```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr  1 14:24:27 2019

@author: naomi
"""
#import libraries:
import numpy as np
from numpy import cos,sin
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy import stats
import time
from numpy.polynomial.polynomial import polyfit
import matplotlib.mlab as mlab
from scipy.stats import norm


#generates a non-uniform distribution between 0 and pi proportional to sin(theta)
#inverse-transformation method
def sin_dis(num):
    u = np.random.uniform(0,1,num)
    theta = np.arccos(1-2*u)
    return theta
```

```
#generates a non-uniform distribution between 0 and pi proportional to sin(theta)
#reject-accpet method
def reject_accept(num):

    ran = []
    # Counter test to calculate the percentage of points used
    naccept=0
    x = np.random.uniform(0,np.pi,num)
    y = np.random.uniform(0,1,num)
    criterion = y < np.sin(x)
    for i in range(num):
        if criterion[i]:
            ran.append(x[i])
            naccept=naccept+1

    percent_accept = (naccept/num)* 100
    return ran, percent_accept

#function that returns the random decay postion and time
def position_decay(num_events=1000000):
    tau = 550E-6
    speed = 2000
    time = np.random.exponential(scale = tau, size = 1000000)
```

```
    position = speed*time

    return time,position


#function that returns the random decay angle

def angle_decay(num):

    theta = sin_dis(num)

    phi =  np.random.uniform(0,2*np.pi,num)

    theta -= np.pi/2

    return theta,phi


#function that returns the hit position on the detector

def position_on_detector(theta,phi):

    time,position = position_decay(num_events=1000000)

    rho = (2 - position)/cos(theta)

    x = rho*sin(theta)*cos(phi)

    y = rho*sin(theta)*sin(phi)

    #Resolution of the detector causes a smear on the gaussian distribution of the hit

positions

    res_x = 0.1

    res_y = 0.3

    smear_x = x + np.random.normal(0, res_x, 1000000)

    smear_y = y + np.random.normal(0,res_y, 1000000)
```

```
    return x,y,smear_x, smear_y


#Test of the distribution for a unit sphere of a uniform distribution
def uniform_dist(num):
    theta = np.random.uniform(0,np.pi,num) # Opening angle to beam
    phi = np.random.uniform(0,2*np.pi,num) # Polar angle on screen, about axis of
beam

    x1 = sin(theta)*cos(phi)
    y1= sin(theta)*sin(phi)
    z1 = cos(theta)
    return x1,y1,z1


#Test of the distribution for a unit sphere of a non-uniform distribution
def non_uniform_dist(num):
    theta = sin_dis(num) # Opening angle to beam
    phi = np.random.uniform(0,2*np.pi,num)

    #where rho is 1 in spherical coordinates
    x2 = sin(theta)*cos(phi)
    y2 = sin(theta)*sin(phi)
    z2 = cos(theta)
```

```
    return x2,y2,z2


#Collider experiment to calculate the cross section of total number of candidate events
observed is 5 with 95 percent confidence level
def confidence_interval(mincross, maxcross):
    steps = 100
    repeats = 10000

    percent = []
    over5 = []
    X = []

    for q in range(steps +1):
        total_signal = []
        background = []
        signal = []

        for i in range(repeats):
            bg_noise = np.random.normal(5.7,0.4)#background noise found using a
Gaussian distribution
            lum_error = np.random.normal(12,0.5)#Integrated luminosity uncertainty found
using a Gaussian distribution
```

```
            cross_sec = mincross + (maxcross-mincross)*q/steps #This allows zooming in
around in a certain range of cross sections
            bg_prod =  np.random.poisson(bg_noise)#Poisson variation in the background
production
            lum = np.random.poisson(lum_error*cross_sec)#Poisson variation in the
signal production
            combined = np.random.poisson(lum_error*cross_sec + bg_noise )#combined
signal production using a poisson distribution
            total_signal.append(combined)
