STAT 580 Homework 1 Yonghyun Kwon

February 23, 2020

1 Problem 1

Proof by Mathematical Induction:

1.1 step 1

If n = 1, $(X_1)\text{mod}1 = X_1 \sim U(0,1)$

1.2 step 2

Suppose it is true for n and write $\{x\} = (x)\text{mod}1 = x - \lfloor x \rfloor$: the fractional part of x. Then $\{\sum_{i=1}^{n+1} X_i\} = \{\sum_{i=1}^{n} X_i + X_{n+1}\} = \{\{\sum_{i=1}^{n} X_i\} + \{X_{n+1}\}\}$

By the assumption, $\{\sum_{i=1}^{n} X_i\} \sim U(0,1) \perp\!\!\!\perp \{X_{n+1}\} \sim U(0,1)$

Hence, it is only left to show that

$$\{U_1 + U_2\} \sim U(0,1) \;\; \text{where } U_1, U_2 \overset{iid}{\sim} U(0,1)$$

Recall that(convolution formula)

$$f_{X+Y}(a) = \int_{\infty}^{\infty} f_Y(a-x) f_X(x) dx$$

so the probability density function of $U_1 + U_2$ is

$$\begin{aligned} f_{U_1+U_2}(u) &= \int_{\infty}^{\infty} f_{U_2}(u-t) f_{U_1}(t) dt \\ &= \int_0^1 f_{U_2}(u-t) dt \\ &= \begin{cases} \int_0^u dt = u & 0 < u \le 1 \\ \int_{u-1}^1 dt = 2 - u & 1 < u < 2 \end{cases} \end{aligned}$$

Hence, for $0 \leq u \leq 1$,

$$\begin{aligned} P(\{U_1 + U_2\} \leq u) &= P(\{U_1 + U_2\} \leq u \wedge U_1 + U_2 \leq 1) + P(\{U_1 + U_2\} \leq u \wedge U_1 + U_2 > 1) \\ &= P(U_1 + U_2 \leq u) + P(1 < U_1 + U_2 \leq u + 1) \\ &= \int_0^u t dt + \int_1^{u+1} (2 - t) dt \\ &= \frac{1}{2}u^2 + (u - \frac{1}{2}u^2) = u \end{aligned}$$

Clearly, if $u < 0$, $P(\{U_1 + U_2\} \leq u) = 0$ and if $u > 1$, $P(\{U_1 + U_2\} \leq u) = 1$

This implies $U_1 + U_2 \sim U(0, 1)$

Thus, the assumption is true for n + 1.

2 Problem 2

2.1 Lemma

Suppose F is a cumulative distribution function of a random variable and $u \in (0, 1)$ and $x \in \mathbb{R}$

$$F^{-1}(u) \leq x \Leftrightarrow u \leq F(x)$$

proof:

$(\Rightarrow)$: By definition of F^{-1}, and since F is nondecreasing function, $u \leq F(F^{-1}(u)) \leq F(x)$

$(\Leftarrow)$: By definition of F^{-1}, for any y such that $F(y) \geq u, F^{-1}(u) \leq y$. By the assumption, $F(x) \geq u$, which implies $F^{-1}(u) \leq x$. $\square$

From the lemma above, we can show the following:

For $x \in \mathbb{R}$,

$$P(F^{-1}(U) \leq x) = P(U \leq F(x)) = F(x)$$

This implies $F^{-1}(U) \sim F$

3 Problem 3

3.1 (a)

Box-Muller Algorithm

i) Sampple $U_1, U_2 \overset{iid}{\sim} U(0, 1)$

ii) Compute $R = \sqrt{-2 \ln U_1}, \theta = 2\pi U_2$

iii) Compute $X = R \cos\theta, Y = R \sin\theta$

Then $(X, Y) \sim N_2(0, I_2)$

proof) Note that (X, Y) and (R, θ) are one-to-one and onto.

