Stats 102C, Homework 2

Charles Liu (304804942)

Modify this file with your answers and responses.

Academic Integrity Statement

By including this statement, I, **Charles Liu**, declare that all of the work in this assignment is my own original work. At no time did I look at the code of other students nor did I search for code solutions online. I understand that plagiarism on any single part of this assignment will result in a 0 for the entire assignment and that I will be referred to the dean of students.

I did discuss ideas related to the homework with Josephine Bruin for parts 2 and 3, with John Wooden for part 2, and with Gene Block for part 5. At no point did I show another student my code, nor did I look at another student's code.

Reading

Reading is important!

- Chapter 2 section 2 in Introducing Monte Carlo Methods with R
- Chapter 3 sections 1-3 in Introducing Monte Carlo Methods with R

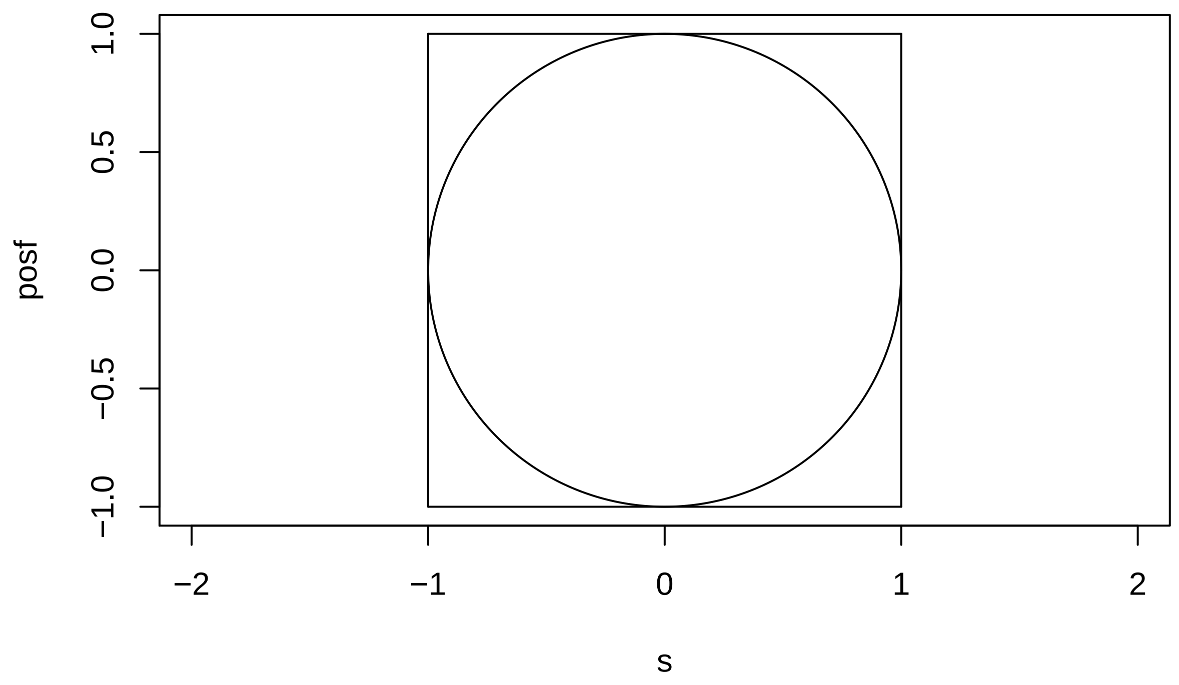
Problem 1 - Estimate pi (poorly)

A simple Monte Carlo Exercise . . . Get an estimate of pi by using random uniform numbers. (It won't be a very good estimate.)

In this first exercise, we can see how a simple source of randomness (in our case, R's `runif()` function) can be used to estimate tough quantities.

We will find an estimate of pi by estimating the ratio between the area of a circle and its encompassing square.

```r
s <- seq(-1, 1, by = 0.001)
posf <- sqrt(1 - s ^ 2)
plot(s, posf, type = "l", asp = 1, ylim = c(-1, 1))
lines(s, -1 * posf)
segments(-1, -1, -1, 1)
segments(-1, -1, 1, -1)
segments(1, 1, -1, 1)
segments(1, 1, 1, -1)
```



To calculate the area of the circle analytically, we would need to integrate the function drawing the upper semi-circle and then multiply that by 2. This process requires the use of trig substitutions, and while doable, can illustrate a time where the analytic solution is not easy.

$$Area = 2 \times \int_{-1}^{1} \sqrt{1 - x^2} dx$$

For the Monte-Carlo approach, we will use `runif(n, min = -1, max=1)` to generate a bunch of random pairs of x and y coordinates. We will see how many of those random uniform points fall within the circle. This is easy - just see if $x^2 + y^2 \leq 1$. The total area of the square is 4. The total area of the circle is pi. Thus, the proportion of coordinates that satisfy the inequality $x^2 + y^2 \leq 1 \approx \pi/4$.

Instructions:

