Homework 1 **Name:** Ivan Wang

(Parts 1-3; 35 pts)

1. Investigate so-called vectorized operations. (8 pts)

a. Create (and print) a vector of length 8 using whose elements are all 10. This can be done using the rep(.) function.

```
> elements <- c(rep(10,8))
> elements
[1] 10 10 10 10 10 10 10 10
>
```

Creates vector that uses rep(.) to repeat 10, 8 times and assign it to elements

b. Define (and print) a vector of length 3 whose elements are [0 1 2].

```
> threeElements <- c(0,1,2)
> threeElements
[1] 0 1 2
>
```

Creates vector and assign it values 0,1,2

c. Provide the result of subtracting the vector in part (b) from the vector in part (a).

```
> subtract <- elements - threeElements
Warning message:
In elements - threeElements :
  longer object length is not a multiple of shorter object length
> subtract
[1] 10  9  8 10  9  8 10  9
>
```

Uses subtraction operation to subtract elements vectors from previous question

d. Consider the subtraction in the previous part. What does R do when told to perform a vectorized operation on vectors that have different lengths? Discuss both the printed warning message and the result of the subtraction.

R gives a warning message stating that the length of one of the vectors is longer then the shorter vectors.

As an result, R will just print the values by every 3 elements.

10–0, 10-1, 10-2 … repeating until there's no more.

2. Define matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, vector $\boldsymbol{C}$, and scalar d as shown below, and perform the indicated matrix operations using R commands. Note that $\boldsymbol{A}'$ denotes the transpose of matrix $\boldsymbol{A}$. (8 pts)

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 \\ 2 & 4 \\ -1 & 2 \end{bmatrix} \quad \boldsymbol{B} = \begin{bmatrix} 1 & 2 & -1 \\ 0 & 3 & -4 \end{bmatrix} \quad \boldsymbol{C} = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} \quad d = 2$$

a) $\boldsymbol{C}'\boldsymbol{A}$

```
> firstOp <- t(C) %*% A
> firstOp
     [,1] [,2]
[1,]    5    8
>
```

Perform transpose of matrix C multiply by matrix A

b) $\boldsymbol{A}'\boldsymbol{C}$

```
> secondOp <- t(A) %*% C
> secondOp
     [,1]
[1,]    5
[2,]    8
>
```

Perform transpose of matrix A multiply by matrix C

c) $\boldsymbol{B}' + d\boldsymbol{A}$

```
> thirdOp <- D*A + t(B)
> thirdOp
     [,1] [,2]
[1,]    3    0
[2,]    6   11
[3,]   -3    0
>
```

Perform transpose of matrix B and add it to scalar d multiply by matrix A

d) $(\boldsymbol{BA})'$

```
> forthOp <- t((B%*%A))
> forthOp
     [,1] [,2]
[1,]    6   10
[2,]    6    4
>
```

Perform transpose of matrix B multiply A

3. Use a *while* loop (and brute force) to identify the first positive integer whose natural log is larger than 10. Outside the loop, set a numeric variable (call it x) equal to 1. Inside the loop, print the value of log(x), and increase the value of x by 1. DO NOT submit the entirety of the output that is printed to the console. Instead, provide your code and the value of x that causes the loop to terminate. (4 pts)

```
x <- 1
while(log(x) < 10){
  print(paste("At", x, "the log value is", log(x)))
  x <- x + 1
}
```

```
[1] "At 22026 the log value is 9.99997885272489"
```

Creates x variable and assign it to 1. Create a while loop with condition log(x) < 10. Inside we print the value of log(x) and increase x by 1.