Suppose $(X, Y) \sim N_2(0, I_2)$ and find the distribution of (R, θ).

$$\begin{aligned} f_{R,\theta}(r, \theta) &= f_{X,Y}(r\cos\theta, r\sin\theta)|r| \\ &= \frac{1}{2\pi}\exp(-\frac{1}{2}r^2)rI\{r > 0\}I\{0 < \theta < 2\pi\} \end{aligned}$$

Since the joint pdf is splitted into two functions of θ and r, θ and R are independent.

Now integrating with respect to θ, $f_R(r) = \exp(-\frac{1}{2}r^2)rI\{r > 0\}$, which implies $f_\theta(\theta) = \frac{1}{2\pi}I\{0 < \theta < 2\pi\}$, so $\theta \overset{d}{=} 2\pi U(0, 1)$

Now suppose $R = \sqrt{-2\ln V}$ then

$$\begin{aligned} f_V(v) &= f_R(\sqrt{-2\ln v})\frac{1}{v\sqrt{-2\ln v}} \\ &= \exp(\ln v)\sqrt{-2\ln v}\frac{1}{v\sqrt{-2\ln v}}I\{0 < v < 1\} \\ &= I\{0 < v < 1\} \end{aligned}$$

Thus, $R \overset{d}{=} \sqrt{-2\ln U_1} \perp\!\!\!\perp \theta \overset{d}{=} 2\pi U_2$ $\square$

3.2 (b)

Polar Algorithm:

i) Sample $V_1, V_2 \overset{iid}{\sim} U(-1, 1)$ until $V_1^2 + V_2^2 < 1$

ii) Compute $W = V_1^2 + V_2^2, \quad c = \sqrt{-2W^{-1}\ln W}$

iii) Return $X = cV_1, \quad Y = cV_2$

proof) Consider the distribution of $V_1, V_2 | V_1^2 + V_2^2 < 1$ then

$$f_{V_1,V_2|V_1^2+V_2^2<1}(v_1, v_2) = \frac{1}{\pi}I\{v_1^2 + v_2^2 < 1\}$$

Write $V_1 = T\cos\theta, V_2 = T\sin\theta$, where $0 < \theta < 2\pi, \quad 0 < T < 1$

Then (V_1, V_2) and (T, θ) are one to one and onto.

Then the joint distribution of (T, θ) is

$$
\begin{aligned}
f_{T\theta}(t,\theta) &= f_{V_1V_2}(v_1,v_2)\Big|\frac{\partial(v_1,v_2)}{\partial(t,\theta)}\Big| \\
&= \frac{1}{2\pi}I\{0<\theta<2\pi\}2tI\{0<t<1\}
\end{aligned}
$$

Since the joint distribution is splited into function of θ and function of t, θ and T are independent and

$$\theta \sim U(0,1) \perp\!\!\!\perp T \sim f_T(t) = 2tI\{0<t<1\}$$

Observe that

$$
\begin{aligned}
W &= V_1^2 + V_2^2 = T^2 \\
c &= \frac{2\sqrt{-\log T}}{T} \\
X &= cV_1 = 2\sqrt{-\log T}\cos\theta \\
Y &= cV_2 = 2\sqrt{-\log T}\sin\theta
\end{aligned}
$$

The distribution of $R = 2\sqrt{-\log T}$ is

$$
\begin{aligned}
f_R(r) &= f_T(t)\Big|\frac{\partial t}{\partial r}\Big| \\
&= 2e^{-r^2/4}I\{r>0\}\frac{r}{2}e^{-r^2/4} \\
&= re^{-r^2/2}I\{r>0\}
\end{aligned}
$$

since $T = \exp(-\frac{R^2}{4})$, $\frac{\partial T}{\partial R} = -\frac{R}{2}\exp(-\frac{R^2}{4})$

Note, however, that we have shown that $(R\cos\theta, R\sin\theta) \sim N_2(0, I_2)$ if $R \sim f_R(r) = \exp(-\frac{1}{2}r^2)rI\{r>0\} \perp\!\!\!\perp \theta \sim f_\theta(\theta) = \frac{1}{2\pi}I\{0<\theta<2\pi\}$ in problem 3 (a). Thus, we conclude that $(X,Y) \sim N_2(0, I_2)$. $\square$

4 Problem 4

4.1 (a) Generating method

We try to sample from the truncated normal distribution $X = Z|Z > d$ where $Z \sim N(0,1)$.

