STA 3431 Assignment #1

Yihan Duan

08/10/2021

Name: Yihan Duan Student #: 1003118547 Department: CS Program: MScAC Year: first year master
E-mail: yihan.duan@mail.utoronto.ca

Question 1

In this question, we choose another set of parameters for LCG. As discussed in class, by the Hull–Dobell Theorem, we want to make sure:

1. gcd(b, m) = 1
2. every "prime or 4" divisor of m also divides a - 1.

```
# define random function
m = 2^32
a = 4 * 69069 + 1
b = 23606797 * 7
latestval = 12345
nextrand = function() {
    latestval <<- (a * latestval + b)%%m
    return(latestval/m)
}

# record 100000 random variables
n = 1e+06
rand_vals = c(n)
for (i in 1:n) {
    rand_vals[i] = nextrand()
}
```

As we choose m to be 2^32, we only need b to be odd and a - 1 to be a multiple of 4. To avoid similarities between Un and Un-1, we choose a relatively large a.

Let's see the statistics compared to theoretical values of Uniform(0, 1).

```
# compare statistics with theoretical values
limits= c(100, 1000, 10000, 100000)
for (lmt in limits) {
  cat("For the first ", lmt, "observations:\n")
  cat("Real mean: ", mean(rand_vals[1:lmt]), ", should be: ", 0.5, '\n')
  cat("Real standard deviation: ", sd(rand_vals[1:lmt]), ", should be: ", sqrt(1/12), '\n')
}
```

```
## For the first  100 observations:
## Real mean:  0.4692917 , should be:  0.5
## Real standard deviation:  0.3109622 , should be:  0.2886751
## For the first  1000 observations:
```

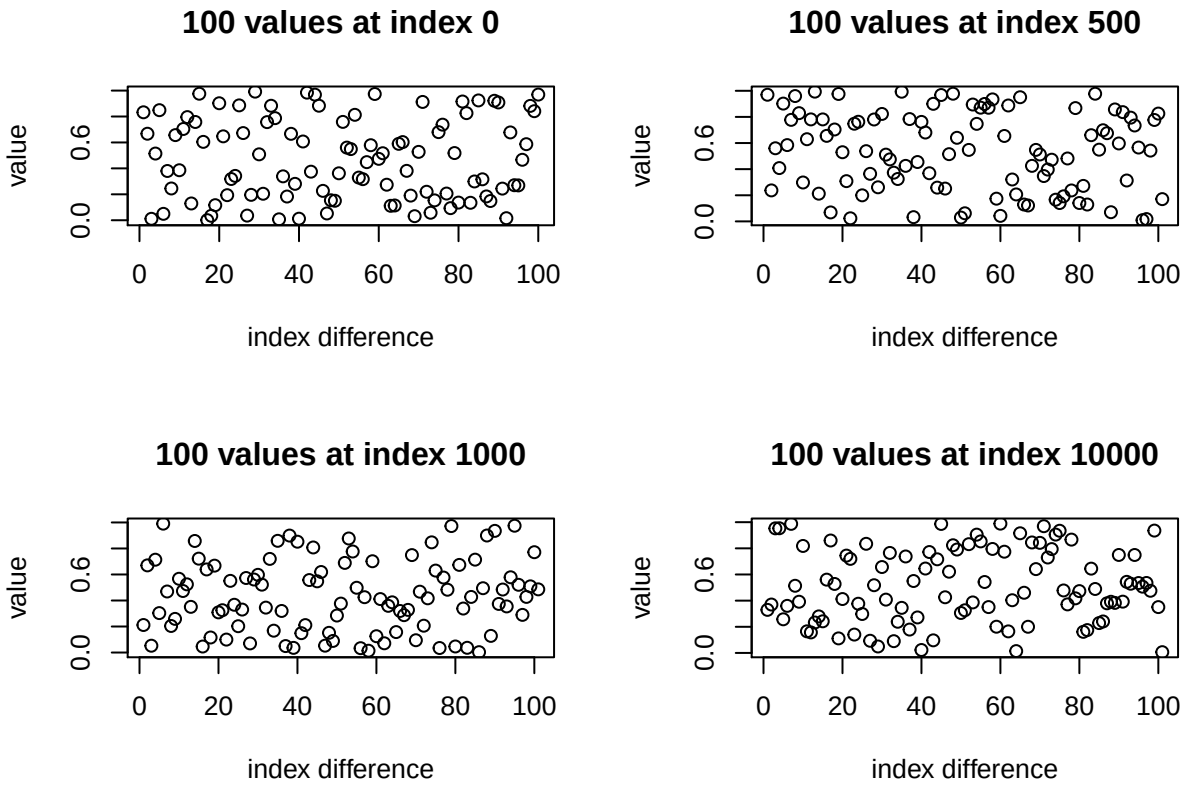
```
## Real mean:  0.5035736 , should be:  0.5
## Real standard deviation:  0.2943177 , should be:  0.2886751
## For the first  10000 observations:
## Real mean:  0.4992396 , should be:  0.5
## Real standard deviation:  0.2905168 , should be:  0.2886751
## For the first  1e+05 observations:
## Real mean:  0.5003146 , should be:  0.5
## Real standard deviation:  0.2885985 , should be:  0.2886751
```

