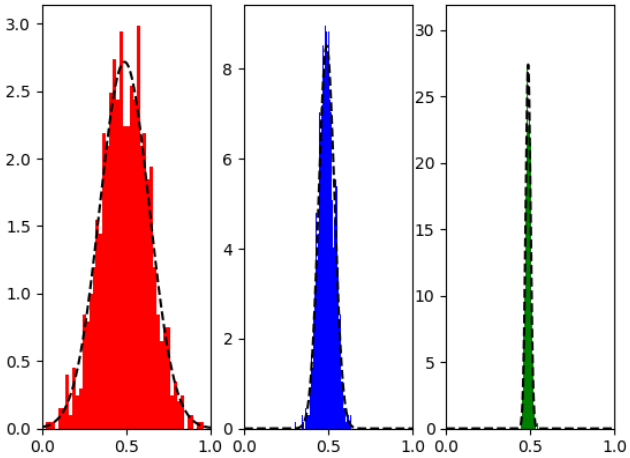
Homework 10

Ryan Branagan

April 9, 2019

Problem 1

```
#Ryan Branagan
#Collaborators: N/A
#Branagan_hw10_p1.py
#4/2/19

import numpy as np
import pylab as p
#%%
#Define Monte Carlo and Function
def MC(f,a,b,N):
    h = ((b-a)/N)
    x = np.array([np.random.uniform(a,b) for i in range(N)])
    return np.sum(f(x)*h)

def Fres(x):
    return np.cos((np.pi*(x**2))/2)

def Gaussian(x,Sigma,Mu):
    return (1/(np.sqrt(2*np.pi)*Sigma))*np.exp(-1*(((x-Mu)**2)/(2*(Sigma**2))))
#%%
#Parameters
f = Fres
a = 0
b = 2
Ns = np.array([10**2,10**3,10**4])
Trials = 1000

#Get results
Results = np.array([np.zeros(Trials),np.zeros(Trials),np.zeros(Trials)])
for i,N in enumerate(Ns):
```




```
    for j in range(Trials):
        Results[i,j] = MC(f,a,b,N)

#Gaussian stuff
STD1 = np.std(Results[0])
STD2 = np.std(Results[1])
STD3 = np.std(Results[2])
Mu1 = np.mean(Results[0])
Mu2 = np.mean(Results[1])
Mu3 = np.mean(Results[2])

#Plotting
fig,(ax1,ax2,ax3) = p.subplots(1,3)
ax1.hist(Results[0],bins=50,color="red",normed=True)
ax2.hist(Results[1],bins=50,color="blue",normed=True)
ax3.hist(Results[2],bins=50,color="green",normed=True)
x = np.linspace(a,b,Ns[2])
ax1.plot(x,Gaussian(x,STD1,Mu1),'k--')
ax2.plot(x,Gaussian(x,STD2,Mu2),'k--')
ax3.plot(x,Gaussian(x,STD3,Mu3),'k--')
ax1.set_xlim(0,1)
ax2.set_xlim(0,1)
ax3.set_xlim(0,1)
```

The Gaussian and histogram in each case have excellent agreement. Sometimes the bars go lower or higher than the Gaussian but still overall conform to the same shape. As the number of trials goes up they should have even better agreement since there should

