Assignment 4

xw-zeng

2022-12-01

Problem sets.

- `5.4`
- `5.5 (a)(b)`
- `6.3 (a)(b)`
- `6.5 (a)(b)`
- `6.9 (a)`

List of optimization functions.

- `Romberg_Integration`: Compute the `Triangular Array` of estimates in Romberg Integration method.
- `Importance_Sampling`: Estimation using Importance Sampling method with or without standardized weights.
- `Rejection_Sampling`: Rejection Sampling strategy with or without squeezing function.
- `SIS_1`: Generate 1 sample using Sequence Importance Sampling.
- `SIS_mn`: Generate m samples and n resamples using Sequence Importance Sampling.

Load the R packages.

```
library(ggplot2)
```

5.4

1. Briefly summarize Romberg Integration.

When $[a, b]$ is partitioned into n subintervals of equal length $h = (b-a)/n$, then the trapezoidal rule estimate is:

$$\int_a^b f(x)\mathrm{d}x \approx \frac{h}{2}f(a) + h\sum_{i=1}^{n-1} f(a+ih) + \frac{h}{2}f(b) = \widehat{T}(n)$$

Triangular Array of m = 6, in which $\widehat{T}_{i,0} = \widehat{T}(2^i)$.

$$\begin{array}{ccccccc}
\widehat{T}_{0,0} & & & & & & \\
\widehat{T}_{1,0} & \widehat{T}_{1,1} & & & & & \\
\widehat{T}_{2,0} & \widehat{T}_{2,1} & \widehat{T}_{2,2} & & & & \\
\widehat{T}_{3,0} & \widehat{T}_{3,1} & \widehat{T}_{3,2} & \widehat{T}_{3,3} & & & \\
\widehat{T}_{4,0} & \widehat{T}_{4,1} & \widehat{T}_{4,2} & \widehat{T}_{4,3} & \widehat{T}_{4,4} & & \\
\widehat{T}_{5,0} & \widehat{T}_{5,1} & \widehat{T}_{5,2} & \widehat{T}_{5,3} & \widehat{T}_{5,4} & \widehat{T}_{5,5} & \\
\widehat{T}_{6,0} & \widehat{T}_{6,1} & \widehat{T}_{6,2} & \widehat{T}_{6,3} & \widehat{T}_{6,4} & \widehat{T}_{6,5} & \widehat{T}_{6,6}
\end{array}$$

2. Apply Romberg Integration method to this problem.

X and Y:

$$X \sim Unif[1, a], Y = \frac{a-1}{X}$$

Pdf of X:

$$f(x) = \frac{1}{a-1}$$

Expectation of Y:

$$E(Y) = \int_1^a \frac{a-1}{x} f(x)\mathrm{d}x = \int_1^a \frac{1}{x}\mathrm{d}x$$

So the problem of calculating $E(Y)$ is transformed to computing the integral of $\frac{1}{x}$ on $[1, a]$.

Define the function $g(x) = \frac{1}{x}$.

```
g <- function(x){1 / x}
```

Define the function of computing the `Triangular Array` of estimates.

- `m`: Size of triangular array.
- `l`: lower bound of x.
- `u`: upper bound of x.

```
Romberg_Integration <- function(m, l, u){

  ###INITIAL VALUES###
  T_hat <- matrix(NA, nrow = m + 1, ncol = m + 1)
  width <- u - l

  ###MAIN###
  T_hat[1, 1] <- width * (g(l) + g(u)) / 2
  for (i in 1:m){ ##i is the first subscript of T_hat
    h <- width / (2 ^ i)
    T_hat[i + 1, 1] <- h * (g(l) + g(u)) / 2 + h * sum(g(seq(l + h, u - h, h)))
    for (j in 1:i){ ##j is the second subscript of T_hat
      T_hat[i + 1, j + 1] <- (4 ^ j * T_hat[i + 1, j] - T_hat[i, j]) / (4 ^ j - 1)}
  }

  ###OUTPUT###
  row.names(T_hat) = colnames(T_hat) = 0:m
  return(T_hat)
}
```