That is, $f_X(x) = c_1\exp(-x^2/2)I\{x>d\}$, where c_1 is normalizing constant.

Set an envelop Y as $Y = \sqrt{V+d^2}$ where $V \sim \exp(2)$.

Applying changing of variables, we get

$$f_Y(y) = ye^{-\frac{y^2-d^2}{2}} I\{y > d\}$$

To choose an envelop constant c_2, find c_2 such that $c_2 f_Y(x) \geq f_X(x)$ for any $x > d$.

$$c_1 \exp(-\frac{1}{2}x^2) \leq c_2 x \exp(-\frac{x^2-d^2}{2}) \iff c_2 \geq \frac{c_1}{x} e^{-\frac{d^2}{2}} \Rightarrow \text{set } \ c_2 = \frac{c_1}{d} e^{-\frac{d^2}{2}}$$

Now we will accept Y if

$$\begin{aligned}
& f_X(Y) > U c_2 f_Y(Y) \\
\iff & c_1 \exp(-\frac{1}{2}Y^2) \leq U \frac{c_1}{d} e^{-\frac{d^2}{2}} Y \exp(-\frac{Y^2-d^2}{2}) \\
\iff & U < \frac{d}{Y}
\end{aligned}$$

To summarize, the generating method is

i) Generate $Y = \sqrt{V + d^2}$ where $V \sim \exp(2)$

ii) Generate $U \sim U(0, 1)$

iii) If $U < \frac{d}{Y}$, select Y, otherwise, go back to step i).

Here is the code to generate random deviates and acceptance rate. *rtnorm* generates random deviates from truncated standard normal distribution.

[1]:
```r
rtnorm <- function(n, d,..., rate = FALSE){
  res <- c() # random deviates
  if(rate == FALSE){ # Only generate random deviates
    for(i in 1:n){
      while(TRUE){
        y = sqrt(rexp(n = 1, rate = 1/2) + d^2)
        u = runif(1)
        if(u < d / y){
          res <- c(res, y)
          break
        }
      }
    }
    return(res)
  }else if(rate == TRUE){ # Also compute the acceptance rate
    cnt = 0 # number of all trials
    for(i in 1:n){
      while(TRUE){
        cnt = cnt + 1
        y = sqrt(rexp(n = 1, rate = 1/2) + d^2)
        u = runif(1)
        if(u < d / y){
```

```
            res <- c(res, y)
            break
          }
        }
      }
      return(list(res = res, rate = n / cnt))
    }
  }
```

acceptance function computes theoretical acceptance rate. Since $c_1 = \frac{1}{\Phi^{-1}(-d)\sqrt{2\pi}}$, the acceptance rate is

$$\frac{1}{c_2} = \frac{d}{c_1} e^{\frac{d^2}{2}} = \Phi^{-1}(-d)\sqrt{2\pi} d e^{\frac{d^2}{2}}$$

[2]:
```
acceptrate <- function(d){
  return(1/(1/pnorm(-d)/sqrt(2*pi)*exp(-d^2/2)/d))
}
```

4.2 (b) Details

Here are some random samples generated from tail of a standard normal distribution when $d = 1, 3, 5, 7$.

[7]:
```
dvec <- c(1,3,5,7)
sample1 <- sapply(dvec, function(x){rtnorm(10,x)})
colnames(sample1) <- dvec
cat("Tandom samples generated from truncated normal distribution, d = 1, 3, 5,␣
↪7")
sample1
```

```
random samples generated from truncated normal distribution, d = 1, 3, 5, 7
```

1	3	5	7
3.040835	3.131044	5.389290	7.186755
1.320584	3.387844	5.051107	7.238265
1.584693	3.702092	5.133704	7.226739
2.808587	3.446541	5.048883	7.027476
1.821270	3.720499	5.182413	7.084731
1.353542	3.676070	5.143730	7.050176
1.775521	3.084172	5.117327	7.080486
1.050506	3.551490	5.174406	7.287921
1.493640	3.568631	5.083861	7.429499
1.033487	3.385880	5.165718	7.075532

We can see that as d increases, more samples are generated far from d.