As the simulation size grow, the mean and standard deviation of the generated samples get very close to the theoretical mean and standard deviation of a Uniform(0,1) distribution.

Randomness

We first try to plot 100 consecutive values at different indices and see if they appear random.

```r
# compare statistics with theoretical values
par(mfrow=c(2,2))
limits= c(0, 500, 1000, 10000)
for (lmt in limits) {
  plot(rand_vals[lmt:(lmt+100)],
       xlab='index difference',
       ylab='value',
       main=paste('100 values at index', lmt))
}
```



Yes,the values seems to be randomly scattered in all the plots.

If the distribution is random, then the time it takes to reach a duplicate (very close) value should have high variance. Although in theory the same number should never appear twice. To verify that, we first check out if duplicates exists in the numbers we generated.

```r
# see when the RNG gets back to exactly the same value
get_next_index_with_same_value <- function(x, pivot_idx) {
  pivot_val = x[pivot_idx]
  for (i in 1:n) {
    val = x[i]
    if (val == pivot_val && i != pivot_idx) {
      cat("Value at index ", i, " and ", pivot_idx, " are the same.\n")
      break
    }
  }
  cat("No duplicates for value ", pivot_val, " in 1000000 generated numbers.\n")
}

for (i in 1:10) {
  get_next_index_with_same_value(x=rand_vals, pivot_idx=i)
}
```