Compute the Triangular Array (taking $a = 10$ as an example).

```
Romberg_Integration(6, 1, 10); print(paste0('ln(10) = ', log(10)))
print(paste0('ERROR = ', Romberg_Integration(6, 1, 10)[7, 7] - log(10)))
```

```
##          0        1        2        3        4        5        6
## 0 4.950000       NA       NA       NA       NA       NA       NA
## 1 3.293182 2.740909       NA       NA       NA       NA       NA
## 2 2.629221 2.407901 2.385700       NA       NA       NA       NA
## 3 2.397737 2.320576 2.314754 2.313628       NA       NA       NA
## 4 2.327952 2.304690 2.303631 2.303455 2.303415       NA       NA
## 5 2.309061 2.302764 2.302635 2.302619 2.302616 2.302615       NA
## 6 2.304213 2.302598 2.302586 2.302586 2.302586 2.302586 2.302586
## [1] "ln(10) = 2.30258509299405"
## [1] "ERROR = 4.65695659812582e-07"
```

So it turns out that $E(Y) = loga$. $\widehat{T}_{6,6} - log(10) \approx 4.657e^{-7}$, while the theoretical error should be around $O(2^{-72})$, namely $O(1e^{-22})$. This difference may be attributed to the loss of precision in the floating-point arithmetic.

5.5

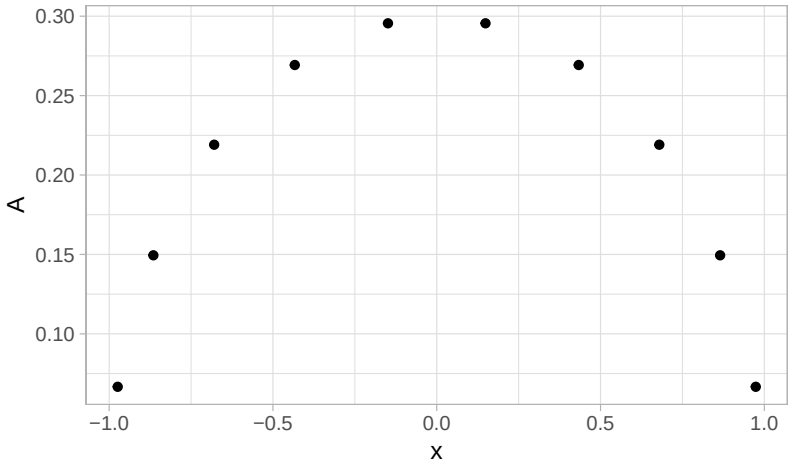
Import the nodes and weights for Gauss–Legendre quadrature on the range $[-1, 1]$.

```
x <- c(0.148874338981631, 0.433395394129247, 0.679409568299024,
       0.865063366688985, 0.973906528517172)
x <- c(-x[5:1], x)
A <- c(0.295524224714753, 0.269266719309996, 0.219086362515982,
       0.149451394150581, 0.066671344308688)
A <- c(A[5:1], A)
```

(a)

Plot the weights versus the nodes.

```
ggplot() + geom_point(aes(x, A), size = 1.5) + theme_light()
```



(b)

Compute the exact area under the curve $f(x) = x^2$ between -1 and 1.

$$\int_{-1}^{1} x^2 \mathrm{d}x = \frac{1}{3}x^3\Big|_{-1}^{1} = \frac{2}{3}$$

Define the function $f(x) = x^2$.

```
f <- function(x){x ^ 2}
```

Use `Gauss-Legendre Quadrature` method to compute the area under the curve $f(x) = x^2$ between -1 and 1.

```
gl_int <- t(A) %*% f(x); gl_int
```

```
##           [,1]
## [1,] 0.6666667
```

The difference between the approximation and the exact integration is:

```
gl_int - 2 / 3
```

```
##               [,1]
## [1,] 6.735012e-08
```

The precision is very high, which means Gaussian-Legendre Quadrature is very suitable for approximation of integration of polynomials if good nodes and corresponding weights are chosen.

6.3

(a)

Density of X:

$$f(x) \propto exp\{-\frac{|x|^3}{3}\}$$

Expectation of X^2:

$$\begin{aligned}\sigma^2 = E(X^2) &\propto \int_{-\infty}^{+\infty} x^2 exp\{-\frac{|x|^3}{3}\}\mathrm{d}x \\ &\propto \int_{-\infty}^{+\infty} x^2 exp\{-\frac{|x|^3}{3}\}exp\{\frac{x^2}{2}\}exp\{-\frac{x^2}{2}\}\mathrm{d}x \\ &\propto \int_{-\infty}^{+\infty} h(x)w^*(x)g(x)\mathrm{d}x\end{aligned}$$

In which $h(x) = x^2, w^*(x) = exp\{-\frac{|x|^3}{3}\}\big/exp\{-\frac{x^2}{2}\}, g(x) = exp\{-\frac{x^2}{2}\}$.