            signal.append(lum)
            background.append(bg_prod)

        x = sum(float(n) > 5 for n in total_signal)
        if 100*x/repeats > 95:
            X.append(cross_sec)

        over5.append(100*x/repeats)
        percent.append(mincross+(maxcross-mincross)*q/steps)

    return over5, percent, total_signal, background, signal, X


MyInput = '0'
```

```
while MyInput != 'q':
    MyInput = input('Enter a choice, \n "a)" to investigate the analytical method,\n "b)"
to investigate the reject-accept method and compare the two methods, \n "c)" to
explore the partical experiment, \n "d)" to explore the statistical investigation, \n "q)" to
quit: ')
    print('You entered the choice: ',MyInput)


    if MyInput == 'a':
        print('You have chosen to generate random angles 0 < theta < pi  in a distribution
proportional to sin using the inverse transformation method.')


        print('Probablity density function of the generated deviates compared to the sin(x)
function using the inverse transformation method:')
        num_bins = 100
        theta = sin_dis(100000)
        x = np.linspace(0, np.pi, 100)


        sinx =  1/2 * np.sin(x)
        plt.plot(x, sinx, linewidth=2, color = "red", label="$\\sin(\\theta)$")
        n, bins, patches = plt.hist(theta, num_bins , density = 0.5, facecolor='blue',
alpha=0.7, label="sinusoidal random \n number")
        plt.xlim([0,np.pi])
```

```
    plt.legend(loc="upper left", fontsize="x-small", borderpad=1)
    plt.xlabel("Random angle, $\\theta$")
    plt.ylabel("Normalized frequency")
    plt.show()
    print('Test of the difference between the probablity density function of the generated deviates for each bin and sin(x) function using the inverse transformation method:')
    plt.subplot(2,1,2)
    plt.bar(x, (sinx-n), width = 0.01, align='center', alpha=1)
    plt.xlabel("Random angle, $\\theta$")
    plt.ylabel("Difference between \n PDF and sin(x)")
    plt.show()



    print('The first 4 statistical moments of the sinusiodal distribution between 0 and pi')
    print("Number","  ", "Moment")     #table column headings
    for x in range(1,5):
        print(x,'      ',stats.moment(theta, moment = x))
  elif (MyInput =='b'):
    print('Probablity density function of the generated deviates compared to the sin(x) function using the reject and accept method:')
    theta, percent_accept = reject_accept(num=100000)
```

```
print('The percent of points accpeted using the reject-accept method:', percent_accept, '%')
num_bins = 100
n, bins, patches = plt.hist(theta, num_bins, color="red", density = 1, alpha=0.7, label="Sinusoidal random number")

x = np.linspace(0, np.pi,100)

sinx =  1/2* np.sin(x)
plt.plot(x, sinx, linewidth=2, color="blue", label="$\\sin(\\theta)$")
plt.xlim([0,np.pi])

plt.xlabel("Random angle, $\\theta$")
plt.ylabel("Normalized Frequency")

plt.legend(loc="upper left", fontsize="x-small", borderpad=1)

plt.show()

print('Test of the difference between the probability density function of the generated deviates for each bin and sin(x) function:')
plt.subplot(2,1,2)
plt.bar(x, (sinx-n), width = 0.01, align='center', alpha=1)
```

```
plt.xlabel("Random angle, $\\theta$")

plt.ylabel("Difference between \n PDF and sin(x)")

plt.show()



print('The first 4 statistical moments of the sinusiodal distribution between 0 and pi
')

print("Number","  ", "Moment")     #table column headings

for x in range(1,5):

    print(x,'       ',stats.moment(theta, moment = x))


N = []

t_analytic = []

t_reject = []


print('Please wait while the speed effiecncy of the two methods are compared:')

for num in range(10000,1000000,10000):

    start1 = time.time()

    x = sin_dis(num)

    end1 = time.time()


    start2 = time.time()

    x = reject_accept(num)
```

```
            end2 = time.time()


            t_analytic.append(end1-start1)
