MA226 : Monte-Carlo Simulation
Continuous Random Number Generation

Assignment 4

Turkhade Hrushikesh Pramod
150123044

09-02-2017

1 Problem 1

We have to generate exponential random variable using Inverse Transform Method.Let, $f(x)$ be the density function of exponential random variable and X be the exponential random variable.

$X = f^{-1}(U)$ whare U is a uniform random variable.

1.1 Source code of solution in C++

```
int arr[100010];
double uniform[100010];
double exponential[100010];

void genUni(ll length,ll seed=7)
{
        arr[0]=seed;

        ll a=40692;
        ll m=2147483399;

        ll q=m/a;
        ll r=m%a;

        for(int i=1;i<=length;i++)
        {
                ll x = a*(arr[i-1])%m - (arr[i-1]/q)*r;
                if(x<0)
                {
                        x=x+m;
                }
                arr[i]= x;
                uniform[i]=x*1.0/m;
        }
}

double fInv(double x)
{
        return (-5*log(1-x));
}

void genExpo(int length)
{
        for(int i=1;i<=length;i++)
        {
                exponential[i]=fInv(uniform[i]);
        }
}

double calMean(int length)
{
        double total=0;
        for(int i=1;i<=length;++i)
        {
                total+=exponential[i];
```

```
        }
        return total*1.0/length;
}

double getMax(int length)
{
        double maxito=0;
        for(int i=1;i<=length;++i)
        {
                maxito=max(maxito,exponential[i]);
        }
        return maxito;
}


double getMin(int length)
{
        double minito=0;
        for(int i=1;i<=length;++i)
        {
                minito=min(minito,exponential[i]);
        }
        return minito;
}

double calVar(double mean,int length)
{
        double total=0;
        for(int i=0;i<=length;++i)
        {
                total+= (exponential[i]-mean)*(exponential[i]-mean);
        }
        return total/length;
}

int main()
{
        ll n=5000;
        genUni(n);
        genExpo(n);

        double mean=calMean(n);
        double var=calVar(mean,n);

        cout<<"Mean: "<<mean<<endl;
        cout<<"Var: "<<var<<endl;

        cout<<"Max: "<<getMax(n)<<endl;
        cout<<"Minimum: "<<getMin(n)<<endl;
}
```

1.2 Source code of solution in R

```
fInv<- function(x)
{
        return(-5*log(1-x))
```

```r
}

n=5000
sample<-vector(length=n)

cat("Sample vales are:")
for (i in 1:5000)
{
		u<-runif(1)
		sample[i]=fInv(u)
		cat(i," -> ",sample[i],"\n")
}

png("que1_in_R.png")
hist(sample,breaks=50,col="red",plot=TRUE)

cat("\n")
cat("\nMean: ",mean(sample),"\n")
cat("Variance: ",var(sample),"\n")
cat("Max: ",max(sample),"\n")
cat("Min: ",min(sample),"\n")
```

1.3 Observation

Here, we observe that the histogram of the generated sample is quite similar to the density function of the exponential distribution.

