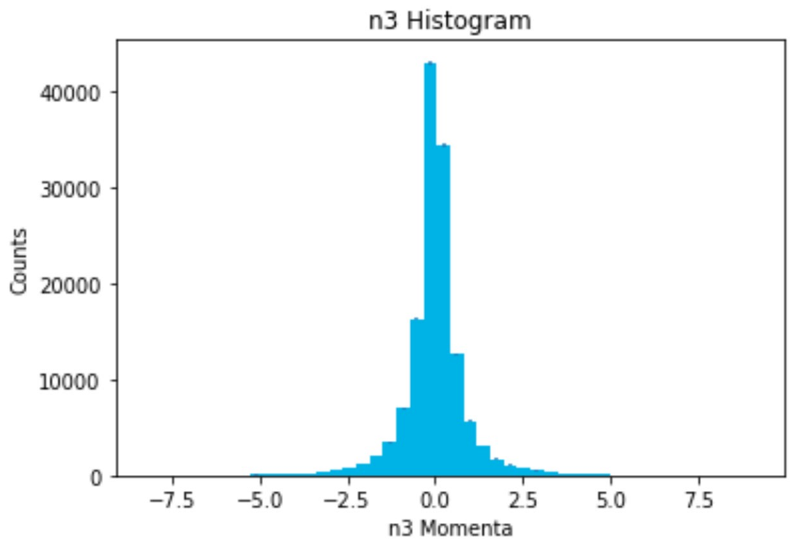
In [20]:
```
n1 = new_momenta[:, 0]
n2 = new_momenta[:, 1]
n3 = new_momenta[:, 2]
```

In [21]:
```
def plot_histogram(samples,xtitle,ytitle, title, nbins):

    #Plot the histogram of the sampled data with nbins and a nice color

    limits = [np.min(samples), np.max(samples)]

    plt.figure()
    n, bins, patches =plt.hist(samples, bins=nbins, range = limits, col
or=(0,0.7,0.9))  #Set the color using (r,g,b) values or
                                                                 #  us
e a built-in matplotlib color"""
    bincenters = 0.5*(bins[1:]+bins[:-1])
    errs = np.sqrt(n)

    plt.errorbar(bincenters, n, yerr=errs, fmt='none')
    #Add some axis labels and a descriptive title
    plt.xlabel(xtitle)
    plt.ylabel(ytitle)
    plt.title(title)

    #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 n, bincenters, patches, errs
```

In [22]:
```
# Histogram of all events' n1, n2, and n3 axes.

nbins = 50
countsn1, bincenters1, patches1, errs1 = plot_histogram(n1, "n1 Moment
a", "Counts", "n1 Histogram", nbins)
countsn2, bincenters2, patches2, errs2 = plot_histogram(n2, "n2 Moment
a", "Counts", "n2 Histogram", nbins)
countsn3, bincenters3, patches3, errs3 = plot_histogram(n3, "n3 Moment
a", "Counts", "n3 Histogram", nbins)
```


n1 Histogram
Counts
n1 Momenta
0
5000
10000
15000
20000
25000
30000
35000
40000
−15
−10
−5
0
5
10
15


n2 Histogram
Counts
n2 Momenta
0
10000
20000
30000
40000
50000
−15
−10
−5
0
5
10


n3 Histogram
Counts
n3 Momenta
0
10000
20000
30000
40000
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5

```
In [23]: print(np.mean(n1))
         print(np.mean(n2))
         print(np.mean(n3))

         0.0005442346795601114
         -0.003628831327892982
         0.0005960696435786939
```

The above plots show that the principle axes momenta are distributed about a Gaussian, with mean ~= 0.

3c.

When physicists observe a new phenomenon, they often like to display individual events to make sure they "look" they way we expect. We can display a single event by making a 3D plot with a vector representing the momentum of each charged particle. Make such four such displays, two for events with $S < 0.05$ and two for events with $S > 0.3$. Do they look the way you expect them to?

```
In [24]: from mpl_toolkits.mplot3d import Axes3D
```

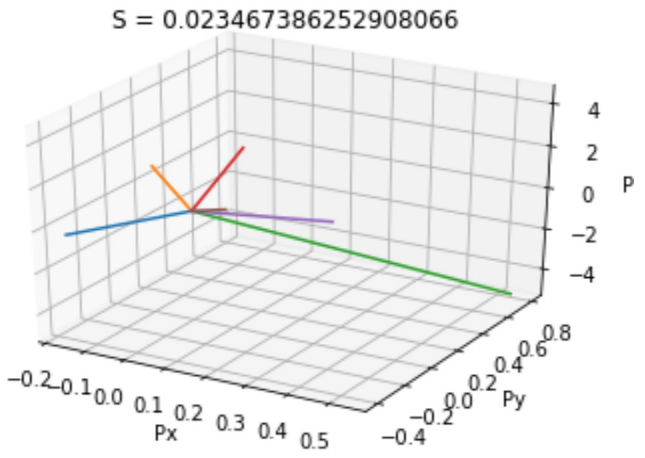
```
In [25]: type(S[0])

Out[25]: numpy.float64
```

