STATS 782, 2017 **Assignment1** **Date: 2017-08-15**

Zhi Zhang, 708439475, zzha822

1. The answers are below:

   (a)
   ```
   > seq(1, 28, by = 3)
    [1]  1  4  7 10 13 16 19 22 25 28
   ```

   (b)
   ```
   > paste(rep(letters[-1:-23], 3:1), rep(letters[1:3], 2), sep = "")
   [1] "xa" "xb" "xc" "ya" "yb" "zc"
   ```

   (c)
   ```
   > rep(1:3 < 2, 2)
   [1]  TRUE FALSE FALSE  TRUE FALSE FALSE
   ```

   (d)
   ```
   > cumsum(10^(seq(0, 8, by = 2)))
   [1]         1       101     10101   1010101 101010101
   ```

   (e)
   ```
   > 1:4 + rep(0:3, rep(4, 4))
    [1] 1 2 3 4 2 3 4 5 3 4 5 6 4 5 6 7
   ```

2. The answers are below:

   (a)
   ```
   > URL = "https://raw.githubusercontent.com/zzdxzhangzhi/assignments/
       adb70032ccc89e683055814da4c87727fcaf332a/782/eurocities.csv"
   > eurocities.df = read.csv(URL, stringsAsFactors = FALSE)
   > str(eurocities.df)
   'data.frame': 209 obs. of  3 variables:
    $ Athens   : chr  "Athens" "Athens" "Athens" "Athens" ...
    $ Barcelona: chr  "Brussels" "Calais" "Cherbourg" "Cologne" ...
    $ X3313    : int  2963 3175 3339 2762 3276 2610 4485 2977 3030 4532 ...
   > head(eurocities.df)
     Athens  Barcelona X3313
   1 Athens   Brussels  2963
   2 Athens     Calais  3175
   3 Athens  Cherbourg  3339
   4 Athens    Cologne  2762
   5 Athens Copenhagen  3276
   6 Athens     Geneva  2610
   >
   > distance.mean = mean(eurocities.df$X3313)
   > distance.mean
   [1] 1496.498
   ```

   (b)
   ```
   > maxd = max(eurocities.df$X3313)
   > con = (eurocities.df$X3313 == maxd)
   > maxdnum = rownames(eurocities.df)[con]
   > eurocities.df[maxdnum, 1:2]
      Athens Barcelona
   10 Athens    Lisbon
   > maxdnum = which.max(eurocities.df$X3313)
   > maxpair = c(eurocities.df$Athens[maxdnum], eurocities.df$Barcelona[maxdnum])
   ```

```
> maxpair
[1] "Athens" "Lisbon"
```

(c)
```
> length(which((eurocities.df$Athens == "Paris"
+        | eurocities.df$Barcelona == "Paris")
+       & eurocities.df$X3313 <= 300))
[1] 2
```

(d)
```
> num = which(eurocities.df$Athens == "Copenhagen"
+             | eurocities.df$Barcelona == "Copenhagen")
>
> copenhargen.df = data.frame(eurocities.df[num,])
> d_sort = sort(copenhargen.df$X3313)
> d_min3 = d_sort[1:3]
> d_min3
[1] 269 460 650
>
> cities3 = character(3)
> for (i in 1:3) {
+   min_No = which(copenhargen.df$X3313 == d_min3[i])
+   cities3[i] = copenhargen.df$Barcelona[min_No]
+   if (cities3[i] == "Copenhagen")
+     cities3[i] = copenhargen.df$Athens[min_No]
+ }
> cities3
[1] "Hook of Holland" "Hamburg"         "Stockholm"
```