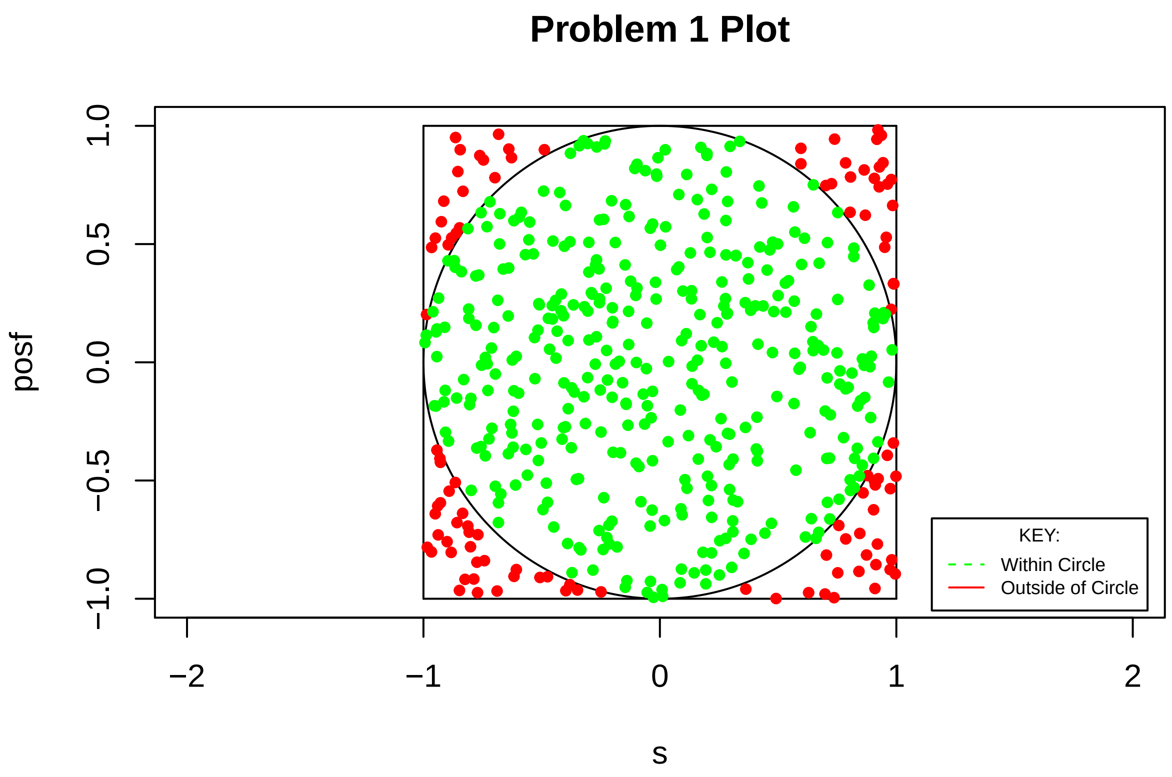
- create a vector x of n random values between -1 and 1. I suggest starting with n = 500
- create a vector y of n random values between -1 and 1. Use the two vectors to make coordinate pairs.
- calculate which of points satisfy the inequality for falling inside the circle.
- Print out your estimate of pi by multiplying the proportion by 4.
- plot each of those (x,y) coordinate pairs. Use pch = 20. Color the points based on whether they fall in the circle or not.

```r
# Create the x vector with n = 500
x <- runif(n = 500, min = -1, max = 1)

# Create the y vector with n = 500
y <- runif(n = 500, min = -1, max = 1)
```

```r
# Create our coordinate pairs
coord_pairs <- matrix(c(x, y), ncol = 2)

# Create our proportion of points within the circle
prop_circle <- ( ((coord_pairs[, 1]^2) + (coord_pairs[, 2]^2))  <= 1)

# Create our estimate for pi
mean_pi <- mean(prop_circle)
estimate_pi <- 4 * mean_pi
estimate_pi
```

```
## [1] 3.144
```

```r
# Plot our estimates
plot(s, posf, type = "l", asp = 1, ylim = c(-1, 1), main = "Problem 1 Plot")
lines(s, -1 * posf)
segments(-1, -1, -1, 1)
segments(-1, -1, 1, -1)
segments(1, 1, -1, 1)
segments(1, 1, 1, -1)
points(coord_pairs, pch = 20,
       col = ifelse(prop_circle >= (pi/4), "green", "red"))
legend(1.15, -0.66,
       legend = c("Within Circle", "Outside of Circle"),
       col = c("green", "red"),
       lty = c(2,1,1,1),
       cex = 0.6,
       title = "KEY:")
```



Monte Carlo Integration

Problem 2

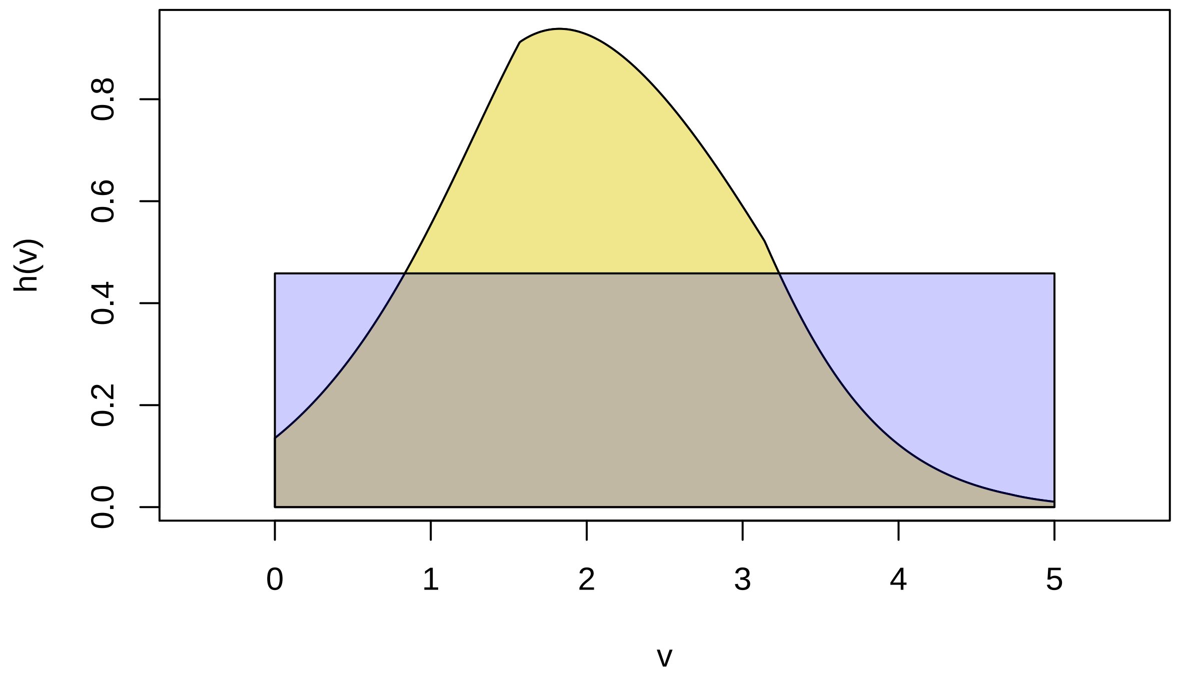
Estimate the integral of the following function by using Monte Carlo integration.