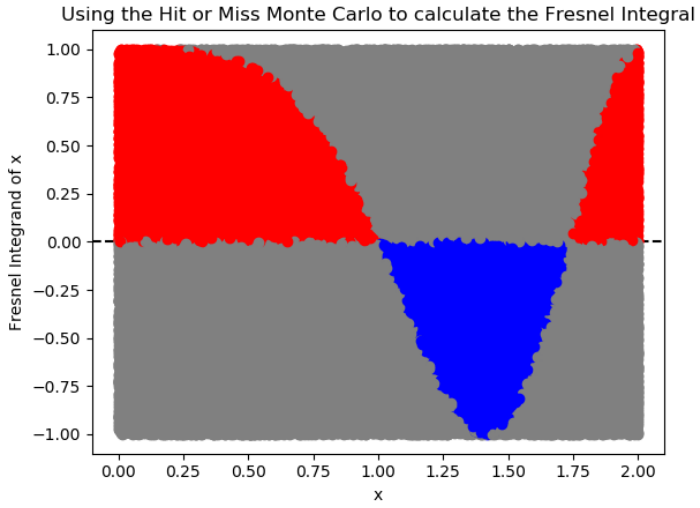
be a better average. A Gaussian works because we have a certain correct value that we are likely to get with this method, but since it is random there are deviations from this correct value. The average is our result for the integral and the standard deviation is more a measure of how accurate our answer is than a measure of the error. with very big sub-intervals I could have large error but get a nice grouping of numbers with little deviation. Since the error for the Monte Carlo method is $N^{-1/2}$ to be accurate to three decimal places I need an error of 10^{-4} which means N should be on the order of 10^8. The answer for the Fresnel integral is 0.48.

Problem 2

```
#Ryan Branagan
#Collaborators: N/A
#Branagan_hw10_p2.py
#4/8/19

import numpy as np
import pylab as p
#%%
#Define important things
def Fres(x):
    return np.cos((np.pi*(x**2))/2)

def HitorMiss(f,a,b,c,d,axis):
    x = np.random.uniform(a,b)
    y = np.random.uniform(c,d)
    T = f(x)
    if y > 0:
        if y > T:
            g = 0
            axis.plot(x,y,color='gray',marker='o')
        if y <= T:
            g = 1
            axis.plot(x,y,'ro')
    if y < 0:
        if y < T:
            g = 0
            axis.plot(x,y,color='gray',marker='o')
        if y >= T:
            g = -1
            axis.plot(x,y,'bo')
    if y == 0:
        g = 0
```

```
        axis.plot(x,y,'ko')
    return g
#%%
#Parameters
f = Fres
a = 0
b = 2
c = -1
d = 1
N = 10**4

#Setup
fig,ax = p.subplots(1,1)
ax.set_title("Using the Hit or Miss Monte Carlo to calculate the Fresnel Integral")
ax.set_xlabel("x")
ax.set_ylabel("Fresnel Integrand of x")
x = np.linspace(a,b,N)
ax.plot(x,Fres(x),'k--')
ax.axhline(0, color='k', linestyle='--')
Np = 0
Neg = 0

#Results
for i in range(N):
    z = HitorMiss(f,a,b,c,d,ax)
    if z > 0:
        Np = Np + z
    if z < 0:
        Neg = Neg - z

Result = (4*(Np-Neg))/N
print(Result)
```

From experimenting I believe the error for the hit or miss Monte Carlo is the same as the mean Monte Carlo; $N^{-1/2}$. Like previously said, to obtain an accuracy of three digit places you would need N to be 10^8. It was not necessary but I wanted to plot the points that my hit or miss was creating. So I wrote my code in a way that positive points under the curve would be red, negative points above the curve would be blue, any points outside of the curve would be gray, and if $y = 0$ was randomly selected it would be colored black and would count as a miss. I decided to make it count as a miss because thinking about an integral, $f(x) = 0$ does not contribute to area under the curve. Even so, I have never seen a black dot in all the times I've ran the code. I understand that the way I have written it is very inefficient and could be done much better but it was the my first idea and I went with it.