Also, we can obtain the mean of truncated standard normal distribution by

$$\mathbb{E}X = \int_d^\infty \frac{1}{\Phi^{-1}(d)} x\phi(x)dx = \frac{\phi(d)}{\Phi^{-1}(-d)}$$

where ϕ and Φ are the pdf and the cdf of standard normal distribution respectively.

So we can compare theoretical mean of truncated standard normal with the sample mean.

[8]:
```
cat("sample means for d = 1, 3, 5, 7")
sapply(dvec, function(x){mean(rtnorm(1000,x))})
cat("theoretical means for d = 1, 3, 5 ,7")
sapply(dvec, function(x){dnorm(x)/pnorm(-x)})
```

```
sample means for d = 1, 3, 5, 7
```

1. 1.51104966146325 2. 3.26936640965458 3. 5.18678675256878 4. 7.142338698849

```
theoretical means for d = 1, 3, 5 ,7
```

1. 1.52513527616098 2. 3.28309865493044 3. 5.18650396712584 4. 7.1375456132265

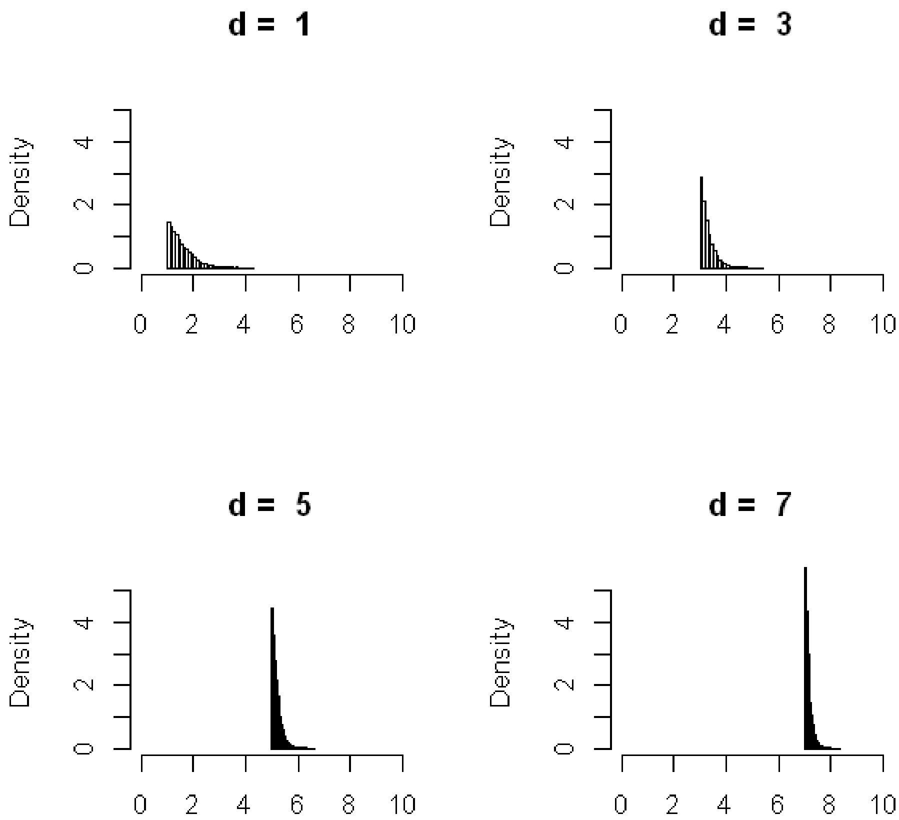
We can observe that the sample means are close to the theoretical mean because the sample size is large enough($n = 1000$).

4.3 (c) Figures and Tables

The pdf of truncated normal distribution can be approximated by histogram of the samples.