$$I = \int_0^5 h(x)dx = \int_0^5 \exp(-0.5(x-2)^2 - 0.1|\sin(2x)|)dx$$

We want to estimate the value of I, the area under the curve from 0 to 5. We can say that area is equal to the average value of $h(x)$ times the width:

$$I = 5 \cdot \mathbb{E}_f[h(X)]$$

```
# what the function looks like
h <- function(x) {exp(-0.5 * (x - 2) ^ 2 - 0.1 * abs( sin(2 * x) ) )}
v <- seq(0, 5, by = 0.01)
plot(v, h(v), type = "l", xlim = c(-0.5, 5.5))
polygon(c(0, v, 5), c(0, h(v), 0), col = 'khaki')
rect_ht = 0.458394 # height of rectangle is the Expected value of h(x)
polygon(c(0, 0, 5, 5), c(0, rect_ht, rect_ht, 0), col = rgb(0, 0, 1, .2))
```



For MC integration, we estimate

$$\mathbb{E}_f[h(X)] = \int_{\mathcal{X}} h(x)f(x)dx$$

For this first problem, we will use the uniform distribution on the interval (0,5) to draw samples of x. Thus, the PDF, $f(x)$ is

$$f(x) = 1/5$$

Thus,

$$I = 5 \cdot \mathbb{E}_f[h(X)] = 5 \cdot \int_0^5 h(x)f(x)dx \approx \frac{5}{N}\sum_{j=1}^{N} h(x_j) = \hat{I}$$

Where $x_j \sim \text{Unif}(0, 5)$

Generate a sample of 5000 values of x to estimate $\hat{I}$. Use `runif()` to generate the random uniform values.

Create a plot using the cumulative mean (aka running mean) of the first 500 values in the sample to show how the estimate of the integral 'settles' over the course of the sampling.

Do this two more times (each with a different starting seed), for a total of three samples of 5000 values each. Plot the cumulative mean of the first 500 values of each sample. There will be three different lines on this plot.

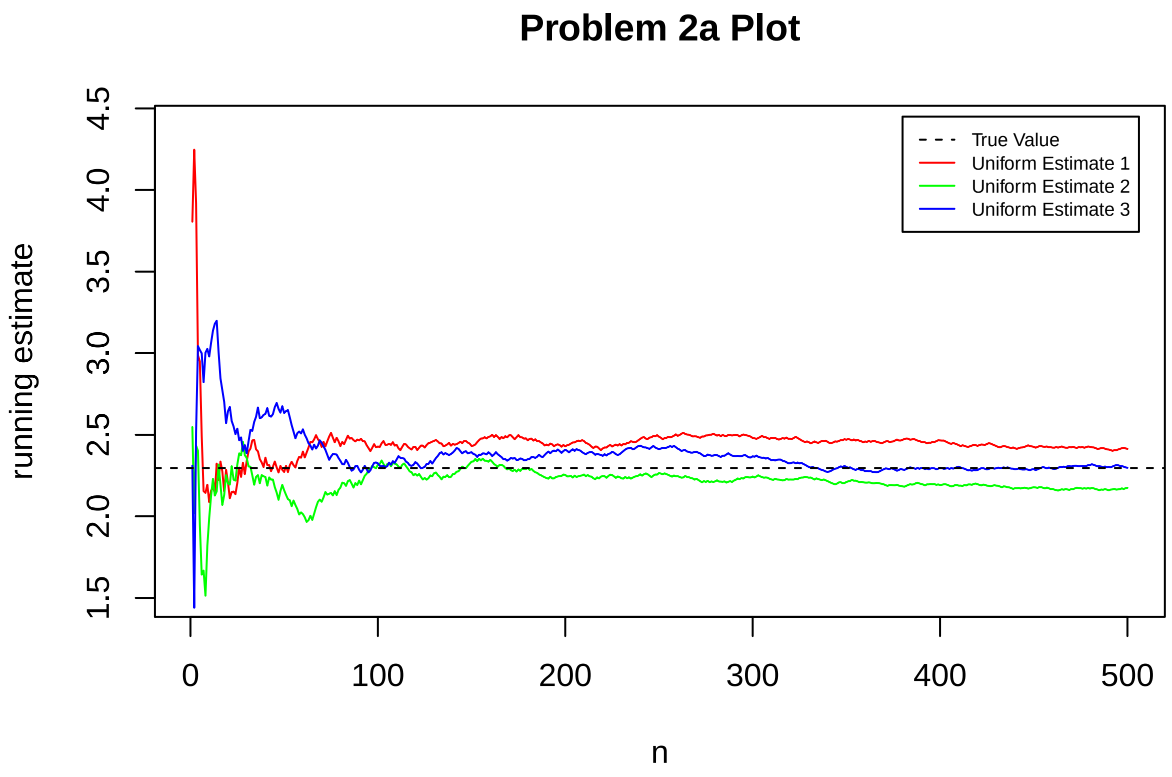
In your plot also include the 'true' value of the integral, 2.29583, as a horizontal line.

