**Jinghui Yuan 4177204**

**Problem 1**

We want to simulate a discrete random variable X that has probability mass function (pmf) given by:

P(X = 0) = 0.3; P(X = 1) = 0.2; and P(X = 3) = 0.5 using R and Python.

First, calculate the cdf of X as:

Thus, the inverse function is:

Then, let X equal to .

**R code:**

> discrete.random.variable<-function(n){

+ k<-0

+ x<-numeric(n)

+ for (k in 1:n){

+ u<-runif(1)

+ if (u<=0.3){

+ x[k]<-0

+ }

+ if (u>0.3 & u<=0.5){

+ x[k]<-1

+ }

+ if (u>0.5 & u<=1){

+ x[k]<-3

+ }

+ k<-k+1

+ }

+ return(x)

+ }

> discrete.random.variable(10)

[1] 1 3 3 3 0 3 0 3 1 3

**Python code:**

def discrete\_random\_variable(n):

k=0

x\_list=[]

while k < n :

u=np.random.uniform(0,1,1)

if u<=0.3:

x=0

elif u<=0.5:

x=1

else:

x=3

x\_list.append(x)

k=k+1

return x\_list

print(discrete\_random\_variable(10))

[0, 0, 3, 3, 0, 3, 1, 0, 1, 0]

**Problem 2**

We want to generate standard normal random variables using the Box-Muller algorithm discussed in class.

1. Show that the variables X1 and X2 are independent standard normal

Since u and v are generated from independent uniform distribution (0,1)

Thus, it can be concluded that X1 and X2 are independent standard normal.

2. Write codes in R and python to generate standard normal random variables using the Box-Muller algorithm

First, sample u and v from uniform (0, 1). Second, set and .

Finally, take and

**R code:**

> standard.random.normal<-function(n){

+ k<-0

+ x1<-numeric(n)

+ x2<-numeric(n)

+ for (k in 1:n){

+ u<-runif(1)

+ v<-runif(1)

+ r<-sqrt(-2\*log(v))

+ theta<-2\*pi\*u

+ x1[k]<-r\*cos(theta)

+ x2[k]<-r\*sin(theta)

+ k<k+1

+ }

+ return(list(x1,x2))

+ }

> standard.random.normal(10)

[[1]]

[1] -1.36309900 1.13583976 0.79999918 0.05690347 0.51867349 0.28534902 -0.29547890 -0.53596514

[9] 0.60864782 -0.01156335

[[2]]

[1] 1.18285713 -0.02747868 0.21771275 0.78452526 0.37453761 -2.23978007 1.35446867 -1.51944311

[9] 1.30403246 0.05714001

**Python code:**

def standard\_random\_normal(n):

x1\_list=[]

x2\_list=[]

k=0

while k < n :

u=np.random.uniform(0,1,1)

v=np.random.uniform(0,1,1)

r=np.sqrt(-2\*np.log(v))

theta=2\*np.pi\*u

x1=r\*np.cos(theta)

x2=r\*np.sin(theta)

x1\_list.append(x1)

x2\_list.append(x2)

k=k+1

return x1\_list, x2\_list

print(standard\_random\_normal(10))

([array([ 0.22834072]), array([-2.66971688]), array([-0.29847637]), array([-0.84318436]), array([-1.12416127]), array([ 0.80803898]), array([ 0.2622776]), array([ 0.04398038]), array([ 0.86983417]), array([ 1.75519074])], [array([-0.5869729]), array([-0.41886163]), array([ 0.2720857]), array([ 0.54090716]), array([ 0.88987996]), array([-1.7354779]), array([-1.00436624]), array([ 0.2360808]), array([ 0.8185115]), array([-0.66703408])])

**Problem 3**

Following the discussion in class, we are interested in using the following data 4, 4, 5, 3, 1, 4, 3, 6, 5, 2 to test the following null and alternative hypotheses about the Monte Carlo testing of the Poisson problem

H0 : λ = 2 vs H1 : λ > 2.

Using R and Python,

1. Perform the test using basic Monte Carlo
2. Based on the given sample, calculate the mean value, as well as the statistical value deta0
3. Generate 10 random samples from poison distribution (2), and calculate the mean value
4. Calculate the statistical value deta
5. Compare the observed deta0 and the sampled deta, if deta<deta0, then add 1
6. Finally, divide the total number of reject samples by the total number we can get the p value
7. Since the p value less than 0.05, then reject the null hypothesis.