1.4 Histograms for the Generated Distribution

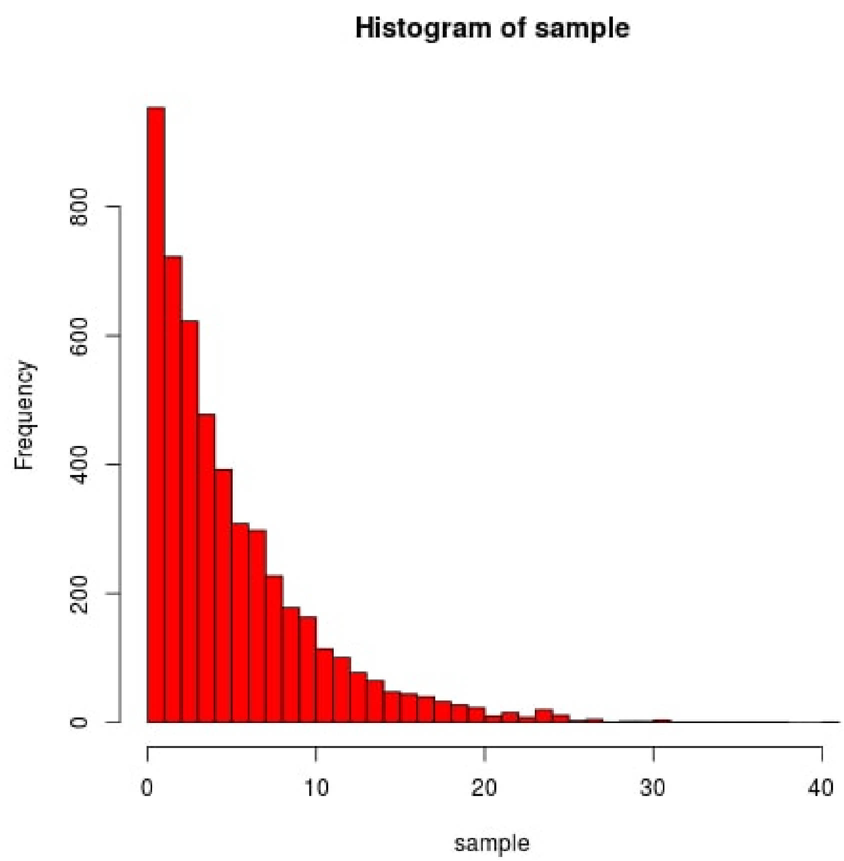

Abbildung 1: Histogram for generated exponential distribution.

2 Problem 2

Now, we have to generate random values from Gamma Random variable. As gamma random variable is sum of α independent exponential random variables of parameter λ, for *Gamma* distribution of parameters (α, λ).

Let, G be gamma random variable and X_i be the i^{th} exponential random variable.

$G = X_1 + X_2 + X_3 + X_4 + X_5$

2.1 Source code for solution in C++

```
int arr[100010];
double uniform[100010];
double gamma_dist[100010];

void genUni(ll length, ll seed=7)
{
        arr[0]=seed;

        ll a=40692;
        ll m=2147483399;

        ll q=m/a;
        ll r=m%a;

        for(int i=1;i<=length;i++)
        {
                ll x = a*(arr[i-1])%m - (arr[i-1]/q)*r;
                if(x<0)
                {
                        x=x+m;
                }
                arr[i]= x;
                uniform[i]=x*1.0/m;
        }
}

double fInv(double x)
{
        return (-(1/5.0)*log(1-x));
}

void genGamma(int length)
{
        for(int i=1;i<=length;i++)
        {
                gamma_dist[i]=(fInv(uniform[i])
                        +fInv(uniform[i+5000])+fInv(uniform[i+10000])
                        +fInv(uniform[i+15000])+fInv(uniform[i+20000]));
        }
}
```

```
double calMean(int length)
{
        double total=0;
        for(int i=1;i<=length;++i)
        {
                total+=gamma_dist[i];
        }
        return total*1.0/length;
}

double getMax(int length)
{
        double maxito=0;
        for(int i=1;i<=length;++i)
        {
                maxito=max(maxito,gamma_dist[i]);
        }
        return maxito;
}


double getMin(int length)
{
        double minito=0;
        for(int i=1;i<=length;++i)
        {
                minito=min(minito,gamma_dist[i]);
        }
        return minito;
}

double calVar(double mean,int length)
{
        double total=0;
        for(int i=0;i<=length;++i)
        {
                total+= (gamma_dist[i]-mean)*(gamma_dist[i]-mean);
        }
        return total/length;
}

int main()
{
        ll n=5000;
        genUni(n*5);
        genGamma(n);

        double mean=calMean(n);
        double var=calVar(mean,n);

        cout<<"Mean: "<<mean<<endl;
        cout<<"Var: "<<var<<endl;

        cout<<"Max: "<<getMax(n)<<endl;
        cout<<"Minimum: "<<getMin(n)<<endl;

}
```

2.2 Source code of solution in R

```
fInv<- function(x)
{
        return(-(1/5.0)*log(1-x))
}

n=5000
sample<-vector(length=n)

cat("Sample vales are:")
for (i in 1:5000)
{
        u<-runif(5,0,1)
        sample[i]=sum(fInv(u))
        cat(i," -> ",sample[i],"\n")
}

png("que2_in_R.png")
hist(sample,breaks=50,col="red",plot=TRUE)

cat("\n")
cat("\nMean: ",mean(sample),"\n")
cat("Variance: ",var(sample),"\n")
cat("Max: ",max(sample),"\n")
cat("Min: ",min(sample),"\n")
```

2.3 Observation

Here, we observe that the histogram of the generated sample is quite similar to the density function of the Gamma distribution.