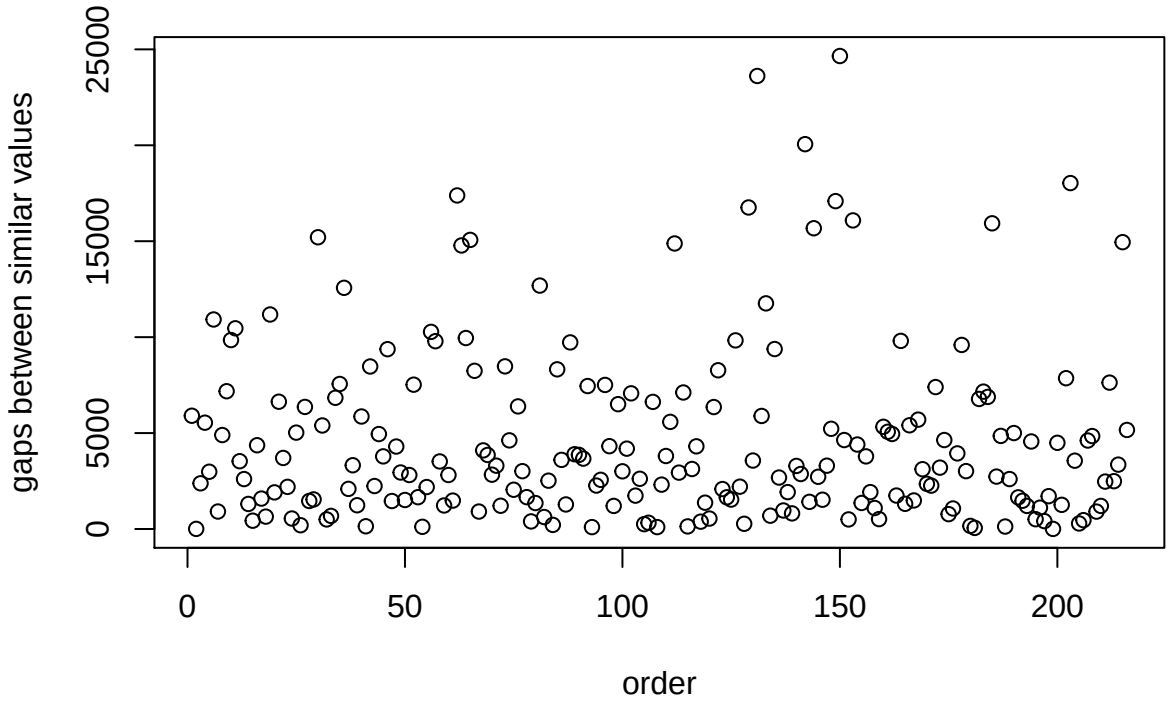
```
## No duplicates for value  0.8325761  in 1000000 generated numbers.
## No duplicates for value  0.6671383  in 1000000 generated numbers.
## No duplicates for value  0.01112462  in 1000000 generated numbers.
## No duplicates for value  0.514273  in 1000000 generated numbers.
## No duplicates for value  0.8488944  in 1000000 generated numbers.
## No duplicates for value  0.0493756  in 1000000 generated numbers.
## No duplicates for value  0.3798152  in 1000000 generated numbers.
## No duplicates for value  0.2450244  in 1000000 generated numbers.
## No duplicates for value  0.6564873  in 1000000 generated numbers.
## No duplicates for value  0.3856769  in 1000000 generated numbers.
```

As we expected, there are no duplicates for the first 10^6 values generated by RNG. Let's try to find the gaps between close values (difference <= 0.0001).

```r
# Find the gap in index between two close values
get_index_gaps_between_close_values <- function(x, pivot_idx, margin) {
  pivot_val = x[pivot_idx]
  indices = c()
  for (i in 1:n) {
    val = x[i]
    if (abs(val - pivot_val) <= margin) {
      indices = c(indices, i)
    }
  }
  gaps = c()
  for (i in 1:(length(indices)-1)) {
    gaps = c(gaps, indices[i+1] - indices[i])
  }

  return(gaps)
}
gaps_for_idx_1 = get_index_gaps_between_close_values(x=rand_vals, pivot_idx=1, margin=0.0001)

# plot the gaps
plot(gaps_for_idx_1,
     ylab="gaps between similar values",
     xlab='order')
```




```
cat("Variance for gaps: ", var(gaps_for_idx_1))
```