Given i.i.d. samples $X_1, X_2, ..., X_n$ drawn from $g(x)$, the estimator of Importance Sampling with standardized weights is:

$$\hat{\sigma}^2_{IS} = \sum_{i=1}^{n} h(X_i)w(X_i)$$

$$w(X_i) = w^*(X_i)/\sum_{j=1}^{n} w^*(X_j)$$

Define the function of `Importance Sampling` method.

- `n`: Number of samples chosen.
- `standard`: Whether the weights should be standardized.

```
Importance_Sampling <- function(n, standard = TRUE){

  ###INITIAL VALUES###
  X <- rnorm(n)

  ###MAIN###
  w_star <- exp(- abs(X) ^ 3 / 3) / exp(- X ^ 2 / 2)
  if (standard == TRUE){w <- w_star / sum(w_star); mu_is <- sum(X ^ 2 * w)}
  else {mu_is <- mean(X ^ 2 * w_star)}

  ###OUTPUT###
```

```
  return(mu_is)
}
```

Estimate σ^2 using Importance Sampling with standardized weights.

```
set.seed(5201314)
print(paste0('n = 100: ', Importance_Sampling(100)))
print(paste0('n = 1000: ', Importance_Sampling(1000)))
print(paste0('n = 10000: ', Importance_Sampling(10000)))
print(paste0('n = 100000: ', Importance_Sampling(100000)))
```

```
## [1] "n = 100: 0.765857116781197"
## [1] "n = 1000: 0.769879757895878"
## [1] "n = 10000: 0.787895300883557"
## [1] "n = 100000: 0.770514796228669"
```

(b)

Envelope:

$$\begin{aligned} e(x) &= exp\{-\frac{x^2}{2}\}/exp\{-\frac{1}{6}\} \\ &= (\frac{1}{\sqrt{2\pi}}exp\{-\frac{x^2}{2}\})\Big/(\frac{1}{\sqrt{2\pi}}exp\{-\frac{1}{6}\}) = \frac{g(x)}{\alpha} \end{aligned}$$

In which $g(x) = \frac{1}{\sqrt{2\pi}}exp\{-\frac{x^2}{2}\}, \alpha = \frac{1}{\sqrt{2\pi}}exp\{-\frac{1}{6}\}$.

Squeezing function:

$$s(x) = exp\{-\frac{x^4}{4} - \frac{1}{12}\}$$

Define the function of `Rejection Sampling` method.

- `n`: Target number of samples.
- `squeezed`: Whether the squeezing function should be introduced.

```
Rejection_Sampling <- function(n, squeezed = TRUE){

  ###INITIAL VALUES###
  keep <- 0; total <- 0; x <- rep(NA, n)

  ###FUNCTIONS###
  f <- function(x){exp(- abs(x) ^ 3 / 3)}
  e <- function(x){exp(- x ^ 2 / 2 - 1 / 6)}
  s <- function(x){exp(- x ^ 4 / 4 - 1 / 12)}
```