(e)
```
> num_start = which((eurocities.df$Athens == "Rome"
+                   | eurocities.df$Athens == "Stockholm")
+                   & (eurocities.df$Barcelona != "Rome"
+                      & eurocities.df$Barcelona != "Stockholm"))
> num_end = which((eurocities.df$Barcelona == "Rome"
+                 | eurocities.df$Barcelona == "Stockholm")
+                 & (eurocities.df$Athens != "Rome"
+                    & eurocities.df$Athens != "Stockholm"))
>
> start.df = data.frame(eurocities.df[num_start,])
> end.df = data.frame(eurocities.df[num_end,])
> ## col is the column that contain the other same cities to the 2 cities
> findcity.min = function(df, col, cityvec) {
+   rows = rownames(df)
+   rownum = nrow(df)
+   colnum = ncol(df)
+   distances = numeric(rownum %/% 2)
+   cityvec.sort = sort(cityvec)
+   for (i in 2:rownum) {
+     # find the row number of two distances from 1 same city
+     twodt.No = rows[df[, col] == cityvec.sort[i]]
+     # add the two distance from the same sity
```

```
+     distances[i %/% 2] = df[twodt.No[1], colnum] + df[twodt.No[2], colnum]
+   }
+
+   mindt = min(distances)
+   distance_num = 1:length(distances)
+   min_num = distance_num[distances == mindt]
+
+   min_city = cityvec.sort[min_num * 2]
+   list(city = min_city, min = mindt)
+ }
>
> lst1 = findcity.min(start.df, 2, start.df$Barcelona)
> lst1
$city
[1] "Vienna"

$min
[1] 3314

> lst2 = findcity.min(end.df, 1, end.df$Athens)
> lst2
$city
[1] "Cologne"   "Gibraltar" "Milan"

$min
[1] 2700

>
> if (lst1$min < lst2$min) lst1$city else lst2$city
[1] "Cologne"   "Gibraltar" "Milan"
```

3. The answers are below:

(a)
```
> hemite1 = function(xseq, n) {
+   for (x in xseq) {
+     res = 0
+     for (m in 0:floor(n / 2)) {
+       res = res + (-1)^m / (factorial(m) * factorial(n - 2 * m))
* x^(n - 2 * m) / 2^m
+     }
+     print(factorial(n) * res)
+   }
+ }
>
> hemite1(seq(-2, 2, by = .2), 5)
[1] 18
[1] 12.42432
```

```
[1] 6.47424
[1] 1.06176
[1] -3.20832
[1] -6
[1] -7.20768
[1] -6.91776
[1] -5.37024
[1] -2.92032
[1] 0
[1] 2.92032
[1] 5.37024
[1] 6.91776
[1] 7.20768
[1] 6
[1] 3.20832
[1] -1.06176
[1] -6.47424
[1] -12.42432
[1] -18
```

(b)
```
> hemite2 = function(xseq, n) {
+   for (x in xseq) {
+     upval = floor(n / 2)
+     mrange = 0:upval
+     res = factorial(n) * sum((-1)^mrange
+                              / (factorial(mrange) * factorial(n - 2 * mrange))
+                              * x^(n - 2 * mrange) / 2^mrange)
+     print(res)
+   }
+ }
>
> hemite2(seq(-2, 2, by = .2), 5)
[1] 18
[1] 12.42432
[1] 6.47424
[1] 1.06176
[1] -3.20832
[1] -6
[1] -7.20768
[1] -6.91776
[1] -5.37024
[1] -2.92032
[1] 0
[1] 2.92032
[1] 5.37024
[1] 6.91776
[1] 7.20768
```

```
[1] 6
[1] 3.20832
[1] -1.06176
[1] -6.47424
[1] -12.42432
[1] -18
```

(c)
```
> hemite3.1 = function(x, m, n) {
+   (-1)^m / (factorial(m) * factorial(n - 2 * m)) * x^(n - 2 * m) / 2^m
+ }
>
> hemite3.2 = function(x, n) {
+   factorial(n) * sum(x)
+ }
>
> hemite.seq = function(x, n) {
+   m = 0:floor(n / 2)
+   hemite3.3 = function(x, m) hemite3.1(x, m, n)
+   hemite3.4 = function(x) hemite3.2(x, n)
+
+   o = outer(x, m, hemite3.3)
+   apply(o, 1, hemite3.4)
+ }
> x = seq(-2, 2, by = .2)
> hemite.seq(x, 5)
 [1]  18.00000  12.42432   6.47424   1.06176  -3.20832  -6.00000  -7.20768
  -6.91776  -5.37024  -2.92032
[11]   0.00000   2.92032   5.37024   6.91776   7.20768   6.00000   3.20832
-1.06176  -6.47424 -12.42432
[21] -18.00000
```