2.4 Histograms for the Generated Distribution

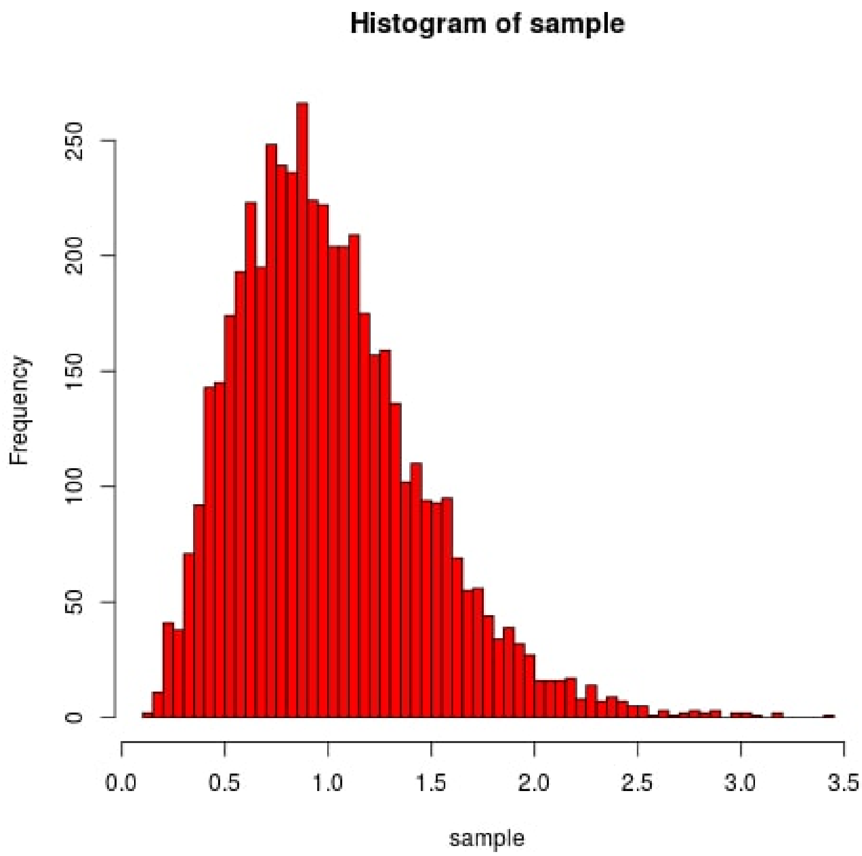

Abbildung 2: Histogram for generated Gamma distribution.

3 Problem 3

In this we have to use Acceptance-Rejection method to generate samples of a probability distribution given by $f(x)$.

$f(x) = 20x(1-x)^3$

Here, we take $g(x) = 1$, that is, and $c = 2.5$. This satisfies the following condition.