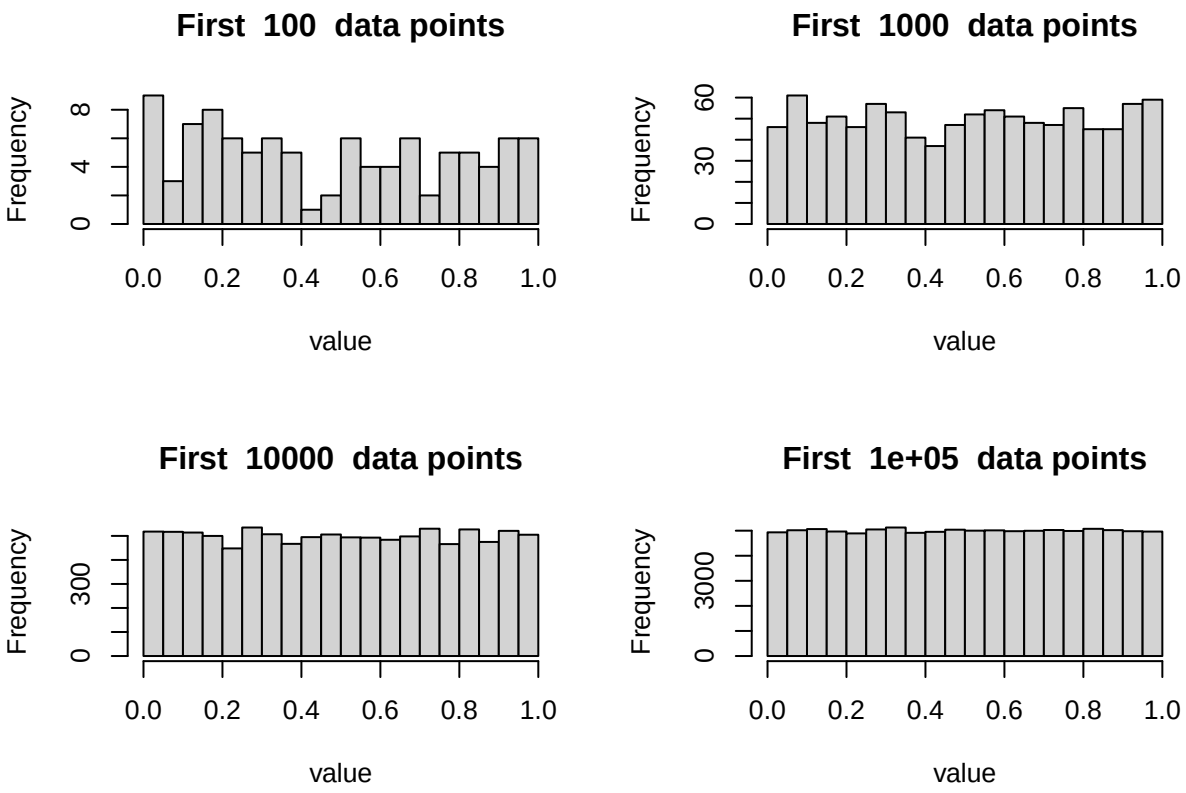
```
## Variance for gaps:  21029295
```

As expected, the time it takes for close values to appear have high variance. Therefore, I believe the RNG has high randomness.

Uniformity

Let's see the distribution of the numbers generated by this algorithm.