Print out your three estimates of I.

```r
h <- function(x) {exp(-0.5 * (x - 2) ^ 2 - 0.1 * abs( sin(2 * x) ) )}

## Monte Carlo Integration using uniform random sampling
n = 5000 # total number of values to generate
k = 500  # number of values to plot

# First series
set.seed(1)
# Generate n points from Unif(0,1)
x1 <- runif(n, 0, 5)
# Computer h(x)
h_x1 <- h(x1)
# Compute cumulative mean(h(X))
hbar_k1 <- (5/(1:k))*cumsum(h_x1[1:k])
# Compute mean for I_hat
E_hbar_k1 <- mean(hbar_k1)

# second series
set.seed(2)
# Generate n points from Unif(0,1)
x2 <- runif(n, 0, 5)
# Computer h(x)
h_x2 <- h(x2)
# Compute cumulative mean(h(X))
hbar_k2 <- (5/(1:k))*cumsum(h_x2[1:k])
# Compute mean for I_hat
E_hbar_k2 <- mean(hbar_k2)

# third series
set.seed(3)
# Generate n points from Unif(0,1)
x3 <- runif(n, 0, 5)
# Computer h(x)
h_x3 <- h(x3)
# Compute cumulative mean(h(X))
hbar_k3 <- (5/(1:k))*cumsum(h_x3[1:k])
# Compute mean for I_hat
E_hbar_k3 <- mean(hbar_k3)

# True Value (=2.29583) vs. Estimates
df_compare1 <- matrix(c(2.29583, E_hbar_k1,
                              E_hbar_k2, E_hbar_k3))
rownames(df_compare1) <- c("True Value", "Estimate 1",
                                "Estimate 2", "Estimate 3")
colnames(df_compare1) <- c("Expected Values")
df_compare1
```

```
##              Expected Values
## True Value          2.295830
## Estimate 1          2.447065
## Estimate 2          2.214991
## Estimate 3          2.378498
```

```r
# plot with all three series
plot(1:k, hbar_k1, type = "l" , xlab = "n",
     ylab = "running estimate", col = "red",
     ylim = c(1.5, 4.4), main = "Problem 2a Plot")
lines(hbar_k2, col = "green")
lines(hbar_k3, col = "blue")
abline(h = 2.29583, lty = 2)
legend(380, 4.45, legend = c("True Value", "Uniform Estimate 1",
                            "Uniform Estimate 2", "Uniform Estimate 3"),
       col = c("black", "red", "green", "blue"),
       lty = c(2,1,1,1), cex = 0.6)
```



Problem 2b

We can estimate the variance of $\bar{h}_n$ with $v_n = \frac{1}{n^2}\sum_{j=1}^{N}[h(x_j) - \bar{h}_n]^2$.

What is variance of the estimate $\hat{I}$?

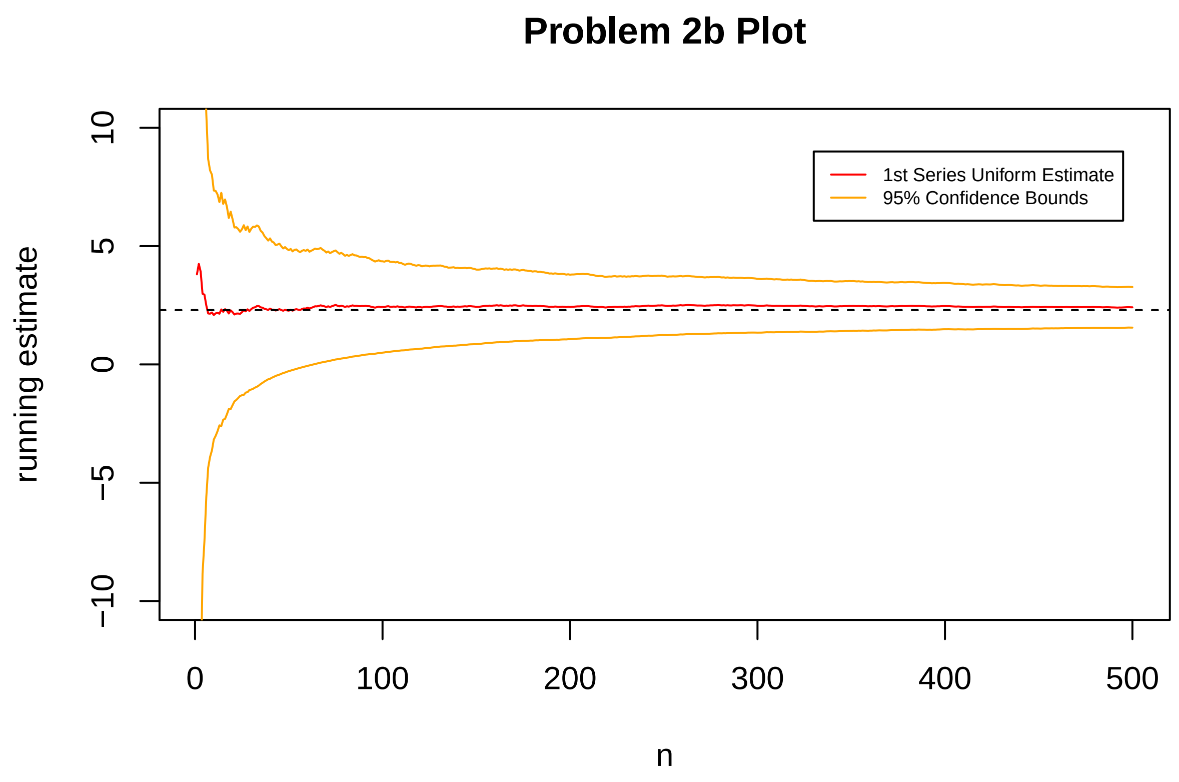
$$Var(\hat{I}) = (b-a)^2 Var(\bar{h}_n) = \frac{(b-a)^2}{n^2}\sum_{j=1}^{N}[h(x_j) - \bar{h}_n]^2$$

Look at your first Monte Carlo series and estimate the running variance of $\hat{I}$ as n goes from 1 to 500.