[18]:
```
par(mfrow = c(2,2))
for(i in dvec){
  hist(rtnorm(10000, i), xlim = c(0, 10), freq = FALSE, ylim = c(0, 5.5), xlab␣
↪= NULL,
       main = paste("d = ", i), breaks = 30)
}
cat("Histogram of the samples")
```

```
Histogram of the samples
```



For $d = 1, 3, 5, 7$, the rejection rate(or acceptance rate) and its 95% confidence interval can be obtained. Then we can compare them with theoretical rejection rate.

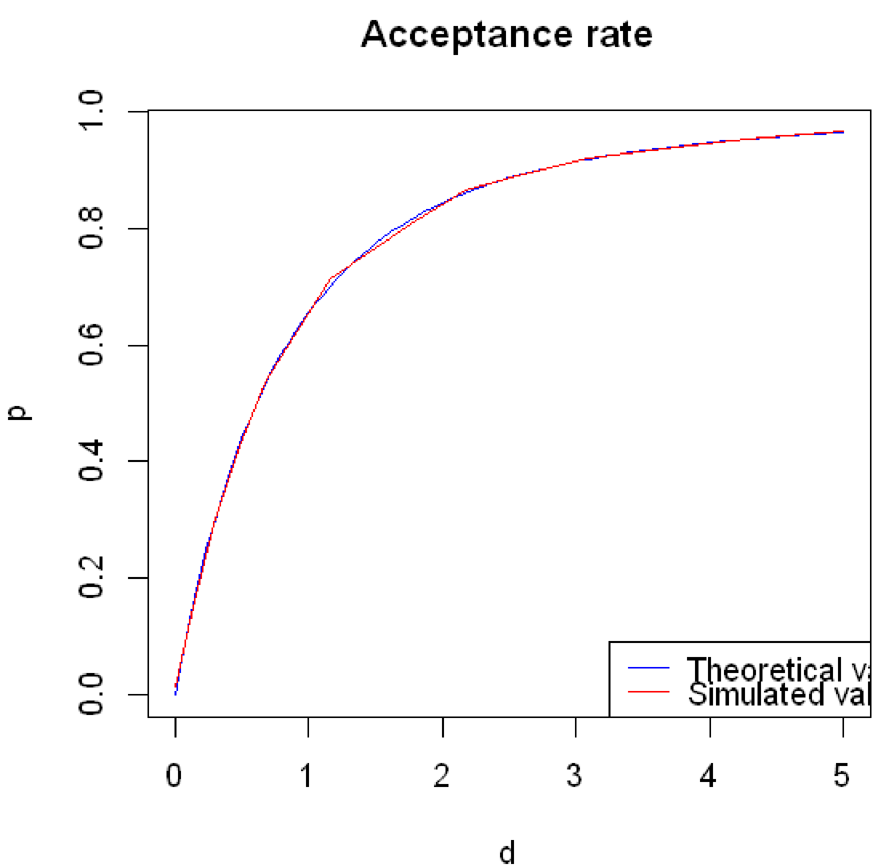
[15]:
```
table1 <- sapply(dvec, function(x){
  simul.rr = 1 - rtnorm(10000, x, rate = TRUE)$rate
  theo.rr = 1 - acceptrate(x)
  sd = sqrt(simul.rr * (1 - simul.rr) / 10000)
  upper = simul.rr + sd*qnorm(0.975)
  lower = simul.rr - sd*qnorm(0.975)
  return(c(theo.rr = theo.rr, simul.rr = simul.rr,
           upperCI = upper, lowerCI = lower
           ))})
colnames(table1) <- dvec
t(table1)
```

	theo.rr	simul.rr	upperCI	lowerCI
1	0.34432046	0.34400420	0.35331486	0.33469354
3	0.08622910	0.08675799	0.09227490	0.08124108
5	0.03595948	0.03400309	0.03755527	0.03045091
7	0.01927072	0.01787468	0.02047155	0.01527781

It turns out that the theoretical rejection rates lie in the 95% confidence interval for the most cases.