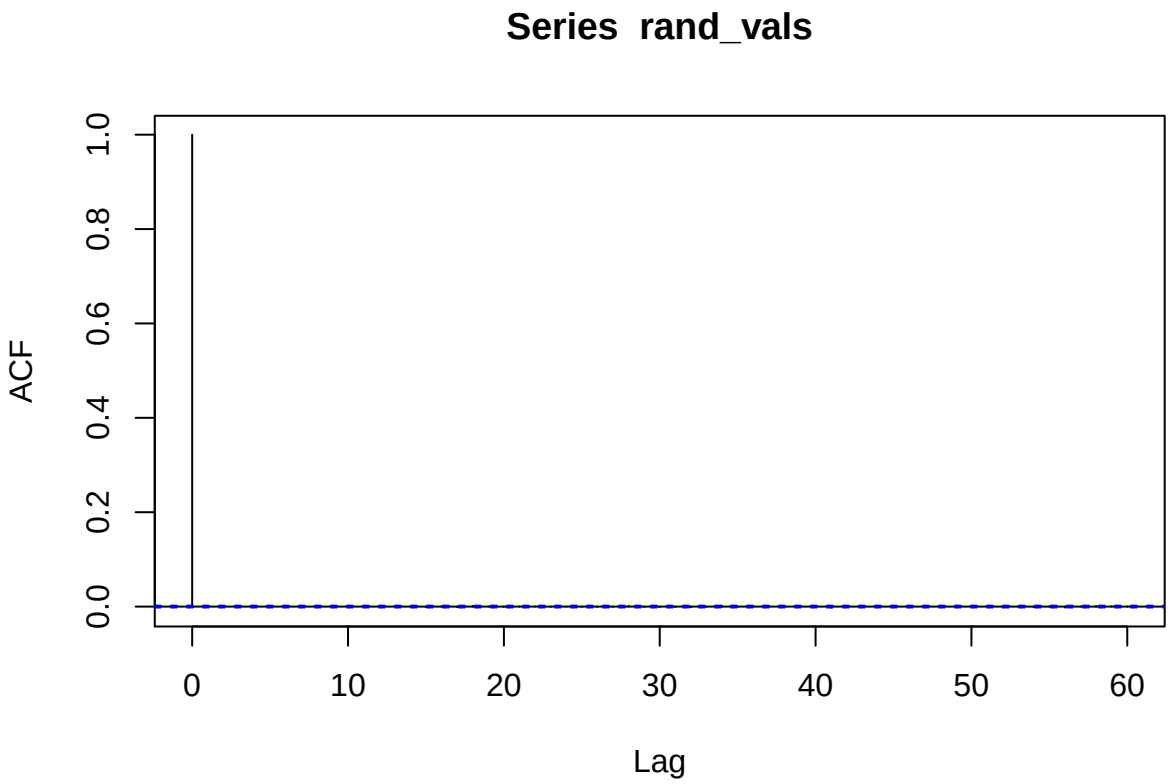
```
# plot frequency of the first N variables.
par(mfrow = c(2, 2))
limits = c(100, 1000, 10000, 1e+05)
for (lmt in limits) {
    hist(rand_vals[1:lmt], main = paste("First ", lmt, " data points"), xlab = "value",
        breaks = 15)
}
```



From the histograms alone, we can see that, as the number of observations increases, the values are evenly (uniformly) distributed between 0 and 1.

Independence

```
acf(rand_vals)
```



Using auto correlation function, we can see that there is no correlation between values generated close to each other. Thus the values are independent.

Question 2

As discussed in class, we can simulate exponential distribution and normal distribution using uniform distribution.

```
# simulation size
n = 1000

# function for acquiring n Uniform random numbers
get_randvals <- function(n) {
  rand_vals = c(n)
  for (i in 1:n) {
    rand_vals[i] = nextrand()
  }
  return(rand_vals)
}

# function for estimating required expectation
get_exp <- function() {
  # lambda = 3
  U1 = get_randvals(n)
  Y= -log(U1) / 3

  # use Box-Muller transformation
  U2 = get_randvals(n)
  U3 = get_randvals(n)
  Z = sqrt(2 * log(1/U2)) * cos(2 * 3.14159265 * U3)
```

```
  # get vars
  X = abs((Y^2) * (Z^5) * sin((Y^3) * (Z^2)))

  # compute and output the mean and standard error
  m = mean(X)
  se = sd(X) / sqrt(n)
  cat("MC:  ", m, " +- ", se, "  (n=", n, ")", "\n", sep='')
  cat("  95% C.I.:  (", m-1.96*se, ",", m+1.96*se, ")\n", sep='')
}

n = 1000000
get_exp()
```

```
## MC:  0.8465307 +- 0.01100011  (n=1e+06)
##   95% C.I.:  (0.8249705,0.8680909)
```

```
get_exp()
```

```
## MC:  0.8255937 +- 0.01109805  (n=1e+06)
##   95% C.I.:  (0.8038416,0.8473459)
```

```
get_exp()
```

```
## MC:  0.8367128 +- 0.01068597  (n=1e+06)
##   95% C.I.:  (0.8157683,0.8576573)
```