Create a plot of the $\hat{I}$ with 95% confidence bounds above and below.

```
# Function to estimate Var(hbar_n)
var_m1 <- function(m) {
# Estimate Var(hbar_m) for any given m
((5-0)^2) * sum((h_x1[1:m] - hbar_k1[m]) ^ 2) / m ^ 2
}
# running estimates of the variance
v_k1 <- apply(t(1:k), 2, var_m1)
# Compute standard error
se_k1 <- sqrt(v_k1)
```

```
# Plot the first Monte Carlo series with Confidence bounds of 95%
plot(1:k, hbar_k1, type = "l" , xlab = "n",
     ylab = "running estimate", col = "red",
     ylim = c(-10, 10), main = "Problem 2b Plot")
abline(h = 2.29583, lty = 2)
# Add approximate 95% confidence band
lines(hbar_k1 + 1.96 * se_k1, col = "orange")
lines(hbar_k1 - 1.96 * se_k1, col = "orange")
legend(330, 9, legend = c("1st Series Uniform Estimate",
                          "95% Confidence Bounds"),
       col = c("red", "orange"), lty = 1:1, cex = 0.6)
```



Importance Sampling

With importance sampling, we won't draw from the distribution $f(x)$, but from a proposal distribution $g(x)$. This can be useful if we don't know how to draw samples directly from $f(x)$.

We revisit the previous problem with importance sampling. Even though it is easy to draw from $f(x)$, which is the uniform distribution, we may see that it can be advantageous to draw from a different distribution that more closely resembles the target function.

We will use something that looks like the normal distribution N(2, 1) as the trial distribution to estimate the same integral by importance sampling.

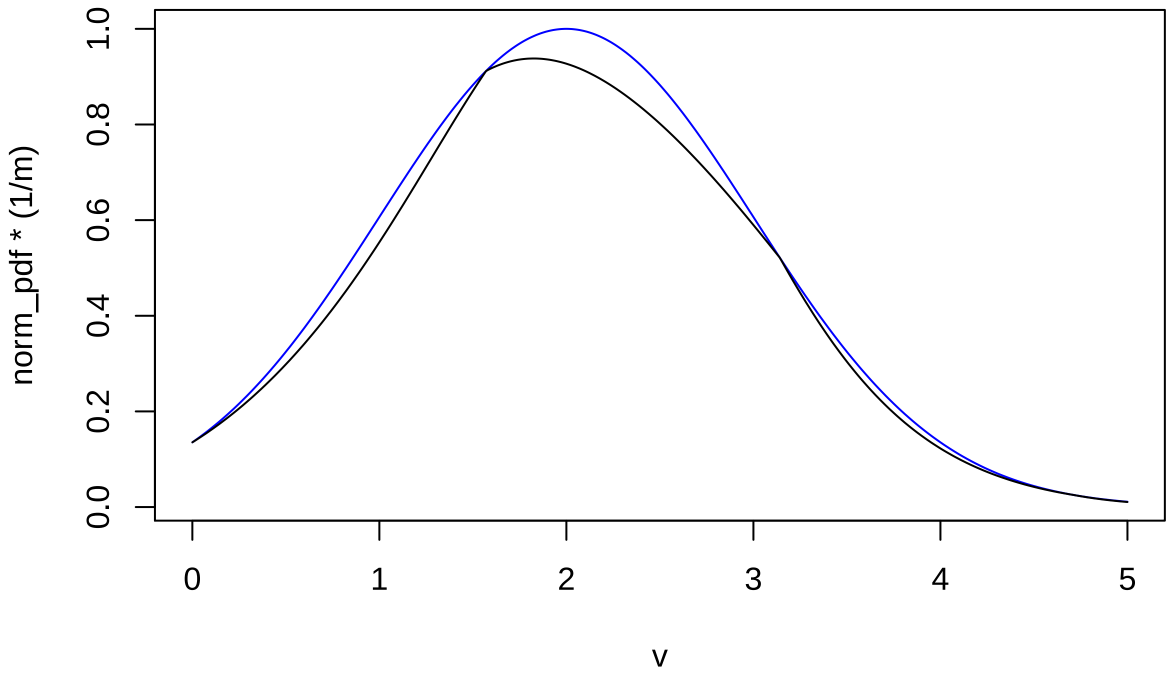
$$I = \int_0^5 \exp(-0.5(x-2)^2 - 0.1|\sin(2x)|)dx$$

$$I = 5 \cdot \mathbb{E}_f[h(X)] = 5 \cdot \int_0^5 h(x)f(x)dx = 5 \cdot \int_0^5 h(x)\frac{f(x)}{g(x)}g(x)dx = 5 \cdot \mathbb{E}_g\left[h(X)\frac{f(x)}{g(x)}\right] \approx \frac{5}{N}\sum_{j=1}^{N} h(x_j)\frac{f(x_j)}{g(x_j)}$$

```r
# what the function looks like, along with the normal distribution
h <- function(x) {exp(-0.5 * (x - 2) ^ 2 - 0.1 * abs( sin(2 * x) ) )}
v <- seq(0, 5, by = 0.01)
norm_pdf <- dnorm(v, mean = 2, sd = 1)

optimize(f = function(x){  dnorm(x,2,1) / h(x) }, interval = c(0, 5))
```

```
## $minimum
## [1] 3.141592
##
## $objective
## [1] 0.3989423
```

```r
# The function inside optimize is the proposal g(x) divided by the function h(x).
# Optimize gives the location where the ratio between the proposal pdf and target function is smallest.
# If we multiply the proposal distribution by 1/ratio, then the proposal distribution will always be >=
# target distribution. So we use this value as our constant M.

m <- optimize(f = function(x){  dnorm(x, 2, 1) / h(x) }, interval = c(0, 5))$objective
# Technically, this does not matter at all for importance sampling, but it makes it easier visually to
# see that the trial distribution matches the desired function quite well.

plot(v, norm_pdf * (1 / m), type = "l", col = "blue")  # trial distribution
lines(v, h(v), type = "l", col = "black")  # desired function
```



```
# we see a pretty good match between the trial distribution and desired function.
```

We will use `rnorm()` to generate random normal values and use importance sampling to estimate $\hat{I}$.

Keep in mind that $f(x) = 1/5$.

Let $g(x)$ be the proposal distribution we will use. $g(x)$ looks very much like the normal density with mean 2 and sd 1. However, it is slighlty different because we will throw away any values outside of the range (0, 5).