            t_reject.append(end2-start2)
            N.append(num)



        plt.plot(np.unique(N), np.poly1d(np.polyfit(N, t_analytic, 1))(np.unique(N)))
        plt.plot(N,t_analytic, color = "b",label = "Analytic")
        plt.plot(np.unique(N), np.poly1d(np.polyfit(N, t_reject, 1))(np.unique(N)))
        plt.plot(N,t_reject, color = "r",label = "Reject")
        plt.legend(title = "Method",fontsize = 15)
        plt.xlabel('Number of angles generated',fontsize = 15)
        plt.ylabel('Time (s)',fontsize = 15)
        plt.show()

    elif (MyInput =='c'):

        time,position = position_decay(num_events=1000000)
        print('Exponential distribution of the random decay times and the random decay
positions ')
        plt.subplot(2,1,1)
        plt.hist(time, 100 ,color="pink")
```

```
#plt.xlim([0,0.0025])
plt.xlabel("Time after injection, s$^{-1}$")
plt.ylabel("Relative Frequency")
plt.title("Random Decay Times")

plt.subplot(2,1,2)
plt.hist(position, 100, color="purple")
plt.xlabel("Position from beam injection, m$^{-1}$")
#plt.xlim([0,6])
plt.ylabel("Relative Frequency")
plt.title("Random Decay Position")

plt.tight_layout()
plt.show()



theta,phi = angle_decay(num = 1000000)
print('Unit sphere of randomly distribution angles with a uniform distribution and
non-uniform distribution using the inverse transformation method ')

num = 10000
x1,y1,z1 = uniform_dist(num)
x2,y2,z2 = non_uniform_dist(num)
```

```
fig = plt.figure(figsize=plt.figaspect(0.5))
ax1 = fig.add_subplot(1,2,1, projection='3d')
ax2 = fig.add_subplot(1,2,2, projection='3d')

ax1.scatter(x1,y1,z1, s = 0.1)
ax1.set_xlabel("x",fontsize = 15)
ax1.set_ylabel("y",fontsize = 15)
ax1.set_zlabel("z",fontsize = 15)
ax1.set_title("Uniform distribution",fontsize = 15)


ax2.scatter(x2,y2,z2, s = 0.1)
ax2.set_title("Non-uniform distribution",fontsize = 15)
ax2.set_xlabel("x",fontsize = 15)
ax2.set_ylabel("y",fontsize = 15)
ax2.set_zlabel("z",fontsize = 15)
plt.show()

print('Histogram plot of decay angles phi and theta in a random distribution ')

plt.hist2d(phi, theta, bins=100, cmap=plt.cm.BuPu)
plt.xlabel("Angle $\\theta$")
```

```
plt.ylabel("Angle $\\varphi$")



cbar = plt.colorbar()
cbar.solids.set_edgecolor("face")
plt.draw()
plt.show()




x,y,smear_x,smear_y = position_on_detector(theta,phi)
# Range limits of the detector
xmin = -1
xmax = 1
ymin = - 1
ymax = 1
detector_range = [[xmin,xmax],[ymin,ymax]]



print('2D Histogram plot of the detector hits without the smearing due to resolution')

plt.hist2d(x, y,bins=40, range=detector_range, cmap=plt.cm.BuPu)
```

```
plt.xlabel("$X$ position on detector, m")
plt.ylabel("$Y$ position on detector, m")
cbar = plt.colorbar()
cbar.solids.set_edgecolor("face")
plt.draw()
plt.show()

print('2D Histogram plot of the detector hits with smearing due to resolution of x= 0.1m and y = 0.3m ')


plt.hist2d(smear_x, smear_y,bins=40, range=detector_range, cmap=plt.cm.BuPu)
plt.xlabel("$X$ position on detector, m")
plt.ylabel("$Y$ position on detector, m")

cbar = plt.colorbar()
cbar.solids.set_edgecolor("face")
plt.draw()
plt.show()

meanx = np.mean(smear_x)
variancex = np.var(smear_x)
sigmax = np.sqrt(variancex)
```

```
print('1D Histogram plot of the detector hits in the x with smearing due to resolution of x= 0.1m ')

plt.hist(smear_x,bins=np.arange(-1, 1,0.01), density =1)
plt.xlabel("$X$ position on detector, m")
plt.ylabel("Frequency of hit on detector")
plt.show()

print('The mean value is',meanx,'the variance is',variancex, 'and the standard deviation is ',sigmax)

print('1D Histogram plot of the detector hits in the y with smearing due to resolution of y= 0.3m ')
plt.hist(smear_y,bins=np.arange(-1, 1,0.01), density =1)
plt.xlabel("$Y$ position on detector, m")
plt.ylabel("Frequency of hit on detector")
plt.show()

meany = np.mean(smear_y)
variancey = np.var(smear_y)
sigmay = np.sqrt(variancey)
```