We can see that the variance is still pretty high. Try increasing the simulation size to see if it gets better.

```
# Incrase n to reduce standard error
n = 10000000
get_exp()
```

```
## MC:  0.8422648 +- 0.00378406  (n=1e+07)
##   95% C.I.:  (0.834848,0.8496815)
```

```
get_exp()
```

```
## MC:  0.8319338 +- 0.003634367  (n=1e+07)
##   95% C.I.:  (0.8248105,0.8390572)
```

```
get_exp()
```

```
## MC:  0.8356876 +- 0.003617307  (n=1e+07)
##   95% C.I.:  (0.8285977,0.8427775)
```

The estimated expectation value is 0.83 with the approximate 95% confidence interval being about (0.82, 0.84). This estimation is not very accurate because:

1. the estimated standard error is relatively high (~ 0.0036).
2. the result of "classical" Monte Carlo method still fluctuates a lot between different runs.

This is because the term $|Y^2Z^5|$ has high variance when $Y \sim$ Exponential(3) and $Z \sim$ Normal$(0, 1)$.

Question 3

My student number is 1003118547. Using the last 4 digits, we have:

$$A = 8, B = 5, C = 4, D = 7$$

Then the function is defined by:

$$g(x_1, x_2, x_3, x_4, x_5) = x_1^{14} 2^{x_2+3} (1 + \cos[x_1 + 2x_2 + 3x_3 + 4x_4 + 8x_5]) e^{-8x_4^2} e^{-9(x_4 - 3x_5)^2} \prod_{i=1}^{5} 1_{0<x_i<1}$$

We can compute the expected value by:

$$I = \frac{\sum(h(x_i)g(x_i)/f(x_i))}{\sum(g(x_i)/f(x_i))}$$

Let:

$$f(x_1, x_2, x_3, x_4, x_5) = f_1(x_1) f_2(x_2) f_3(x_3) f_4(x_4) f_5(x_5)$$

where f_i's are all density function for single variables. We define:

$$f_1(y) = 15y^{14}$$

Then as discussed in class, if $Y = U^{1/15}$ and $U \sim \text{Uniform}(0, 1)$, Y has density function f_1.

We define:

$$f_2(y) = ln(2)2^y = \frac{ln(2)}{8} 2^{y+3}$$

This is a density function for $y \in (0, 1)$ because:

$$\int_0^1 ln(2)2^x dx = 2^x|_0^1 = 1$$

The corresponding CDF and its inverse is defined by:

$$F(x) = 2^x - 1(x \in (0, 1))$$
$$F^{-1}(t) = log_2(t + 1)$$

Using inverse CDF method, we can sample x_2 by:

$$X_2 = log_2(U + 1)$$

For the subsequent variables, we assume they all follows simple Uniform(0,1) distribution.

$$f_i(y) = 1(y \in (0, 1), i = 3, 4, 5)$$

Thus,

$$f(x_1, x_2, x_3, x_4, x_5) = 15x_1^{14} \frac{ln(2)}{8} 2^{x_2+3}$$

We choose this density function to cancel the first part of $g(X)$ function. By doing this, when the final expectation is calculated, the variance of the denominator (also of the numerator) is reduced. However, I didn't figure out the terms in g(X) that includes x_3, x_4, x_5. It will be ideal if we can cancel some of them out as well.

```r
n = 10^6

get_expected <- function(n=10^6) {
  x1list = runif(n)^(1/15)
  x2list = log2(runif(n)+1)
  x3list = runif(n)
  x4list = runif(n)
  x5list = runif(n)

  numlist = ((x1list + x2list^2) / (2 + x3list*x4list + x5list)) * (1 + cos(x1list + 2*x2list + 3*x3lis
  denomlist = (1 + cos(x1list + 2*x2list + 3*x3list + 4*x4list + 8*x5list)) * exp(-8*x4list^2) * exp(-9

  return(sum(numlist)/sum(denomlist))
}

estimates = c()
for (i in 1:20) {
  estimates = c(estimates, get_expected())
}

cat("Mutiple estimates: \n")
```

```
## Mutiple estimates:
```

```r
cat("First 10 estimates: ", estimates[1:10], "\n")
```

```
## First 10 estimates:  0.6173022 0.6172622 0.6179574 0.6183603 0.6170978 0.6177125 0.617852 0.6175008 (
```

```r
cat("Mean of 20 estiamtes: ", mean(estimates), "\n")
```

```
## Mean of 20 estiamtes:  0.6175886
```

```r
cat("Standard deviation of 20 estiamtes: ", sd(estimates), "\n")
```

```
## Standard deviation of 20 estiamtes:  0.0004511299
```