If we use the normal PDF directly, then $g(x)$ is not a probability density because it does not integrate to 1.

$$\int_0^5 \text{Normal PDF (2,1)} \neq \int_{-\infty}^{\infty} \text{Normal PDF (2,1)} = 1$$

To fix this, we need to find a normalizing constant.

$$g(x) = \frac{\text{Normal PDF (2,1)}}{Z_r}$$

Such that:

$$\int_0^5 g(x) = 1$$

Problem 3a

Find Z_r so that $\int_0^5 \frac{1}{Z_r}\text{Normal PDF (2,1)} = 1$. Hint: figure out how much of the distribution is 'cut off' at 0 and 5, and find Z_r accordingly.

```r
## your answer
Z_r <- 1 - (pnorm(0, 2, 1) + (1 - pnorm(5, 2, 1)))
Z_r
```

```
## [1] 0.9759
```

```r
# To confirm that the mean(g(x) * w(x)) = f(x)
g_x <- norm_pdf/Z_r
f_x <- (1/5)
w_x <- f_x/g_x
f_x
```

```
## [1] 0.2
```

```r
mean(g_x * w_x) # It is true
```

```
## [1] 0.2
```

Problem 3b

Perform Importance sampling.

Generate a sample of 5000 values of x to estimate $\hat{I}$.

This time, do not sample values from `runif()`, but rather from `rnorm()` with mean 2 and standard deviation 1. Make sure you remove values of x below 0 and above 5. Adjust how the estimate of the integral is calculated according to importance sampling as well as using the normalizing constant with the normal density.

Create a plot using the cumulative mean (aka running mean) of the first 500 values in the sample to show how the estimate of the integral 'settles' over the course of the sampling.

Do this two more times (each with a different starting seed), for a total of three samples of 5000 values each. Plot the cumulative mean of the first 500 values of each sample.

In your plot also include the 'true' value of the integral, 2.29583, as a horizontal line.

When you create your plot, also adjust the axes to fit the samples better.

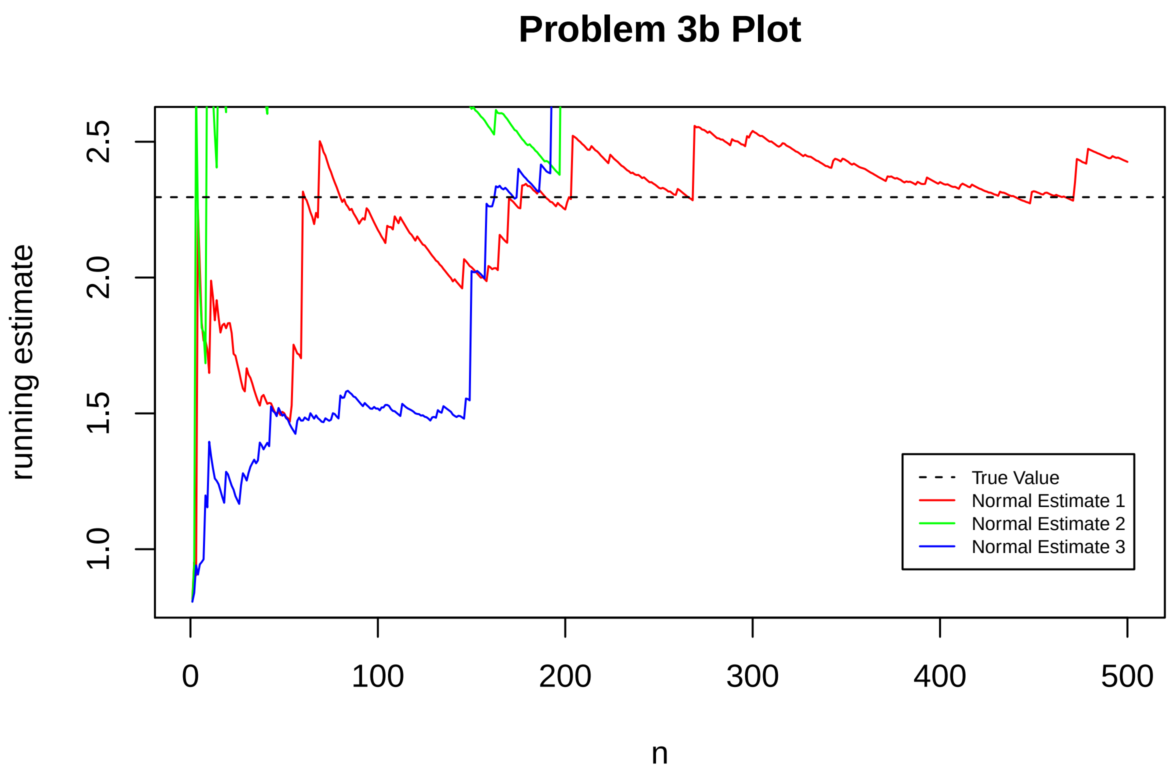
Finally, print out your three estimates of I, and also comment on how quickly the method using importance sampling converges to the expected value versus the uniform sampling method.

```r
## Monte Carlo Integration using importance sampling
n = 5000
k = 500

# First series
set.seed(1)
x_samp1 <- rnorm(n, 2, 1)
x_samp1 <- x_samp1[-which(x_samp1 > 5)]
x_samp1 <- x_samp1[-which(x_samp1 < 0)]
g_x1 <- dnorm(x_samp1, 2, 1)/Z_r
f_x1 <- 1/5
zbar_k1 <- (1/(1:k))*cumsum(x_samp1[1:k]*(f_x1/g_x1)[1:k])
E_zbar_k1 <- mean(zbar_k1)

# Second series
set.seed(2)
x_samp2 <- rnorm(n, 2, 1)
x_samp2 <- x_samp2[-which(x_samp2 > 5)]
```

```r
x_samp2 <- x_samp2[-which(x_samp2 < 0)]
g_x2 <- dnorm(x_samp2, 2, 1)/Z_r
f_x2 <- 1/5
zbar_k2 <- (1/(1:k))*cumsum(x_samp2[1:k]*(f_x2/g_x2)[1:k])
E_zbar_k2 <- mean(zbar_k2)

# Third series
set.seed(3)
x_samp3 <- rnorm(n, 2, 1)
x_samp3 <- x_samp3[-which(x_samp3 > 5)]
x_samp3 <- x_samp3[-which(x_samp3 < 0)]
g_x3 <- dnorm(x_samp3, 2, 1)/Z_r
f_x3 <- 1/5
zbar_k3 <- (1/(1:k))*cumsum(x_samp3[1:k]*(f_x3/g_x3)[1:k])
E_zbar_k3 <- mean(zbar_k3)

# True Value (=2.29583) vs. Estimates
df_compare2 <- matrix(c(2.29583, E_zbar_k1,
                              E_zbar_k2, E_zbar_k3))
rownames(df_compare2) <- c("True Value", "Estimate 1",
                                 "Estimate 2", "Estimate 3")
colnames(df_compare2) <- c("Expected Values")
df_compare2
```

```
##             Expected Values
## True Value        2.295830
## Estimate 1        2.250361
## Estimate 2        3.011482
## Estimate 3        2.330397
```

```r
# plot with all three series
plot(1:k, zbar_k1, type = "l" , xlab = "n",
     ylab = "running estimate", col = "red", main = "Problem 3b Plot")
lines(zbar_k2, col = "green")
lines(zbar_k3, col = "blue")
abline(h = 2.29583, lty = 2)
legend(380, 1.35, legend = c("True Value", "Normal Estimate 1",
                             "Normal Estimate 2", "Normal Estimate 3"),
       col = c("black", "red", "green", "blue"),
       lty = c(2,1,1,1), cex = 0.6)
```