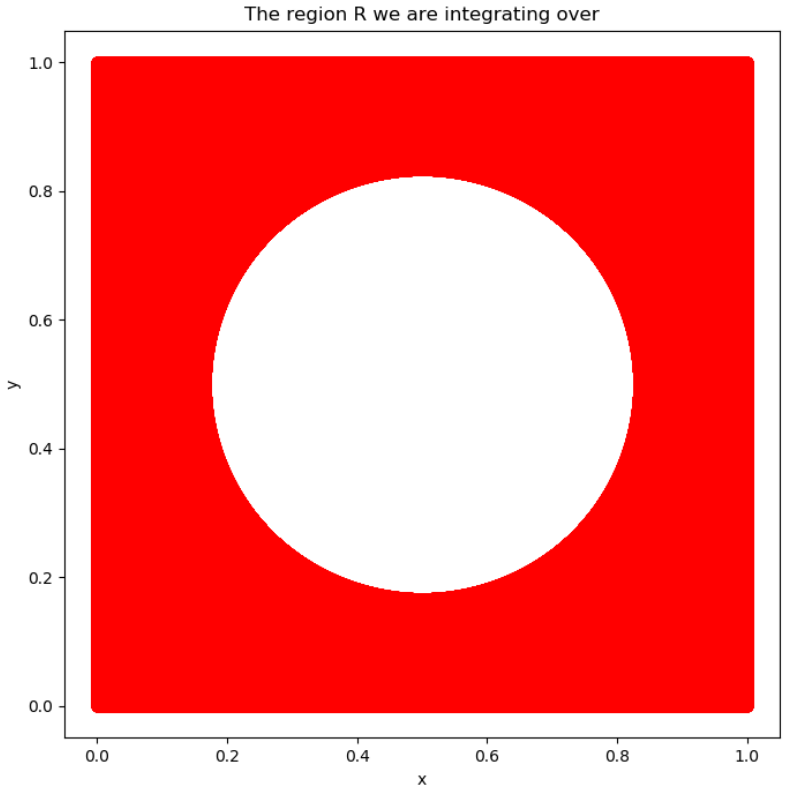


Problem 3

```
#Ryan Branagan
#Collaborators: N/A
#Branagan_hw10_p3.py
#4/8/19

import numpy as np
import pylab as p
#%%
#Define things
def MC2D(f,a,b,c,d,N):
    #Regular Monte Carlo Stuff
    dx = (b-a)/N
    dy = (d-c)
    Points = np.array([[np.random.uniform(a,b),np.random.uniform(c,d)] for i in range(N)]
    #Nonrectangular Region
    for P in Points:
        x = P[0]
        y = P[1]
        while np.sqrt(((x-(1/2))**2)+((y-(1/2))**2)) <= (1/3):
            x = np.random.uniform(a,b)
            y = np.random.uniform(c,d)
        P[0] = x
        P[1] = y
    #Unpack x and y
    xlist = Points[:,0]
```

```
    ylist = Points[:,1]
    #Monte Carlo Integration
    MC = np.sum(f(xlist,ylist)*dx*dy)
    return xlist,ylist,MC

def func(x,y):
    return np.log10(1+x*y)
#%%
#Parameters
f = func
a = 0
b = 1
c = 0
d = 1
N = 10**8

xs,ys,Ans = MC2D(f,a,b,c,d,N)

fig,ax = p.subplots(1,1,figsize=(8,8))
ax.plot(xs,ys,'ro')
ax.set_title("The region R we are integrating over")
ax.set_xlabel("x")
ax.set_ylabel("y")

print("The answer accurate to "+str(int(np.log10(N)/2)-1)+" digits: "+str(round(Ans,int(
```

As we have discussed, the error for the mean Monte Carlo method in any dimension is still $N^{-1/2}$. The question asks for the answer accurate to three significant digits. Using very large N I can tell that the answer starts with 0.08, meaning eight is the first significant digit. This means to get the answer to three significant digits, I need my answer to be accurate to four digit places. Furthermore, this means my error needs to be the fifth digit. Altogether all this means my N should be 10^10. I may be misunderstanding something and instead only need 10^8. It took a long time but I was able to do this, and the answer that I got was: 0.8838. One suitable way of completing this problem would be to find the integral over the square then subtract the integral over the circle. Instead, I wanted to make it so that if a point was inside the circle, it got change to a different point. To me this seems like a better way to doing it since N will always be your number of points instead of having N points over a larger area and then subtracting N points from a smaller area.


The region R we are integrating over
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x