At x = 22026, the loop will terminate

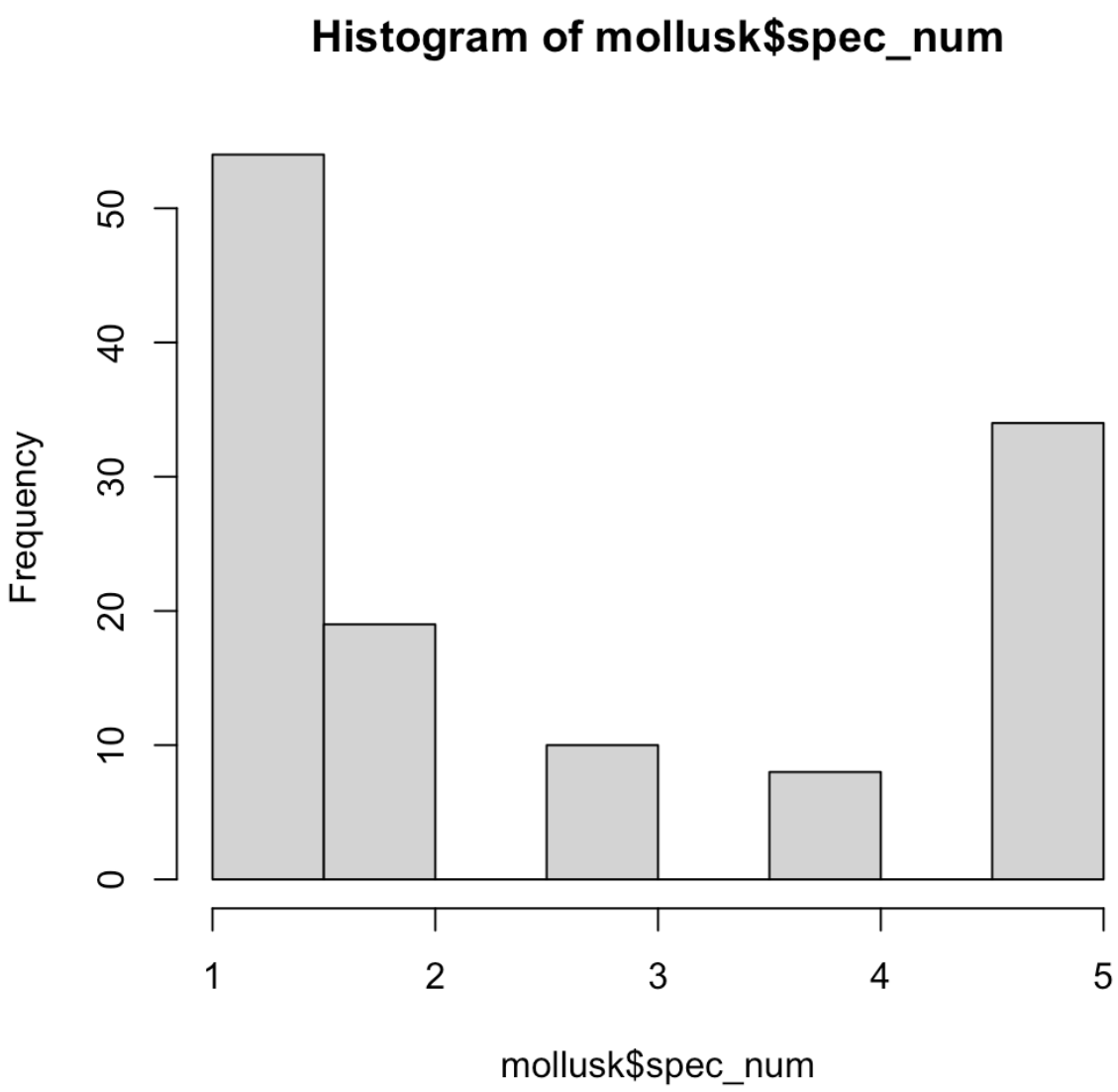
4. The data set mollusk.txt (UBLearns) contains data about five species of mollusk. (10 pts)