The final estimation for the expected value is 0.617. I believe that this estimation is moderately accurate. The first 10 independent estimates are pretty close to each other (most of which are within range (0.617, 0.618)) and the standard deviation of the estimates is small. This is because the first two terms containing x_1, x_2 are cancelled out.If they are not cancelled out, will increase the variance of the denominator and lead to inaccurate estimates. For this particular problem, importance sampler works pretty well.

Question 4

We choose f by:

$$f(x_1, x_2, x_3, x_4, x_5) = f(x_2) = \frac{2}{3}(x_2 + 1) \ (0 \leq x_2 \leq 1)$$

This is a valid density function because:

$$\int_0^1 f(y)dy = \int_0^1 \frac{2}{3}(y+1)dy = (\frac{1}{3}y^2 + \frac{2}{3}y)|_0^1 = 1$$

We will then prove that for $K = 24$, $Kf(X) \geq g(X)$

First consider $g(X)$ only in the region $0 < x_i < 1$ for i = 1, 2, 3, 4, 5.

$$g(X) = x_1^{14} 2^{x_2+3} (1 + \cos[x_1 + 2x_2 + 3x_3 + 4x_4 + 8x_5]) e^{-8x_4^2} e^{-9(x_4-3x_5)^2}$$

We can think of the density function as undefined anywhere else. We have

$$0 \leq x_1^{14} \leq 1,$$

$$-1 \leq cos(x_1 + 2x_2 + 3x_3 + 4x_4 + 8x_5) \leq 1 \Rightarrow 0 \leq 1 + \cos(x_1 + 2x_2 + 3x_3 + 4x_4 + 8x_5) \leq 2,$$

$$-8x_4^2 \leq 0 \Rightarrow 0 \leq e^{-8x_4^2} \leq 1,$$

$$-9(x_4 - 3x_5)^2 \leq 0 \Rightarrow 0 \leq e^{-9(x_4-3x_5)^2} \leq 1$$

Therefore,

$$g(X) \leq 1 * 2^{x_2+3} * 2 * 1 * 1 = 2^{x_2+4}$$

We try to prove:

$$g(X) \leq Kf(X)$$

$$\Leftrightarrow 2^{t+4} \leq 24 * \frac{2}{3} * (t + 1)$$

$$\Leftrightarrow 16 * 2^t \leq 16 * (t + 1)$$

$$\Leftrightarrow 2^t \leq t + 1 \ (0 \leq t \leq 1)$$

To prove this inequality, consider the function:

$$h(x) = x + 1 - 2^x$$

We know that $h(x) = 0$ when $x = 0$ or $x = 1$ and that h is a continuous function on (0,1). To prove the inequality, we get the second derivative:

$$h''(x) = (1 - ln(2)2^x)' = -ln^2(2)2^x$$

Notice the second derivative is less than 0 on (0,1), g is a concave function on (0, 1). We have $g(x) > 0, x \in (0, 1)$. The above inequality holds.