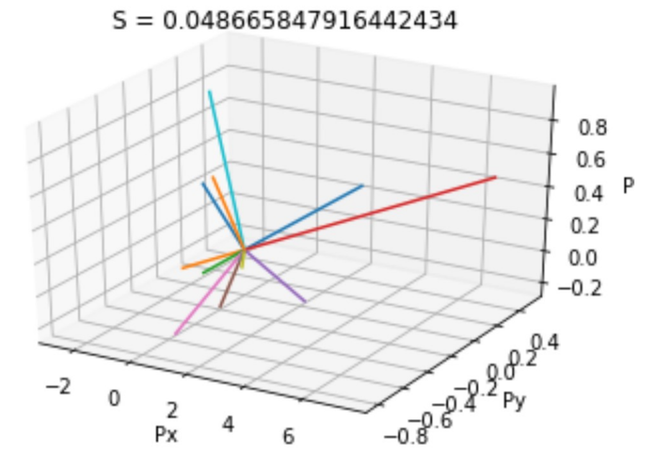
```
In [26]: # S is the sphericity, found from the previous problem
         arrayS = np.array(S)
         smallS = np.argwhere(arrayS < 0.05)
         largeS = np.argwhere(arrayS > 0.3)

         small1 = smallS[0][0]
         small2 = smallS[1][0]
         large1 = largeS[0][0]
         large2 = largeS[1][0]

         indices = [small1, small2, large1, large2]
```

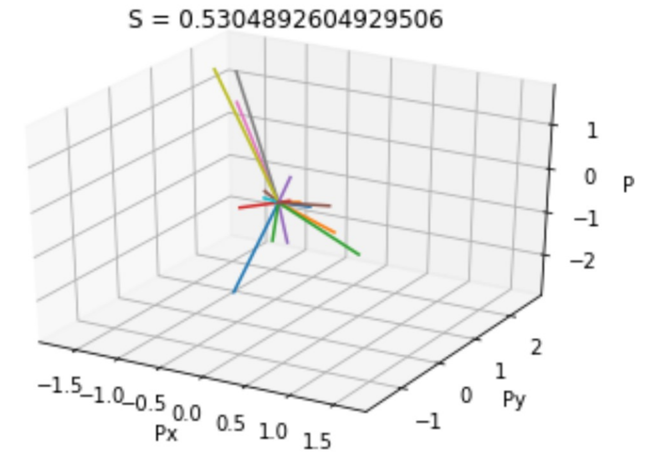
```
In [27]: np.array(mom_events[0])
```

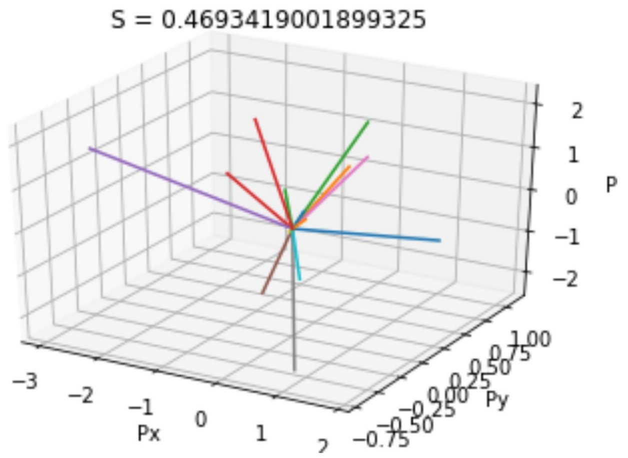
```
Out[27]: array([[ 0.230876 , -1.06223  ,  0.265625 ],
                [-0.100138 , -0.301177 ,  0.161178 ],
                [ 0.143193 , -0.656043 ,  1.63959  ],
                [-0.491384 , -3.39518  ,  1.83697  ],
                [ 0.0266682, -0.0337181, -0.0381424],
                [ 0.133614 ,  0.502632 , -0.218788 ],
                [-0.137105 ,  0.380518 , -0.691605 ],
                [-0.153996 ,  0.316395 , -0.204418 ],
                [ 0.522023 ,  0.479359 ,  0.0370741],
                [-0.389784 , -0.659345 ,  0.240749 ],
                [-0.679757 , -2.81622  ,  1.58652  ],
                [ 0.318718 ,  3.10269  , -1.08666  ],
                [-0.502387 ,  0.0461444,  0.750722 ],
                [-0.440654 ,  0.176097 ,  0.210801 ]])
```

In [28]:
```
for index in indices:
    tempS = arrayS[index]
    momenta = np.array(mom_events[index])
    px = momenta[:, 0]
    py = momenta[:, 1]
    pz = momenta[:, 2]

    fig =  plt.figure()
    title = "S = " + str(tempS)
    ax = fig.add_subplot(111, projection='3d')

    for i in range(len(px)):
        ax.plot([0,px[i]], [0,py[i]], [0,pz[i]])

    ax.set_title(title)
    ax.set_xlabel("Px")
    ax.set_ylabel('Py')
    ax.set_zlabel("Pz")
```


S = 0.023467386252908066
Px
Py
P


S = 0.048665847916442434
Px
Py
P


S = 0.5304892604929506
Px
Py
P



In the plots above, each different color indicates a different momentum per event. As we expect, a higher sphericity corresponds with a more "spherical" spread of vectors, as we can see from the S > 0.3 plots. Low sphericity corresponds with a "flatter" spread of vectors ("more 2D like"), as shown in the S < 0.05 plots. Thus, yes, the events' momentum vectors look how we expect.