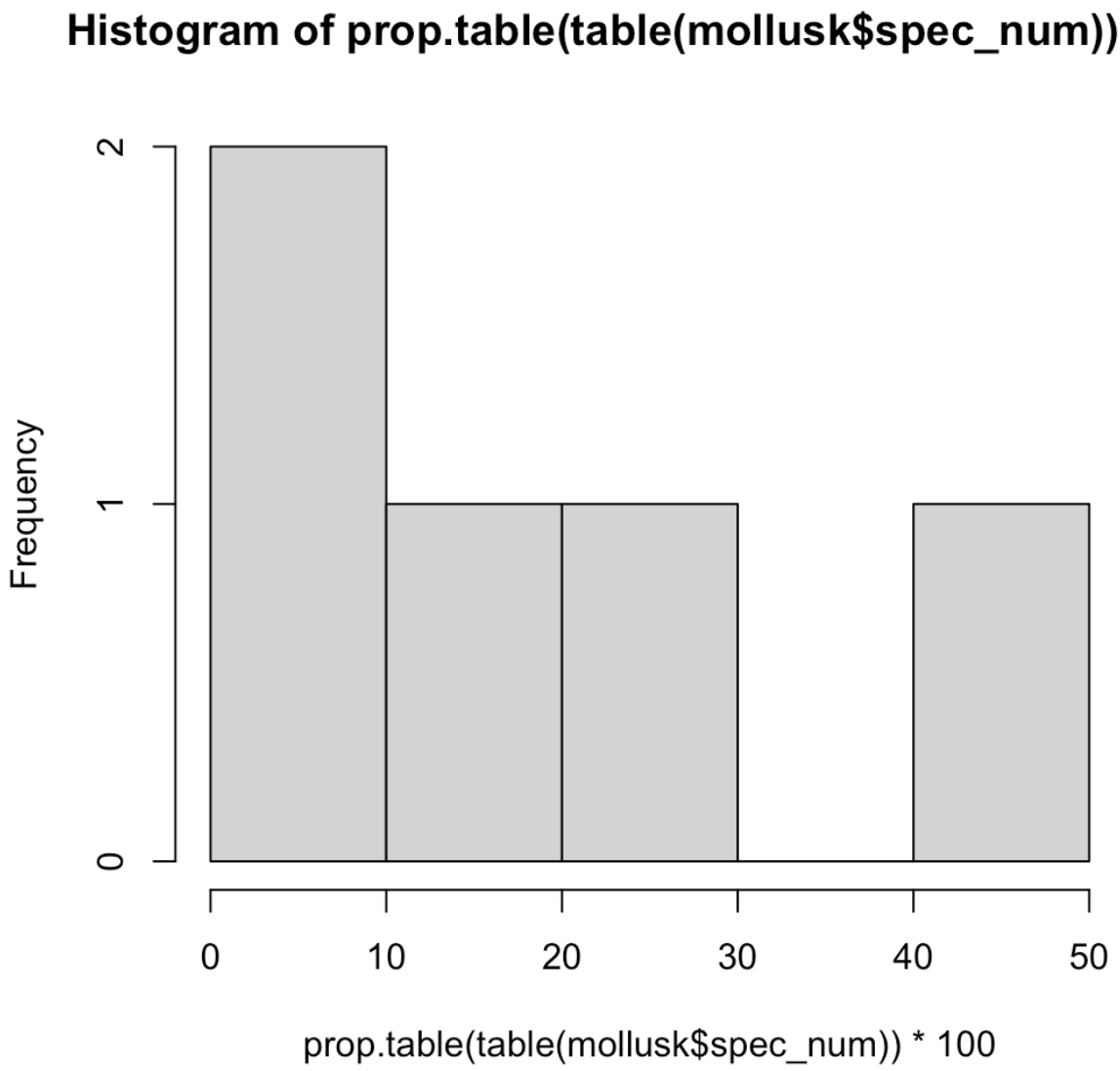
a) Obtain the frequency distribution (counts and percentages) of the Species Number variable.

Obtained the counts and percentages of the species number variable and display both in a histogram.

```
 1  2  3  4  5
54 19 10  8 34
> hist(mollusk$spec_num)
> prop.table(table(mollusk$spec_num)) * 100

   1    2    3    4    5
43.2 15.2  8.0  6.4 27.2
> hist(prop.table(table(mollusk$spec_num)) * 100 )
>
```


Histogram of mollusk$spec_num
Frequency
0
10
20
30
40
50
1
2
3
4
5
mollusk$spec_num



b) Report the sample mean and sample standard deviation of the Impulse Rate variable.

```
> mean <- mean(mollusk$impulse)
> sd <- sd(mollusk$impulse)
> mean
[1] 173.8722
> sd
[1] 176.131
>
```