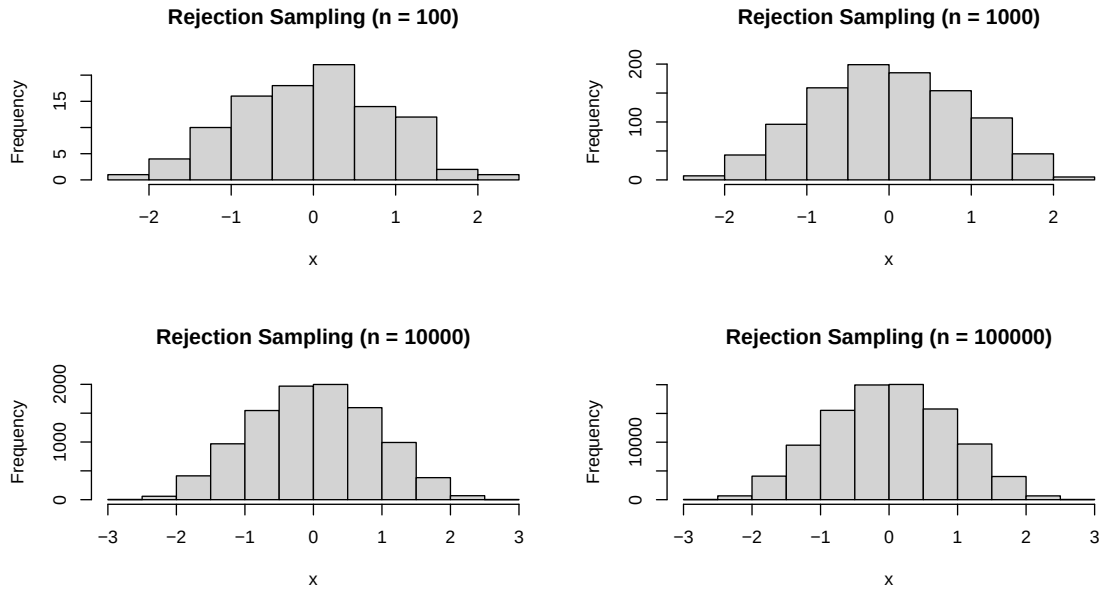
```
  ###MAIN###
  while (keep < n){
    total <- total + 1
    y <- rnorm(1) ##sample Y ~ g
    u <- runif(1) ##sample U ~ Unif(0,1)
    if (squeezed == TRUE & u <= s(y) / e(y)){ ##squeezed rejection
      keep <- keep + 1
      x[keep] <- y
      next
    }
    if (u <= f(y) / e(y)){
      keep <- keep + 1
      x[keep] <- y
    }
  }

  ###OUTPUT###
  print(paste0('Total Samples: ', total, '; Kept Samples: ', n,
               '; Acceptance Rate: ', round(n / total * 100, 2), '%'))
  return(x)
}
```

Define the function of plotting the distribution and estimating σ^2 using these random samples.

```
show_dist <- function(x){
  hist(x, main = paste0('Rejection Sampling (n = ', length(x), ')'))
  print(paste0('n = ', length(x), ': ', mean(x ^ 2)))
}
```

Generate random samples with Rejection Sampling method and estimate σ^2.

```
par(mfrow = c(2, 2))
set.seed(5201314)
show_dist(Rejection_Sampling(100))
show_dist(Rejection_Sampling(1000))
show_dist(Rejection_Sampling(10000))
show_dist(Rejection_Sampling(100000))
```




```
## [1] "Total Samples: 105; Kept Samples: 100; Acceptance Rate: 95.24%"
## [1] "n = 100: 0.78762759601256"
## [1] "Total Samples: 1042; Kept Samples: 1000; Acceptance Rate: 95.97%"
## [1] "n = 1000: 0.831436051368924"
## [1] "Total Samples: 10423; Kept Samples: 10000; Acceptance Rate: 95.94%"
## [1] "n = 10000: 0.798942145857464"
## [1] "Total Samples: 104209; Kept Samples: 1e+05; Acceptance Rate: 95.96%"
## [1] "n = 100000: 0.800199394593475"
```

The acceptance rate of Squeezed Rejection Sampling is about 95.96%, indicating that the envelope is very suitable and thus sampling is very efficient.

Now compare the time spent of Rejection Sampling with and without squeezing function.

```
set.seed(5201314)
start <- Sys.time(); s = capture.output(Rejection_Sampling(1000000, FALSE))
diff <- Sys.time() - start
print(paste0('Without Squeezing Function: ', round(diff, 2), 's'))
start <- Sys.time(); s = capture.output(Rejection_Sampling(1000000, TRUE))
diff <- Sys.time() - start
print(paste0('With Squeezing Function: ', round(diff, 2), 's'))
```

```
## [1] "Without Squeezing Function: 5.17s"
## [1] "With Squeezing Function: 4.04s"
```

It turns out that Squeezed Rejection Sampling is more computationally efficient.

6.5

(a)

Let $h_3(U_1, \dots, U_m) = h_2(1 - U_1, \dots, 1 - U_m)$.

Since both h_1 and h_2 is monotone in each argument, without loss of generality we can suppose that h_1 is increasing function and h_3 is decreasing function of $U_1, \dots, U_m$.