Similarly, we use the inverse CDF method to sample form density function f(x). Let:

$$F(x) = \frac{1}{3}y^2 + \frac{2}{3}y = t$$

Then,

$$3t + 1 = y^2 + 2y + 1 = (y + 1)^2$$

$$F^{-1}(t) = \sqrt{3t + 1} - 1$$

To achieve this particular density function, we can sample from X where:

$$X = \sqrt{3U + 1} - 1, U \sim Uniform(0, 1)$$

```
g <- function(x1, x2, x3, x4, x5) {
  return(x1^14 * 2^(x2+3) * (1 + cos(x1 + 2*x2 + 3*x3 + 4*x4 + 8*x5)) * exp(-8*x4^2) * exp(-9*(x4 - 3*x5
}

h <- function(x1, x2, x3, x4, x5) {
  return((x1 + x2^2) / (2 + x3*x4 + x5))
}

f <- function(x) {
  return((2/3) * (x + 1))
}

K <- 24

# Sampling function
get_sample <- function() {
  return(sqrt(3 * runif(1) + 1) - 1)
}

num_atmpts = 10^6
hlist = c()
num_smpls = 0

for (i in 1:num_atmpts) {
  X1 = runif(1)
  X2 = get_sample()
  X3 = runif(1)
  X4 = runif(1)
  X5 = runif(1)
  U = runif(1)
  alpha = g(X1, X2, X3, X4, X5) / (K * f(X2))
  if (U < alpha) {
    hlist = c(hlist, h(X1, X2, X3, X4, X5))
    num_smpls = num_smpls + 1
  }
}

cat("Out of", num_atmpts, "attempts, obtained", num_smpls, "samples\n")
```

```
## Out of 1e+06 attempts, obtained 1710 samples
```

```
cat("mean of h(X) is about", mean(hlist, na.rm=TRUE), "\n")
```

```
## mean of h(X) is about 0.61149
```

```r
se =  sd(hlist, na.rm=TRUE) / sqrt(num_smpls)
cat("standard error of h(X) is about", se, "\n")
```

```
## standard error of h(X) is about 0.003577743
```

The estimated h(X) is relatively close to the estimation in question 3, but not as accurate. This is because the rate of accepting samples is low and $Kf(X)$ is 'far away' from $g(X)$ (much larger than $g(X)$). In other words, the inequality was too 'loose'. Rejection sampler still works, but this particular $f(X)$ is not ideal.

I just realized that we can further reduce the distance by using the same f as question 3 and use $K = \frac{16}{15ln(2)}$. Proof is mostly the same except we don't apply inequality to the first two terms. Let's just try that.

```r
g <- function(x1, x2, x3, x4, x5) {
  return(x1^14 * 2^(x2+3) * (1 + cos(x1 + 2*x2 + 3*x3 + 4*x4 + 8*x5)) * exp(-8*x4^2) * exp(-9*(x4 - 3*x5
}

h <- function(x1, x2, x3, x4, x5) {
  return((x1 + x2^2) / (2 + x3*x4 + x5))
}

f <- function(x1, x2) {
  return(15*x1^14 *(log(2)/8)* 2^(x2+3))
}

K <- 16/(15*log(2))

num_atmpts = 10^6
hlist = c()
num_smpls = 0

for (i in 1:num_atmpts) {
  X1 = runif(1)^(1/15)
  X2 = log2(runif(1)+1)
  X3 = runif(1)
  X4 = runif(1)
  X5 = runif(1)
  U = runif(1)
  alpha = g(X1, X2, X3, X4, X5) / (K * f(X1, X2))
  if (U < alpha) {
    hlist = c(hlist, h(X1, X2, X3, X4, X5))
    num_smpls = num_smpls + 1
  }
}

cat("Out of", num_atmpts, "attempts, obtained", num_smpls, "samples\n")
```

```
## Out of 1e+06 attempts, obtained 26544 samples
```

```r
cat("mean of h(X) is about", mean(hlist, na.rm=TRUE), "\n")
```

```
## mean of h(X) is about 0.6172286
```

```r
se =  sd(hlist, na.rm=TRUE) / sqrt(num_smpls)
cat("standard error of h(X) is about", se, "\n")
```

```
## standard error of h(X) is about 0.0009191056
```

Now the estimate is more accurate with lower standard error. Although not taking into consideration the last few terms, the acceptance rate is much higher. This algorithm works well.