The acceptance rate can be computed and ploted for more general cases.. Here is the theoretical and simulated acceptance rate for $d \in (0, 5)$

[17]:
```
par(mfrow = c(1,1))
options(repr.plot.width=5, repr.plot.height=5)
curve(acceptrate, xlim = c(0, 5), col = 4, ylab = "p", xlab = "d", main =␣
 ↪"Acceptance rate")
seq1 <- exp(seq(from = 0.01, to = log(6), length.out = 15)) - 1
lines(x = seq1, y = sapply(seq1, function(x){rtnorm(10000, x, rate =␣
 ↪TRUE)$rate}), col = 2)
legend("bottomright", c("Theoretical value", "Simulated value"), col = c(4, 2),␣
 ↪lty = 1)
```



It is found that the theoretical and simulated acceptance rate are similar and they are increasing as d increases.

5 Problem 5

5.1 (a)

Assume that first index of a, r, c are 0.

First, initialize y to be n dimensional 0 vector.

Then, add each element in a to the appropriate element of y.

The algorithm in pseudo-code is as follows:

[ ]:
```
x, a, r, c are given

y[n] = {0} # initialize y to be n dimentional 0 vector
c = c - c[0] # Set the first element of the index pointer to be 0.

# j is the column index of W(j = 1, 2, ..., p+1)
# k is the index of a (or r)
for(j = 0, k = 0; k < length(a); k++){
    while(c[j+1] <= k){
        j++
    }
    y[r[k]] += a[k]*x[j]
}
```

5.2 (b)

5.2.1 i.

To develop a similar CCS-type strategy for storigng symmetric $p \times p$ matrix V, identify a symmetric matrix with $p \times p$ lower triangular matrix L. That is,

The lower and diagonal element of L is equal to V

The upper element of element of L is all zero.

Then V and L are one-to-one and on-to in that $V = L + L^T - \text{diag}(L)$, $L =$ (lower and diagonal part of symmetric matrix V)

Now, we can save storage by storing L instead of V by applying the very method suggested above.

5.2.2 ii.

Consdier an algorithm finding the i-th element y_i of $y = Vx$. And suppose V is stored in a way suggested above.

```
[ ]: x, a, r, c are given

y = 0 # initialize y_j
c = c - c[0] # Set the first element of the index pointer to be 0.

# j is the column index of W(j = 1, 2, ..., p+1)
# k is the index of a (or r)
for(j = 0, k = 0; k < length(a); k++){
    while(c[j+1] <= k){
        j++
    }
    if(r[k] == i){
        y += a[k]*x[j]
    }
    if(j == i){
        y += a[k]*x[r[k]]
        break
    }
}
```

6 Problem 6

6.1 (c)

For convenience, proof 6 - (c) first.

Since $\hat{\mu}_1$ and $\hat{\mu}_2$ do not change, we only focus on the change of

$$\sum_{i=1}^{n_Y} ||Y_i - \bar{Y}||^2 + \sum_{i=1}^{n_Z} ||Z_i - \bar{Z}||^2$$

Let $\bar{Y}_0$ and $\bar{Z}_0$ be existing sample mean and $\bar{Y}$ and $\bar{Z}$ be new sample mean. That is,

$$\bar{Y}_0 = \frac{1}{n_Y}\sum_{i=1}^{n_Y} Y_i, \quad \bar{Z}_0 = \frac{1}{n_Z}\sum_{i=1}^{n_Z} Z_i$$
$$\bar{Y} = \frac{1}{n_Y - 1}\sum_{i=1}^{n_Y-1} Y_i, \quad \bar{Z} = \frac{1}{n_Z + 1}\sum_{i=1}^{n_Z+1} Z_i$$

where $Z_{n_Z+1} = Y_{n_Y}$

Note the there is following relationship:

$$\bar{Y} = \frac{n_Y \bar{Y}_0 - Y_{n_Y}}{n_Y - 1}, \quad \bar{Z} = \frac{n_Z \bar{Z}_0 + Y_{n_Y}}{n_Z + 1}$$

Write $WSS_{Y_0} = \sum_{i=1}^{n_Y} ||Y_i - \bar{Y}||^2$, $WSS_Y = \sum_{i=1}^{n_Y - 1} ||Y_i - \bar{Y}||^2$ and define WSS_{Z_0} and WSS_Z similarly.