1. Univariate situation:

$$
\begin{aligned}
&[h_1(X) - h_1(Y)][h_3(X) - h_3(Y)] \le 0 \\
&E[h_1(X) \cdot h_3(X)] + E[h_1(Y) \cdot h_3(Y)] - E[h_1(X) \cdot h_3(Y)] - E[h_1(Y) \cdot h_3(X)] \le 0 \\
&2E[h_1(X) \cdot h_3(X)] - 2E[h_1(X)] \cdot E[h_3(X)] = 2Cov[h_1(X), h_3(Y)] \le 0
\end{aligned}
$$

2. Suppose the above conclusion holds for all $t < m$, then we have:

$$
\begin{aligned}
&Cov[h_1(U_1, \dots, U_t), h_3(U_1, \dots, U_t) \mid U_m] \le 0 \\
&E[h_1(U_1, \dots, U_t) h_3(U_1, \dots, U_t) \mid U_m] - E[h_1(U_1, \dots, U_t) \mid U_m] E[h_3(U_1, \dots, U_t) \mid U_m] \le 0 \\
\Rightarrow 0 &\ge E\{E[h_1(U_1, \dots, U_t) h_3(U_1, \dots, U_t) \mid U_m]\} - E\{E[h_1(U_1, \dots, U_t) \mid U_m] E[h_3(U_1, \dots, U_t) \mid U_m]\} \\
&= E\{E[h_1(U_1, \dots, U_t) h_3(U_1, \dots, U_t) \mid U_m]\} - E\{g_1(U_m) g_3(U_m)\} \\
&= E\{E[h_1(U_1, \dots, U_t) h_3(U_1, \dots, U_t) \mid U_m]\} - E\{g_1(U_m)\} E\{g_3(U_m)\} - Cov\{g_1(U_m), g_3(U_m)\} \\
&\ge E\{h_1(U_1, \dots, U_t) h_3(U_1, \dots, U_t)\} - E\{g_1(U_m)\} E\{g_3(U_m)\} \\
&= E\{h_1(U_1, \dots, U_t) h_3(U_1, \dots, U_t)\} - E\{E[h_1(U_1, \dots, U_t) \mid U_m]\} E\{E[h_3(U_1, \dots, U_t) \mid U_m]\} \\
&= E\{h_1(U_1, \dots, U_t) h_3(U_1, \dots, U_t)\} - E[h_1(U_1, \dots, U_t)] E[h_3(U_1, \dots, U_t)] \\
&= Cov[h_1(U_1, \dots, U_t), h_3(U_1, \dots, U_t)]
\end{aligned}
$$

Therefore, $Cov\{h_1(U_1, \dots, U_m), h_2(1 - U_1, \dots, 1 - U_m)\} \le 0 \Rightarrow$ proved.

(b)

Pairs of random variables independently generated: $(X_1, Y_1), \dots (X_n, Y_n)$

Control Variate for $\hat{\mu}_1$ with mean zero:

$$
\begin{aligned}
&\hat{\mu}_{MC} = \hat{\mu}_1(X) = \frac{1}{n} \sum_{i=1}^{n} \mu_1(X_i), \quad \hat{\mu}_2(Y) = \frac{1}{n} \sum_{i=1}^{n} \mu_2(Y_i) \\
&Z = \mu_2(Y) - \mu_1(X) \\
&E(Z) = E\{\mu_2(Y) - \mu_1(X)\} = E\{\mu_2(Y)\} - E\{\mu_1(X)\} = \mu - \mu = 0 \\
&\hat{Z} = \hat{\mu}_2(Y) - \hat{\mu}_1(X)
\end{aligned}
$$

Control Variate estimator:

$$\hat{\mu}_{CV} = \hat{\mu}_1(X) + \lambda\hat{Z} = (1-\lambda)\hat{\mu}_1(X) + \lambda\hat{\mu}_2(Y)$$

Derive the optimal λ. Notice that $\hat{\mu}_2(Y)$ is constructed from $Y_1, ..., Y_n$ chosen to be antithetic to $X_1, ..., X_n$, which means the variance of $\hat{\mu}_2(Y)$ is equivalent to $\hat{\mu}_1(X)$. Let $Var\{\hat{\mu}_1(X)\} = Var\{\hat{\mu}_2(Y)\} = \sigma^2$ and let the correlation coefficient be $\rho < 0$.

$$\begin{aligned}
E\{\hat{\mu}_{CV}\} &= \mu, \quad \text{for a given } \lambda \\
Var\{\hat{\mu}_{CV}\} &= (1-\lambda)^2\sigma^2 + \lambda^2\sigma^2 + 2\lambda(1-\lambda)\rho\sigma^2 \\
&= \sigma^2(1 + \lambda^2 - 2\lambda + \lambda^2 + 2\rho\lambda - 2\rho\lambda^2) \\
&= \sigma^2\{(2-2\rho)\lambda^2 - (2-2\rho)\lambda + 1\} \\
&= \sigma^2\{(2-2\rho)(\lambda - \frac{1}{2})^2 + \frac{1+\rho}{2}\} \\
&\geq \frac{1+\rho}{2}\sigma^2
\end{aligned}$$

Consequently, the variance of μ_{CV} reaches minimum when $\lambda = 0.5$.