**R code:**

> MC\_poisson<-function(n){

+ xbar<-mean(4,4,5,3,1,4,3,6,5,2)

+ deta0<-exp(10\*xbar-2\*10)\*(xbar/2)^(-10\*xbar)

+ count<-0

+ for (i in 1:n){

+ x<-rpois(10,2)

+ lambda<-mean(x)

+ deta<-exp(10\*lambda-2\*10)\*(lambda/2)^(-10\*lambda)

+ if (deta < deta0){

+ count<-count+1

+ }

+ i<-i+1

+ }

+ p<-count/n

+ return(p)

+ }

> MC\_poisson(1000)

[1] 0.001

**Python code:**

def MC\_poisson(n):

xbar0=np.mean([4,4,5,3,1,4,3,6,5,2])

deta0=np.exp(10\*xbar0-2\*10)\*(xbar0/2)\*\*(-10\*xbar0)

count=0

k=0

while k < n :

x=np.random.poisson(2,10)

xbar=np.mean(x)

deta=np.exp(10\*xbar-2\*10)\*(xbar/2)\*\*(-10\*xbar)

if deta<deta0:

count=count+1

k=k+1

p=count/n

return(p)

print(MC\_poisson(10000))

0.0004

1. Perform the test using importance sampling method with proposal distribution from lecture
2. Based on the given sample, calculate the mean value, as well as the statistical value deta0
3. Generate 10 random samples from poison distribution (2.4653), based on the significance sampling method calculate the modified mean value (mubar).
4. Calculate the statistical value deta
5. Compare the observed deta0 and the sampled deta, if deta<deta0, then add 1
6. Finally, divide the total number of reject samples by the total number we can get the p value
7. Since the p value less than 0.05, then reject the null hypothesis.

**R code:**

> IS\_poisson<-function(n){

+ xbar<-mean(4,4,5,3,1,4,3,6,5,2)

+ deta0<-exp(10\*xbar-2\*10)\*(xbar/2)^(-10\*xbar)

+ count<-0

+ for (i in 1:n){

+ x<-rpois(10,2.4653)

+ mu<-0

+ for (j in x){

+ ir<-(exp(-2)\*2^j)/(exp(-2.4653)\*2.4653^j)

+ mu<-mu+j\*ir

+ }

+ mubar<-mu/10

+ deta<-exp(10\*mubar-2\*10)\*(mubar/2)^(-10\*mubar)

+ if (deta < deta0){

+ count<-count+1

+ }

+ i<-i+1

+ }

+ p<-count/n

+ return(p)

+ }

> MC\_poisson(10000)

[1] 1e-04

**Python code:**

def IS\_poisson(n):

xbar0=np.mean([4,4,5,3,1,4,3,6,5,2])

deta0=np.exp(10\*xbar0-2\*10)\*(xbar0/2)\*\*(-10\*xbar0)

count=0

k=0

while k < n :

x=np.random.poisson(2.4653,10)

mu=0

for j in x:

ir=(np.exp(-2)\*2\*\*j)/(np.exp(-2.4653)\*2.4653\*\*j)

mu=mu+j\*ir

xbar=mu/10

deta=np.exp(10\*xbar-2\*10)\*(xbar/2)\*\*(-10\*xbar)

if deta<deta0:

count=count+1

k=k+1

p=count/n

return(p)

print(IS\_poisson(100000))

0.0

1. Does the importance sampling, with this proposal, reduce the variance?

Yes.

**Problem 4**

We want to generate Normal random variables from the double exponential. Write R and Python codes to generate normals using the Accept-Reject Algorithm.

The pdf of standard normal distribution is . Let g(x) be the standard double exponential density, . Then for all , so , A random x from g(x) is accepted if

Develop a function of number of required normal random variables as:

**R code:**

> random.normal<-function(n){

+ k<-0

+ y<-numeric(n)

+ for(k in 1:n){

+ u<-runif(1)

+ x<-rdoublex(1,mu=0,lambda=1) #random variable from standard double exponential distribution#

+ if (exp(-0.5\*x^2+abs(x)-0.5)>u){

+ k<-k+1

+ y[k]<-x

+ }

+ }

+ return(y)

+ }

>

> random.normal(10)