Report the mean and standard deviation

c) Use the ifelse(.) function to define a vector called **fast** which classifies each mollusk's response time. The value of **fast** should be set to TRUE if a mollusk's recorded value of Impulse Rate is less than 100. Mollusks with Impulse Rate greater than or equal to 100 should have their value of **fast** set to FALSE.

```
> mollusk$fast <- ifelse(mollusk$impulse < 100, TRUE, FALSE)
> mollusk$fast
  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [24]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [47]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE
 [70] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [93] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[116] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
>
```

Assign vector fast and uses ifelse(.), to check if the mollusk's recorded value of Impulse Rate is less than 100. It should show TRUE or FALSE

d) Create a logical vector equivalent to the one produced in part (c). Instead of populating the vector using ifelse(.), use an inequality sign directly.

```
> mollusk$fast2 <- mollusk$impulse < 100
> mollusk$fast2
  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [24]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [47]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE
 [70] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [93] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[116] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Uses logical vector equivalent to the one produced in part (c).

e) Verify that the two vectors from parts (c) and (d) are equivalent using the identical(.) function.

```
> ?identical()
> identical(mollusk$fast, mollusk$fast2)
[1] TRUE
>
```

Uses identical function to check if previous two vectors are equivalent.

f) Give the total number of mollusks in the data set having **fast** equal to TRUE.

```
> sum(mollusk$fast == TRUE)
[1] 71
>
```

Use sum function and pass in mullusk$fast == TRUE to get the total number of mollusks with Impulse Rate is less than 100.

5. A graduate student is writing a program that requires 100 objects to be defined in advance, each of them a vector of length 5. For simplicity, the objects can be named obj_1, obj_2, …, obj_100. (5 pts)

a) The student considers use of the paste(.) function to combine the text string "obj" with the digit 1, separated by an underscore. Refer to the paste(.) function's help file, and use the function to produce the character string "obj_1."

```
> ?paste()
> obj1 <- paste ("obj", 1, sep = "_", collapse = NULL, recycle0 = FALSE)
> obj1
[1] "obj_1"
>
```

Create obj1 and assign it to "obj_1" using paste function

b) The student realizes that printing the characters "obj_1" to the console is insufficient; he intended to have a stored object in memory called **obj_1**, which he could set equal to an empty numeric vector of length 5. After some online reading, he discovers the assign(.) function, which allows the user to give both a variable name and the value assigned to that variable. Use the assign(.) function to create **obj_1**, and demonstrate the following result in console:

```
obj_1
[1] 0 0 0 0 0
```

```
> ?assign(.)
> obj_1 <- assign("obj_1", numeric(5))
> obj_1
[1] 0 0 0 0 0
>
```

Create obj_1 and use assign function to object in memory called "obj_1" with vector length 5

c) Adapt the code from part (b) to execute in a *for* loop, so that obj_1, obj_2, …, obj_100 are all created in memory, each equal to an empty numeric vector of length 5. No output is required for this part; just your code block.

```
"""
c)  Adapt the code from part (b) to execute in a for loop, so that obj_1, obj_2, …, obj_100 are all created in memory,
each equal to an empty numeric vector of length 5. No output is required for this part; just your code block.
"""

for (x in 1:100) {
  obj_x <- assign(paste("obj", x), numeric(5))
  print(paste("obj", x, "is created in memory"))
  print(obj_x)
}
```

Complete R code:

```
###1. Investigate so-called vectorized operations. (8 pts)

"""
a. Create (and print) a vector of length 8 using whose elements are all 10.
This can be done using the rep(.) function.
"""

elements <- c(rep(10,8))
elements

"""
b. Define (and print) a vector of length 3 whose elements are [0 , 1, 2].
"""
threeElements <- c(0,1,2)
threeElements


"""
c.	Provide the result of subtracting the vector in part (b) from the vector in part (a).
"""
subtract <- elements - threeElements
subtract

"""
2. Define matrices A and B, vector C, and scalar d as shown below, and perform the
indicated matrix operations using R commands. Note that A' denotes the transpose of
```

```
matrix A. (8 pts)
"""