What we find is that the values converge much faster. The reason for this is that $h(x)$ and $g(x)$ are nearly proportional to each other. That is to say that the variance of $(h(x)/g(x))$ is very low.

Self-Normalizing Importance Sampling

In the previous problem, we had to find the normalizing constant Z_C, so that the density from which we drew values would still integrate to 1. We drew values from the normal density, which is well understood, so calculating this constant Z_c was not too difficult.

In some situations, however, it is not always possible to calculate the normalizing constants for our densities.

We may only know a function $q(x)$ that is proportional to the target density $f(x)$. If is equal to $q(x)$ divided by some unknown normalizing constant Z_q:

$$f(x) = q(x)/Z_q$$

Similarly, there may be a function $r(x)$ that is proportional to the proposal distribution $g(x)$. In this scenario, we assume we are able to generate values from $g(x)$, but we just don't know the exact function equation for $g(x)$. Instead, we know the function $r(x)$ is proportional to $g(x)$, and the normalizing constant Z_r is unknown.

$$g(x) = r(x)/Z_r$$

As it applies to the current example

In our scenario, we can generate values from the proposal distribution $g(x)$. $g(x)$ is proportional to the normal distribution with mean 2 and standard deviation 1. We will call the PDF of the normal distribution

$r(x)$, keeping in mind that $g(x) = r(x)/Z_r$. We can generate random values easily using `rnorm()` and then throwing away any value larger than 5 or less than 0. We can also easily find the value of the function $r(x)$ by using `dnorm()`.

Unlike the previous problem, we will not calculate the normalizing constant Z_r for $g(x)$. Instead, we will perform self-normalizing Importance Sampling.

In self-normalizing importance sampling, we do not bother with calculating these normalizing constants.

We can estimate

$$\mathbb{E}_f[h(X)] \approx \frac{\sum_{j=1}^{N} h(x_j)w(x_j)}{\sum_{j=1}^{N} w(x_j)}$$

Where

$$w(x_j) = \frac{q(x_j)}{r(x_j)}$$

Problem 4

Perform Self-normalizing Importance sampling.

Generate a sample of 5000 values of x to estimate $\hat{I}$.

Let $f(x)$ be the uniform distribution from 0 to 5. You can let $q(x)$ be 0.2, and $Z_q = 1$.

Let $g(x)$ be proportional to the normal distribution with mean 2 and sd 1. We can let $r(x)$ be the normal PDF, and leave Z_r unknown.

Adjust how the estimate of the integral is calculated according to self-normalized importance sampling.

Create a plot using the cumulative mean (aka running mean) of the first 500 values in the sample to show how the estimate of the integral 'settles' over the course of the sampling.

Do this two more times (each with a different starting seed), for a total of three samples of 5000 values each. Plot the cumulative mean of the first 500 values of each sample.

In your plot also include the 'true' value of the integral, 2.29583, as a horizontal line.

When you create your plot, also adjust the axes to fit the samples better.

Finally, print out your three estimates of I, and also comment on how quickly the method using importance sampling converges to the expected value versus the uniform sampling method.

Keep in mind that because we are estimating the normalizing constant Z_r using our samples, the performance of the self-normalizing method will not be as good as when we knew Z_r directly.