[1] 0.00000000 0.95092335 0.00000000 0.00000000 0.07054515 -0.91809021 0.00000000 0.03188382

[9] 0.00000000 -0.81547127 -1.02978778

**Python code:**

def random\_normal(n):

k=0

y\_list=[]

while k < n :

u=np.random.uniform(0,1,1)

x=np.random.laplace(0,1,1)

if np.exp(-0.5\*x\*\*2+np.abs(x)-0.5)>u:

y=x

k=k+1

y\_list.append(y)

return y\_list

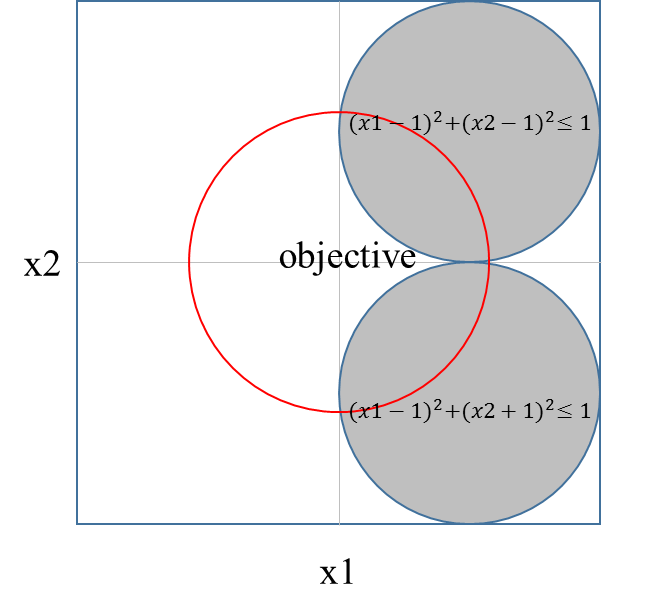
print(random\_normal(10))

[array([ 0.70016431]), array([ 1.08573093]), array([ 1.08573093]), array([ 0.40175184]), array([ 0.40175184]), array([-1.77403244]), array([-0.66013667]), array([ 1.22557586]), array([-1.03122574]), array([-1.02751384]), array([-1.02751384]), array([ 0.45951476]), array([ 1.1191334])]

**Problem 5**

Solve problem 5.26 on page 282 of Boyd and vandenberghe textbook

1. The objective function is as shown in the following figure. The optimal point is the intersection point of two constraint functions, which is (1, 0) and the optimal value is 12+02=1.



1. The KKT conditions are

For the optimal point (), these conditions could be calculated as

There is no solution.



The Lagrange dual problem is given by

Max

s.t.

Since L is symmetric, the optimum occurs with. The dual function then derived as

This formula tends to be 1 as approaching , then the optimal value equal to 1, but the dual optimum is not attained.

**Problem 6**

Simulate a continuous Exponential-Gamma mixture. Suppose that the rate parameter Λ has Gamma(r, β) distribution and Y has Exp(Λ) distribution. That is, (Y |Λ = λ) ∼ fY (y|λ) = λe−λy. Generate 1000 random observations from this mixture with r = 4 and β = 2.

As mentioned in the problem, a random variable lamda from Gamma(4,2) was generated, then the random y could be generated by exponential distribution with the rate of lamda.

**R code:**

> exponential.gamma<-function(n){

+ k<-0

+ y<-numeric(n)

+ for(k in 1:n){

+ lamda<-rgamma(n=1,shape=4,rate=2) #generate random gamma distribution#

+ y[k]<-rexp(1,rate=lamda) #generate random exponential distribution#

+ k<k+1

+ }

+ return(y)

+ }

> exponential.gamma(10)

[1] 2.51701657 0.03657927 0.51534908 0.27734226 0.48515090 0.87010883 1.33300321 0.06721030 0.85019554

[10] 0.33986487

**Python code:**

def exponential\_gamma(n):

k=0

y\_list=[]

while k < n :

lamda=np.random.gamma(4,2,1)

y=np.random.exponential(lamda,1)

k=k+1

y\_list.append(y)

return y\_list

print(exponential\_gamma(10))

[array([ 31.41152934]), array([ 0.3907319]), array([ 20.8803253]), array([ 8.92420356]), array([ 20.92430862]), array([ 3.12064193]), array([ 3.98599205]), array([ 0.14138828]), array([ 4.14760382]), array([ 2.41899164])]