Now we try to express $WSS_Y - WSS_{Y_0}$ with respect to Y_{n_Y} and $\bar{Y}_0$

$$\begin{aligned}
WSS_Y &= \sum_{i=1}^{n_Y - 1} ||Y_i - \bar{Y}||^2 = \sum_{i=1}^{n_Y - 1} ||Y_i||^2 - (n_Y - 1)||\bar{Y}||^2 \\
&= \sum_{i=1}^{n_Y} ||Y_i||^2 - n_Y ||\bar{Y}_0||^2 - ||Y_{n_Y}||^2 - \frac{1}{n_Y - 1} ||n_Y \bar{Y}_0 - Y_{n_Y}||^2 + n_Y ||\bar{Y}_0||^2 \\
&= WSS_{Y_0} - ||Y_{n_Y}||^2 - \frac{1}{n_Y - 1} ||n_Y \bar{Y}_0 - Y_{n_Y}||^2 + n_Y ||\bar{Y}_0||^2
\end{aligned}$$

Hence,

$$\begin{aligned}
WSS_Y - WSS_{Y_0} &= -\frac{n_Y}{n_Y - 1} ||Y_{n_Y}||^2 + \frac{2n_Y}{n_Y - 1} < \bar{Y}_0, Y_{n_Y} > -\frac{n_Y}{n_Y - 1} ||\bar{Y}_0||^2 \\
&= -\frac{n_Y}{n_Y - 1} ||Y_{n_Y} - \bar{Y}_0||^2
\end{aligned}$$

Similarly, we can express $WSS_Z - WSS_{Z_0}$ with respect to Z_{n_Y} and $\bar{Z}_0$

$$\begin{aligned}
WSS_Z &= \sum_{i=1}^{n_Z + 1} ||Z_i - \bar{Z}||^2 = \sum_{i=1}^{n_Z + 1} ||Z_i||^2 - (n_Z + 1)||\bar{Z}||^2 \\
&= \sum_{i=1}^{n_Z} ||Z_i||^2 - n_Z ||\bar{Z}_0||^2 + ||Y_{n_Y}||^2 - \frac{1}{n_Z + 1} ||n_Z \bar{Z}_0 + Y_{n_Y}||^2 + n_Z ||\bar{Z}_0||^2 \\
&= WSS_{Z_0} + ||Y_{n_Y}||^2 - \frac{1}{n_Z + 1} ||n_Z \bar{Z}_0 + Y_{n_Y}||^2 + n_Z ||\bar{Z}_0||^2
\end{aligned}$$

which implies

$$\begin{aligned}
WSS_Z - WSS_{Z_0} &= \frac{n_Z}{n_Z + 1} ||Y_{n_Y}||^2 - \frac{2n_Z}{n_Z - 1} < \bar{Z}_0, Y_{n_Y} > +\frac{n_Z}{n_Z - 1} ||\bar{Z}_0||^2 \\
&= \frac{n_Z}{n_Z + 1} ||Y_{n_Y} - \bar{Z}_0||^2
\end{aligned}$$

Thus, the amount of change of total WSS is

$$WSS - WSS_0 = WSS_Y - WSS_{Y_0} + WSS_Z - WSS_{Z_0} = -\frac{n_Y}{n_Y - 1} ||Y_{n_Y} - \bar{Y}_0||^2 + \frac{n_Z}{n_Z + 1} ||Y_{n_Y} - \bar{Z}_0||^2$$

6.2 (a)

Now we go back to the problem 6 - (a). By identifying (U, W) with Y and (V, X) with Z in problem 6 - (c), we can follow the same steps tried above. Note, however that the existing sample mean is weighted mean.