6.9

(a)

Probability distribution for the bug's path through time t:

$$f_t\left(x_{1:t}\right) = f_1\left(x_1\right) f_2\left(x_2 \mid x_1\right) \ldots f_t\left(x_t \mid x_{1:t-1}\right)$$

Bridging distribution:

$$\begin{aligned} f_t\left(x_{1:t}\right) &\propto exp\{-(|v_t| + |w_t|) - R_t(x_t)/2\} \\ f_t\left(x_t|x_{1:t-1}\right) &= f_t\left(x_{1:t}\right) / f_{t-1}\left(x_{1:t-1}\right) \\ &= exp\{-(|v_t| + |w_t|) - R_t(x_t)/2 + (|v_{t-1}| + |w_{t-1}|) + R_{t-1}(x_{t-1})/2\} \end{aligned}$$

Sampling distribution:

$$g_t\left(x_t|x_{1:t-1}\right) = \frac{1}{4}$$

Importance weights at step t:

$$\begin{aligned} w_t\left(x_{1:t}\right) =& \frac{f_1\left(x_1\right) f_2\left(x_2 \mid x_{1:1}\right) f_3\left(x_3 \mid x_{1:2}\right) \cdots f_t\left(x_t \mid x_{1:t-1}\right)}{g_1\left(x_1\right) g_2\left(x_2 \mid x_{1:1}\right) g_3\left(x_3 \mid x_{1:2}\right) \cdots g_t\left(x_t \mid x_{1:t-1}\right)} \\ =& w_{t-1}\left(x_{1:t-1}\right) \frac{f_t\left(x_t \mid x_{1:t-1}\right)}{g_t\left(x_t \mid x_{1:t-1}\right)} \\ =& w_{t-1}\left(x_{1:t-1}\right) u_t \end{aligned}$$

Define the function of `Sequence Importance Sampling` method (One Sample).

- `t`: Time t in a sequence.

```
SIS_1 <- function(t){

  ###INITIAL VALUES###
  x <- matrix(NA, nrow = t + 1, ncol = 2); x[1, ] <- c(0, 0); freq_list <- c()

  ###FUNCTIONS###
  f_c <- function(oldx, newx){ ##compute conditional density function
    oldname <- paste0('(', oldx[1], ',', oldx[2], ')')
    newname <- paste0('(', newx[1], ',', newx[2], ')')
    out <- exp(- sum(abs(newx)) + sum(abs(oldx)) -
                 ifelse(newname %in% names(freq_list), freq_list[newname], 0) / 2 +
                 ifelse(oldname %in% names(freq_list), freq_list[oldname], 0) / 2)
    return(out)
  }
```

```
  next_step <- function(oldx){ ##generate sample from g
    idx <- sample(1:4, 1); newx <- oldx
    if (idx == 1){newx[1] <- oldx[1] + 1}
    else if (idx == 2){newx[2] <- oldx[2] - 1}
    else if (idx == 3){newx[1] <- oldx[1] - 1}
    else if (idx == 4){newx[2] <- oldx[2] + 1}
    return(newx)
  }

  update_freq <- function(newx){ ##update the frequency list of coordinates
    name <- paste0('(', newx[1], ',', newx[2], ')')
    if (name %in% names(freq_list)){freq_list[name] <- freq_list[name] + 1}
    else {freq_list[name] <- 1}
    return(freq_list)
  }

  generate_u <- function(oldx, newx){ ##compute the incremental weight
    4 * f_c(oldx, newx)
  }

  ###MAIN###
  x[2, ] <- next_step(x[1, ])
  freq_list <- update_freq(x[2, ])
  w <- 4 * f_c(x[1, ], x[2, ])
  for (i in 1:(t - 1)){
    x[i + 2, ] <- next_step(x[i + 1, ])
    freq_list <- update_freq(x[i + 2, ])
    u <- generate_u(x[i + 1, ], x[i + 2, ])
    w <- w * u
  }

  ###OUTPUT###
  structure(list(x = x, freq = freq_list, weight = w))
}
```

Define the function of `Sequence Importance Sampling` method (including resample).