A <- matrix(c(1,2,-1,0,4,2), nrow = 3, ncol = 2)
B <- matrix(c(1,0,2,3,-1,-4), nrow = 2, ncol = 3)
C <- matrix(c(1,2,0), nrow = 3, ncol = 1)
D <- 2
A
B
C

"""
a.	C'A
"""
firstOp <- t(C) %*% A
firstOp

"""
b.	A'C
"""
secondOp <- t(A) %*% C
secondOp


"""
c.	B'+dA
"""
thirdOp <- D*A + t(B)
```

```
thirdOp



"""
d.        (BA)'
"""
forthOp <- t((B%*%A))
forthOp



"""
3. Use a while loop (and brute force) to identify the first positive integer whose natural log
is larger than 10. Outside the loop, set a numeric variable (call it x) equal to 1. Inside
the loop, print the value of log(x), and increase the value of x by 1. DO NOT submit the entirety
of the output that is printed to the console. Instead, provide your code and the value of x that
causes the loop to terminate. (4 pts)
"""
x <- 1
while(log(x) < 10){
  print(paste("At", x, "the log value is", log(x)))
  x <- x + 1
}



"""
4. The data set mollusk.txt (UBLearns) contains data about five species of mollusk. (10 pts)
```

"""

"""

a) Obtain the frequency distribution (counts and percentages) of the Species Number variable.

"""

```
mollusk <- read.csv("~/Documents/STA 361/mollusk.txt", sep="")
View(mollusk)

table(mollusk$spec_num)
hist(mollusk$spec_num)

prop.table(table(mollusk$spec_num)) * 100
hist(prop.table(table(mollusk$spec_num)) * 100 )
```

"""

b) Report the sample mean and sample standard deviation of the Impulse Rate variable.

"""

```
mean <- mean(mollusk$impulse)
sd <- sd(mollusk$impulse)
mean
sd
```

"""

c) Use the ifelse(.) function to define a vector called fast which classifies each mollusk's response time.

The value of fast should be set to TRUE if a mollusk's recorded value of Impulse Rate is less than 100.

Mollusks with Impulse Rate greater than or equal to 100 should have their value of fast set to FALSE.

"""

```
mollusk$fast <- ifelse(mollusk$impulse < 100, TRUE, FALSE)
mollusk$fast
```

"""

d) Create a logical vector equivalent to the one produced in part (c). Instead of populating the vector using ifelse(.),

use an inequality sign directly.

"""

```
mollusk$fast2 <- mollusk$impulse < 100
mollusk$fast2
```

"""

e) Verify that the two vectors from parts (c) and (d) are equivalent using the identical(.) function.

"""

```
identical(mollusk$fast, mollusk$fast2)
```

"""

f) Give the total number of mollusks in the data set having fast equal to TRUE.

"""

```
sum(mollusk$fast == TRUE)
```

"""

A graduate student is writing a program that requires 100 objects to be defined in advance, each of them a vector of length 5.

For simplicity, the objects can be named obj_1, obj_2, …, obj_100. (5 pts)

"""

"""

a) The student considers use of the paste(.) function to combine the text string “obj” with the digit 1, separated by an underscore.

Refer to the paste(.) function’s help file, and use the function to produce the character string “obj_1.”

"""

```
obj1 <- paste ("obj", 1, sep = "_", collapse = NULL, recycle0 = FALSE)
obj1
```

"""

b) The student realizes that printing the characters “obj_1” to the console is insufficient; he intended to have a stored object

in memory called obj_1, which he could set equal to an empty numeric vector of length 5.

After some online reading, he discovers the assign(.) function, which allows the user to give both a

variable name and the value assigned to that variable. Use the assign(.) function to create obj_1, and

demonstrate the following result in console:

"""

obj_1 <- assign("obj_1", numeric(5))

obj_1

"""

c) Adapt the code from part (b) to execute in a for loop, so that obj_1, obj_2, …, obj_100 are all created in memory,

each equal to an empty numeric vector of length 5. No output is required for this part; just your code block.

"""

```
for (x in 1:100) {
  obj_x <- assign(paste("obj", x), numeric(5))
  print(paste("obj", x, "is created in memory"))
  print(obj_x)
}
```