4. The answers are below:

```
> hemite.coef = function(n) {
+   if (n > 0) {
+     x = seq(length = n + 1, from = 1)
+
+     # contruct the value matrix
+     A = outer(x, 0:n, "^")
+     H = hemite.seq(x, n) # use functions in question 3
+
+     # using round when x are not intergers
+     round(solve(A, H))
+   }
+   else 1
+ }
>
> hemite = function(n) {
```

```
+   coef.matrix = matrix(0, nrow = n + 1, ncol = n + 1,
+                        dimnames = list(paste("H", 0:n, sep = ""),
+                                        paste("x^", 0:n, sep = "")))
+   for (i in 0:n) {
+     coef.matrix[i + 1,] = c(hemite.coef(i), rep(0, n - i))
+   }
+   coef.matrix
+ }
>
> hemite(0)
   x^0
H0   1
> hemite(1)
   x^0 x^1
H0   1   0
H1   0   1
> hemite(3)
   x^0 x^1 x^2 x^3
H0   1   0   0   0
H1   0   1   0   0
H2  -1   0   1   0
H3   0  -3   0   1
> hemite(10)
     x^0  x^1  x^2   x^3   x^4 x^5 x^6 x^7 x^8 x^9 x^10
H0     1    0    0     0     0   0   0   0   0   0    0
H1     0    1    0     0     0   0   0   0   0   0    0
H2    -1    0    1     0     0   0   0   0   0   0    0
H3     0   -3    0     1     0   0   0   0   0   0    0
H4     3    0   -6     0     1   0   0   0   0   0    0
H5     0   15    0   -10     0   1   0   0   0   0    0
H6   -15    0   45     0   -15   0   1   0   0   0    0
H7     0 -105    0   105     0 -21   0   1   0   0    0
H8   105    0 -420     0   210   0 -28   0   1   0    0
H9     0  945    0 -1260     0 378   0 -36   0   1    0
H10 -945    0 4725     0 -3150   0 630   0 -45   0    1
```

5. The answers are below:

   (a)
```
> f = function(x) sin(x) + exp((-1) * x / 10)
>
> root = function(f, interval) {
+   a = interval[1]
+   b = interval[2]
+
+   i = 1
+   while ((f(a) * f(b)) < 0
+          && abs(f(a) - f(b)) > 1e-08
```

```
+          && i < 1000) {
+     c = (a + b) / 2
+     tmp = f(c)
+     if(sign(tmp) == sign(f(a)))
+       a = c
+     else
+       b = c
+
+     i = i + 1
+   }
+
+   if (i < 1000) {
+     if ((f(a) * f(b)) < 0) a
+     else {
+       print("couldn't find root within this interval!")
+       NA
+     }
+   }
+   else {
+     print("no root can be found!")
+     NA
+   }
+ }
>
> root(f, c(0, 5))
[1] 3.886584
```

(b)
```
> allroot = function(f, interval) {
+   a = interval[1]
+   b = interval[2]
+   points = seq(a, b, length.out = 1000)
+   con = f(points) >= 0
+   positives = sort(points[con])
+   negatives = sort(points[!con])
+
+   #print(positives)
+   #print(negatives)
+
+   if (positives[1] < negatives[1]) {
+     low = positives[1]
+     high = negatives[1]
+     con = TRUE
+   }
+   else {
+     low = negatives[1]
+     high = positives[1]
+     con = FALSE
```

```
+   }
+
+   roots = numeric(1000)
+
+   i = 1
+   j = 1
+   up1 = length(positives)
+   up2 = length(negatives)
+   while(low < high) {
+     if (con) {
+       lows = positives[i:up1]
+     }
+     else {
+       lows = negatives[i:up2]
+     }
+
+     nums = which(lows < high)
+     len = length(nums)
+     low = lows[nums[len]]
+     #print(c(low, high))
+
+     # save each root
+     roots[j] = root(f, c(low, high))
+     j = j + 1
+
+     low = high
+     if (nums[len] == length(lows))
+       high = lows[nums[len]]
+     else
+       high = lows[nums[len] + 1]
+
+     if (con)
+       i = which(negatives == low)
+     else
+       i = which(positives == low)
+
+     con = !con
+   }
+
+   roots[1:(j - 1)]
+ }
>
> allroot(f, c(0, 50))
 [1]  3.886584  5.680796  9.809140 12.268792 15.913058
 18.694733 22.101059 25.050983 28.333185 31.372511
[11] 34.588989 37.676002 40.857516 43.969983 47.132865
```