print('The mean value is',meany,'the variance is',variancey, 'and the standard deviation is ',sigmay)

elif (MyInput =='d'):

over5, percent, total_signal, background, signal , X = confidence_interval(mincross = 0.01 ,maxcross = 1)

print('The first cross section that produces a 95% confidence level that the event is over 5 is', min(X), 'nb')

print('Confidence level over 5 events for a range of cross sections, the red dashed line marks a 95% confidence interval with a range of 0.01 to 1:')

plt.hlines(y = 95, xmin = 0.01, xmax= 1, color='red', linestyles='dashed', label='95 confidence')

plt.xlabel("Diameter of cross section")

plt.ylabel("Perecent of events that occur over 5")

plt.plot(percent,over5)

plt.show()

```
over5, percent, total_signal, background, signal , X =
confidence_interval(mincross = 0.35 ,maxcross = 0.45)

print('Confidence level over 5 events for a range of cross sections, the red dashed
line marks a 95% confidence interval with a range of 0.35 to 0.45:')
plt.hlines(y = 95, xmin = 0.35, xmax= 0.45, color='red', linestyles='dashed',
label='95 confidence')
plt.xlabel("Diameter of cross section")
plt.ylabel("Perecent of events that occur over 5")
plt.plot(percent,over5)
plt.show()

over5, percent, total_signal, background, signal , X =
confidence_interval(mincross = 0.01 ,maxcross = 1)

num_bins = 100

print('Background and signal production modelled using Poisson distributions')
n, bins, patches = plt.hist(background, num_bins, density = 1, color="red",
alpha=0.5, label = 'Background')
n, bins, patches = plt.hist(signal,num_bins, density = 1, color="blue",
alpha=0.5,label = 'Signal')
plt.xlabel('Number of candiate events')
```

```
plt.ylabel('Production Frequency')
plt.legend(loc="upper right", fontsize="medium", borderpad=1)
plt.show()

meanb = np.mean(background)
varianceb = np.var(background)
sigmab = np.sqrt(varianceb)

means = np.mean(signal)
variances = np.var(signal)
sigmas = np.sqrt(variances)

print('The mean value of the background is',meanb,'the variance  is',varianceb,
'and the standard deviation is ',sigmab)
print('The mean value of the signal is',means,'the variance is',variances, 'and the
standard deviation is ',sigmas)

print('The combined background and signal production modelled using Poisson
distributions')
num_bins = 100
n, bins, patches = plt.hist(total_signal, num_bins, density =1, color="red",
alpha=0.5)
plt.xlabel('Number of candiate events')
```

```
        plt.ylabel('Production Frequency')
        plt.show()


        meant = np.mean(total_signal)
        variancet = np.var(total_signal)
        sigmat = np.sqrt(variancet)


        print('The mean value of the total signal is',meant,'the variance is',variancet, 'and
the standard deviation is ',sigmat)
    elif MyInput != 'q':
         print('This is not a valid choice')


print('You have chosen to finish - goodbye.')
```