$f(x) < c * g(x) \; \forall \; x \in [0, 1]$

3.1 Source code for solution in C++

```
int arr[100010];
double uniform[1000010];
double sample[1000010];

void genUni(ll length, ll seed=7)
{
        arr[0]=seed;

        ll a=40692;
        ll m=2147483399;

        ll q=m/a;
        ll r=m%a;

        for(int i=1;i<=length;i++)
        {
                ll x = a*(arr[i-1])%m - (arr[i-1]/q)*r;
                if(x<0)
                {
                        x=x+m;
                }
                arr[i]= x;
                uniform[i]=x*1.0/m;
        }
}

double f(double x)
{
        return 20*x*(1-x)*(1-x)*(1-x);
}

void acceptReject(ll n)
{
        int i=0,index=1;
        while(index<=n)
        {
                double u1=uniform[i];
                double u2=uniform[i+50000];
```

```
                double c=2.5;

                if(c*u2 < f(u1))
                {
                        sample[index++]=u1;
                }
                i++;
        }
}

double getMax(int length)
{
        double maxito=0;
        for(int i=1;i<=length;++i)
        {
                maxito=max(maxito,sample[i]);
        }
        return maxito;
}


double getMin(int length)
{
        double minito=0;
        for(int i=1;i<=length;++i)
        {
                minito=min(minito,sample[i]);
        }
        return minito;
}

double calVar(double mean,int length)
{
        double total=0;
        for(int i=0;i<=length;++i)
        {
                total+= (sample[i]-mean)*(sample[i]-mean);
        }
        return total/length;
}

double calMean(int length)
{
        double total=0;
        for(int i=1;i<=length;++i)
        {
                total+=sample[i];
        }
        return total*1.0/length;
}


int main()
{
        ll n=5000;
        genUni(n*200);
        acceptReject(n);
```

```
        double mean=calMean(n);
        double var=calVar(mean,n);

        cout<<"Mean: "<<mean<<endl;
        cout<<"Var: "<<var<<endl;

        cout<<"Max: "<<getMax(n)<<endl;
        cout<<"Minimum: "<<getMin(n)<<endl;
}
```

3.2 Source code of solution in R

```
f<-function(x)
{
        return(20*x*(1-x)*(1-x)*(1-x))
}

n=10000
sample<- vector(length=n)

index<-1
c<-2.5

while (index<n)
{
        u1<-runif(1)
        u2<-runif(1)

        if(c*u2 < f(u1))
        {
                sample[index]=u1
                index<-index+1
        }
}

png("que3_in_R.png")
hist(sample,breaks=50,col="red",plot=TRUE)

cat("Mean: ",mean(sample),"\n")
cat("Variance: ",var(sample),"\n")
cat("Max: ",max(sample),"\n")
cat("Min: ",min(sample),"\n")
```

3.3 Histograms for the Generated Distribution

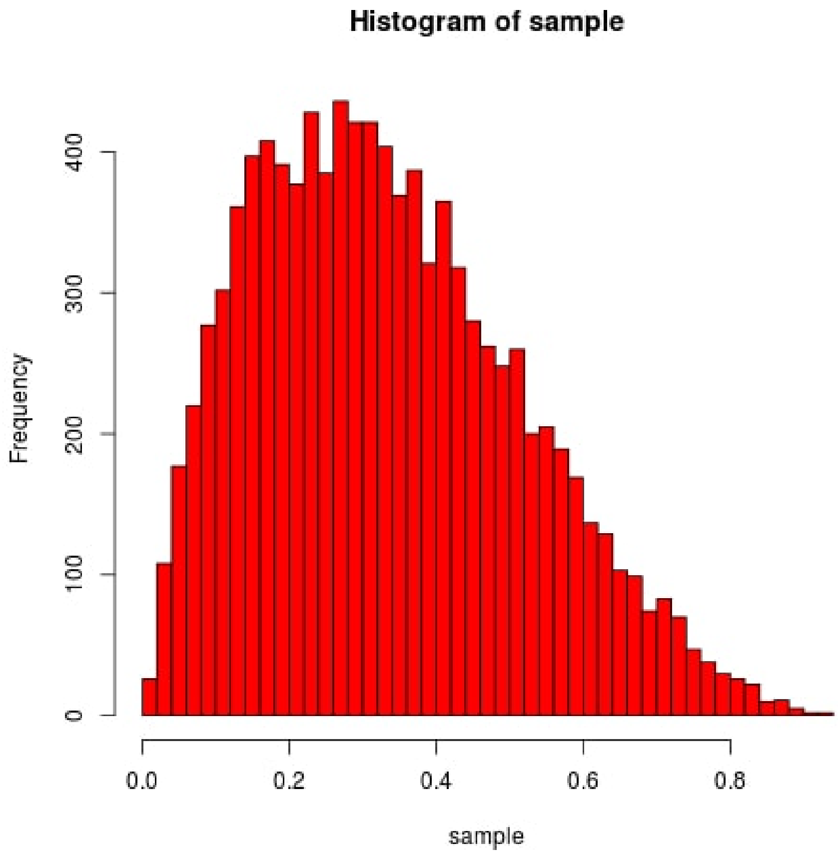

Abbildung 3: Histogram for generated distribution.