Thus, the amount of change of total WSS is

$$WSS - WSS_0 = -\frac{n_U + n_W}{n_U + n_W - 1}||W_{n_W} - \frac{n_U \bar{U}_0 + n_W \bar{W}_0}{n_U + n_W}||^2 + \frac{n_V + n_X}{n_V + n_X + 1}||W_{n_W} - \frac{n_V \bar{V}_0 + n_X \bar{X}_0}{n_V + n_X}||^2$$

where $\bar{U}_0, \bar{W}_0, \bar{V}_0, \bar{X}_0$ are existing sample means of U, W, Y, X.

6.3 (b)

Similarly, after identifying (U, W) with Y and Y with Z in problem 6 - (c), we get

$$WSS - WSS_0 = -\frac{n_U + n_W}{n_U + n_W - 1}||W_{n_W} - \frac{n_U \bar{U}_0 + n_W \bar{W}_0}{n_U + n_W}||^2 + \frac{n_Y}{n_Y + 1}||W_{n_W} - \bar{Y}_0||^2$$

7 Problem 7

In problem 7, details about compiling procedure and discussions are provided as a comment in the code.

7.1 a

Observe that Celsious temperature(xºC) and the Fahrenheit temperature(yºF) has following relationship.

$$(y°F - 32) \times 5/9 = x°C$$
$$(x°C \times 9/5) + 32 = y°F$$

Based on the formula, we can write a c program to covert temperature.

[ ]:
```
/*
@file temperature.c
@author Yonghyun Kwon
  - converts temperature from Faherheit to the Celsius scale and vice-versa.
  - If the input value is 1, convert from F to C.
  - If the input value if 2, convert form C to F.

  -How to compile
  gcc -o temperature temperature.c -ansi -Wall -pedantic
```

```
  -How to excute
  ./temperature

  -Results(Example)
  First)
  From Fahernheit to Celsius, enter 1
  From Celsius to Fahernheit, enter 2
  1
  Enter a value: 0
  0.00 F = -17.78 C

  Second)
  From Fahernheit to Celsius, enter 1
  From Celsius to Fahernheit, enter 2
  2
  Enter a value: 0
  0.00 C = 32.00 F

  Third)
  From Fahernheit to Celsius, enter 1
  From Celsius to Fahernheit, enter 2
  123
  Please enter suitable input value(1 or 2)
*/

#include <stdio.h>

int main(void){
  int i; /* i is the indicator variable. If i == 1, F -> C; If i ==2, C -> F */
  double i2, res; /* i2 is the input value; res is the output value */
  printf("From Fahernheit to Celsius, enter 1\nFrom Celsius to Fahernheit, enter 2\n");
  scanf("%d", &i);
  if((i != 1) & (i != 2)){
    printf("Please enter suitable input value(1 or 2)\n");
    return 0;
  }
  printf("Enter a value: ");
  scanf("%lf", &i2);
  if(i == 1){
    res = (i2 -32) *5/9;
    printf("%.2f F = %.2f C\n", i2, res);
  }else if (i == 2){
    res = (i2 * 9/5) + 32;
    printf("%.2f C = %.2f F\n", i2, res);
  }
```

```
  return 0;
}
```

7.2 b

[ ]:
```
/* @file product.c
 * @author Yonghyun Kwon
 *
 * How to compile
 * gcc -o product product.c -ansi -Wall -pedantic
 *
 * How to execute
 * ./product
 *
 * Results
 * Type short: 16384
 * Type int: 81920
*/

# include <stdio.h>

int main(void){
  int i = 320; int j = 256;
  short res1 = i * j;
  int res2 = i * j;
  printf("Type short:%d\n", res1);
  printf("Type int:%d\n", res2);
  return 0;
}

/* We can find out that the type short output is 16384
 * and the type int output is 81920, which is expected.
 * Storing as short type, arithmetic overflow occurs,
 * since the maximum value of short type is 2^15 - 1 = 32767
 * Short type consists of 2 Bytes(=16 Bits), so 82920(mod 2^16) = 16384 is␣
 ↪obtained.
 * Storing as int type, however, artihmetic overflow does not occur,
 * since the maximum value of int type is 2^31 - 1.
 * Int type consists of 4 Bytes(=32 Bits) so 82920(mod 2^32) = 82920 is␣
 ↪obtained.
*/
```