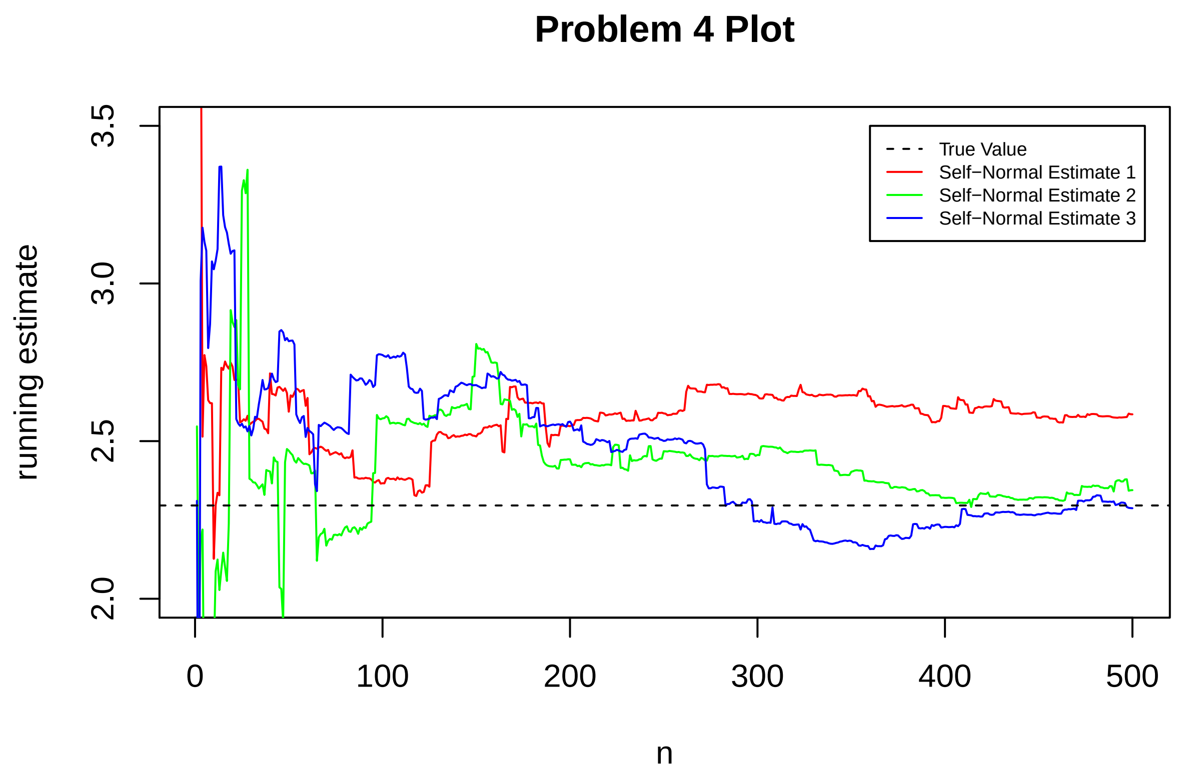
```r
## Monte Carlo Integration using self-normalizing importance sampling
n = 5000
k = 500

# Created a function for upper & lower bounds, sampling from rnorm(), and takes into account the (n) an
importance_samp_dnorm <- function(k, mu, sd, lower_bound, upper_bound, n) {
  norm_sample <- dnorm(n, mu, sd)
  norm_sample <- norm_sample[norm_sample >= lower_bound
                              & norm_sample <= upper_bound]
  if (length(norm_sample) >= k) {
    return(sample(norm_sample, k))
  }
  else {
          (print("(k) must be less than (n)"))
  }
}


# First series
set.seed(1)
# Z_r as unknown
X <- runif(n, 0, 5)
r_x <- importance_samp_dnorm(k = 500, mu = 2, sd = 1,
                              lower_bound = 0, upper_bound = 5, n = X)
q_x <- 0.2
w_x <- q_x/r_x
# Compute h(X)
h_X <- h(X)
# Compute cumulative means
wbar_k1 <- cumsum(w_x[1:k])/(1:k)
hwbar_k1 <- 5*cumsum((h_X*w_x)[1:k])/(1:k)
# Compute mean for I_hat
E_selfbar_k1 <- mean(hwbar_k1/wbar_k1)


# Second series
set.seed(2)
# Z_r as unknown
X <- runif(n, 0, 5)
r_x <- importance_samp_dnorm(k = 500, mu = 2, sd = 1,
                              lower_bound = 0, upper_bound = 5, n = X)
q_x <- 0.2
w_x <- q_x/r_x
# Compute h(X)
h_X <- h(X)
# Compute cumulative means
wbar_k2 <- cumsum(w_x[1:k])/(1:k)
hwbar_k2 <- 5*cumsum((h_X*w_x)[1:k])/(1:k)
# Compute mean for I_hat
E_selfbar_k2 <- mean(hwbar_k2/wbar_k2)


# Third series
set.seed(3)
# Z_r as unknown
X <- runif(n, 0, 5)
r_x <- importance_samp_dnorm(k = 500, mu = 2, sd = 1,
                              lower_bound = 0, upper_bound = 5, n = X)
q_x <- 0.2
w_x <- q_x/r_x
# Compute h(X)
h_X <- h(X)
# Compute mean(h(X))
mean(h_X)
```

```
## [1] 0.458739
```

```r
# Compute cumulative means
wbar_k3 <- cumsum(w_x[1:k])/(1:k)
hwbar_k3 <- 5*cumsum((h_X*w_x)[1:k])/(1:k)
# Compute mean for I_hat
E_selfbar_k3 <- mean(hwbar_k3/wbar_k3)


# True Value (=2.29583) vs. Estimates
df_compare3 <- matrix(c(2.29583, E_selfbar_k1,
                              E_selfbar_k2, E_selfbar_k3))
rownames(df_compare3) <- c("True Value", "Estimate 1",
                                 "Estimate 2", "Estimate 3")
colnames(df_compare3) <- c("Expected Values")
df_compare3
```

```
##            Expected Values
## True Value        2.295830
## Estimate 1        2.583607
## Estimate 2        2.416433
## Estimate 3        2.457986
```

```r
# plot with all three series
plot(1:k, hwbar_k1/wbar_k1, type = "l" , xlab = "n",
     ylab = "running estimate", col = "red", ylim = c(2, 3.5),
     main = "Problem 4 Plot")
lines(hwbar_k2/wbar_k2, col = "green")
lines(hwbar_k3/wbar_k3, col = "blue")
abline(h = 2.29583, lty = 2)
legend(360, 3.5, legend = c("True Value", "Self-Normal Estimate 1",
                             "Self-Normal Estimate 2",
                           "Self-Normal Estimate 3"),
       col = c("black", "red", "green", "blue"),
       lty = c(2,1,1,1), cex = 0.6)
```



What we find is that the values converge much slower than Problem 3b's estimates. The reason for this is that we are not using a normalizing constant. That is to say that the variance of $(f(x)/g(x))$ is quite high.