- `t`: Time t in a sequence.
- `m`: Sample size.
- `n`: Resample size. Notice that $n/m \leq 1/10$ is required for distributional convergence.

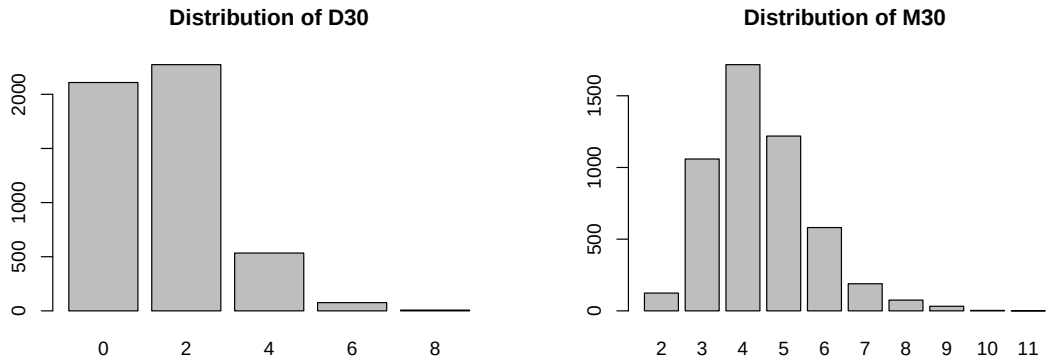
```
SIS_mn <- function(t, m, n){

  ###INITIAL VALUES###
  D_list <- c(); M_list <- c(); w_list <- c()

  ###MAIN###
  for (i in 1:m){
    result <- SIS_1(t)
    D_list <- c(D_list, sum(abs(result$x[t + 1, ])))
    M_list <- c(M_list, max(result$freq))
    w_list <- c(w_list, result$weight)
  }
  w_list_std <- w_list / sum(w_list)
  D_list_res <- sample(D_list, n, replace = TRUE, prob = w_list_std)
  M_list_res <- sample(M_list, n, replace = TRUE, prob = w_list_std)

  ###OUTPUT###
  structure(list(D = D_list, D_res = D_list_res, M = M_list,
                 M_res = M_list_res, w = w_list, w_std = w_list_std))
}
```

Generate samples using `Sequence Importance Sampling` method.

```
set.seed(1314); result <- SIS_mn(30, 100000, 5000)
```

Simulate the distribution of $D_{30}(x_{30})$ and $M_{30}(x_{1:30})$.

```
par(mfrow = c(1, 2))
barplot(table(result$D_res), main = 'Distribution of D30')
barplot(table(result$M_res), main = 'Distribution of M30')
```



Estimate the mean of $D_{30}(x_{30})$ and $M_{30}(x_{1:30})$. $w_t^{(i)}$ is the standardized weight.

$$\hat{\mu}_t = \sum_{i=1}^{m} w_t^{(i)} h\left({x_{1:t}}^{(i)}\right)$$

```
D_mean <- sum(result$D * result$w_std)
M_mean <- sum(result$M * result$w_std)
print(paste0('Mean of D30: ', D_mean))
print(paste0('Mean of M30: ', M_mean))
```

```
## [1] "Mean of D30: 1.45312853099466"
## [1] "Mean of M30: 4.449601128715"
```

Estimate the standard deviation of $D_{30}(x_{30})$ and $M_{30}(x_{1:30})$.

$$\hat{\sigma}_t = \left[\frac{1}{1 - \sum_{i=1}^{n}\left(w_t^{(i)}\right)^2} \sum_{i=1}^{n} w_t^{(i)} \left[h\left({x_{1:t}}^{(i)}\right) - \hat{\mu}_t\right]^2\right]^{\frac{1}{2}}$$

```
D_std <- sqrt(sum(result$w_std * (result$D - D_mean) ^ 2) /
                (1 - sum(result$w_std ^ 2)))
M_std <- sqrt(sum(result$w_std * (result$M - M_mean) ^ 2) /
                (1 - sum(result$w_std ^ 2)))
print(paste0('Standard deviation of D30: ', D_std))
print(paste0('Standard deviation of M30: ', M_std))
```

```
## [1] "Standard deviation of D30: 1.46246421541149"
## [1] "Standard deviation of M30: 1.26961147184237"
```