# HW5

2024-09-29

## Homework 5

Due October 15

### Q1

Q:

Use the state.x77 data matrix and the tapply(), and separately, the aggregate() function to obtain

a.

The mean per capita income of the states in each of the four regions defined by the factor state.region,

b.

The maximum illiteracy rates for states in each of the nine divisions defined by the factor state.division,

c.

The number of states in each region,

d.

The names of the states in each division,

e.

The median high school graduation rates for groups of states defined by combinations of the factors state.region and state.size.

A:

a.

```
incomes <- state.x77[,2]
incomesf <- factor(state.region)

tapplyMean <- tapply(X = incomes,
                     INDEX = incomesf,
                     FUN = mean)
tapplyMean
```

```
##     Northeast         South North Central          West
##      4570.222      4011.938      4611.083      4702.615
```

```
aggCount <- aggregate(x = incomes,
          by = list(incomesf),
          FUN = mean)

aggCount
```

```
##         Group.1        x
## 1     Northeast 4570.222
## 2         South 4011.938
## 3 North Central 4611.083
## 4          West 4702.615
```

b.

```
illit <- state.x77[,3]
illitf <- factor(state.division)

tapplyMax <- tapply(X = illit,
                    INDEX = illitf,
                    FUN = max)
tapplyMax
```

```
##        New England    Middle Atlantic     South Atlantic East South Central
##                1.3                1.4                2.3                2.4
## West South Central East North Central West North Central           Mountain
##                2.8                0.9                0.8                2.2
##            Pacific
##                1.9
```

```
aggCount <- aggregate(x = illit,
          by = list(illitf),
          FUN = max)

aggCount
```

```
##              Group.1   x
## 1        New England 1.3
## 2    Middle Atlantic 1.4
## 3     South Atlantic 2.3
## 4 East South Central 2.4
## 5 West South Central 2.8
## 6 East North Central 0.9
## 7 West North Central 0.8
## 8           Mountain 2.2
## 9            Pacific 1.9
```

c.

```r
states <- state.x77[,0]
statesf <- factor(state.region)

tapplyCount <- tapply(X = statesf,
                      INDEX = state.region,
                      FUN = length)
tapplyCount
```

```
##     Northeast         South North Central          West
##             9            16            12            13
```

```r
aggCount <- aggregate(x = state.region,
          by = list(statesf),
          FUN = length)

aggCount
```

```
##         Group.1  x
## 1     Northeast  9
## 2         South 16
## 3 North Central 12
## 4          West 13
```

d.

```r
# states <- state.x77[,0]
statesf <- factor(state.division)

tapplyNames <- tapply(X = state.name,
                      INDEX = state.division,
                      FUN = function(x) {x})
tapplyNames
```

```
## $`New England`
## [1] "Connecticut"   "Maine"         "Massachusetts" "New Hampshire"
## [5] "Rhode Island"  "Vermont"
##
## $`Middle Atlantic`
## [1] "New Jersey"   "New York"     "Pennsylvania"
##
## $`South Atlantic`
## [1] "Delaware"       "Florida"        "Georgia"        "Maryland"
## [5] "North Carolina" "South Carolina" "Virginia"       "West Virginia"
##
## $`East South Central`
## [1] "Alabama"     "Kentucky"    "Mississippi" "Tennessee"
##
## $`West South Central`
## [1] "Arkansas"  "Louisiana" "Oklahoma"  "Texas"
##
## $`East North Central`
## [1] "Illinois"  "Indiana"   "Michigan"  "Ohio"      "Wisconsin"
##
## $`West North Central`
## [1] "Iowa"         "Kansas"       "Minnesota"    "Missouri"     "Nebraska"
## [6] "North Dakota" "South Dakota"
##
## $Mountain
## [1] "Arizona"    "Colorado"   "Idaho"      "Montana"    "Nevada"
## [6] "New Mexico" "Utah"       "Wyoming"
##
## $Pacific
## [1] "Alaska"     "California" "Hawaii"     "Oregon"     "Washington"
```

```
aggCount <- aggregate(x = state.name,
          by = list(statesf),
          FUN = function(x) {paste(x, collapse = ", ")})

aggCount
```

```
##              Group.1
## 1        New England
## 2    Middle Atlantic
## 3     South Atlantic
## 4 East South Central
## 5 West South Central
## 6 East North Central
## 7 West North Central
## 8           Mountain
## 9            Pacific
##                                                                                  x
## 1                       Connecticut, Maine, Massachusetts, New Hampshire, Rhode Island, Vermont
## 2                                                     New Jersey, New York, Pennsylvania
## 3 Delaware, Florida, Georgia, Maryland, North Carolina, South Carolina, Virginia, West Virginia
## 4                                              Alabama, Kentucky, Mississippi, Tennessee
## 5                                                   Arkansas, Louisiana, Oklahoma, Texas
## 6                                           Illinois, Indiana, Michigan, Ohio, Wisconsin
## 7                       Iowa, Kansas, Minnesota, Missouri, Nebraska, North Dakota, South Dakota
## 8                          Arizona, Colorado, Idaho, Montana, Nevada, New Mexico, Utah, Wyoming
## 9                                           Alaska, California, Hawaii, Oregon, Washington
```

e.

```
state.size <- cut(x = state.x77[, "Population"],
                  breaks = c(0, 2000, 10000, Inf),
                  labels = c("Small", "Medium", "Large")
                  )
```

```
hsRates <- state.x77[,6]
df <- cbind.data.frame(hsRates, state.region, state.size)

tapplyMed <- tapply(X = hsRates,
                    INDEX = list(state.region, state.size),
                    FUN = median)

tapplyMed
```

```
##                Small Medium Large
## Northeast       55.9  56.00 51.45
## South           48.1  41.30 47.40
## North Central   53.3  54.50 52.90
## West            62.4  61.75 62.60
```

```
aggMed <- aggregate(x = hsRates ~ state.region + state.size,
                    data = df,
                    FUN = median,
                    na.rm = T)
aggMed
```

```
##     state.region state.size hsRates
## 1      Northeast      Small   55.90
## 2          South      Small   48.10
## 3  North Central      Small   53.30
## 4           West      Small   62.40
## 5      Northeast     Medium   56.00
## 6          South     Medium   41.30
## 7  North Central     Medium   54.50
## 8           West     Medium   61.75
## 9      Northeast      Large   51.45
## 10         South      Large   47.40
## 11 North Central      Large   52.90
## 12          West      Large   62.60
```

## Q2

Q:

For a sample $x_1, x_2, ..., x_n$, the MAD estimator of scale is defined as $1.4826 \cdot median\{|x_i - \bar{x}|\}$ were $\bar{x} = \text{median}\{x_i\}$

Use the matrix object mtcars to compute the MAD estimator of scale for the columns in mtcars

a.

Using the apply() function with the mad() function

b.

By calculating it directly from the definition (i.e., not using the mad() function). You may use the apply() and the sweep() functions but avoid using any loops

A:

a.

```r
data(mtcars)

mad <- function(x) {
  median <- median(x, na.rm = TRUE)
  mad <- median(abs(x - median), na.rm = TRUE)
  1.4826 * mad
}

madDf <- apply(X = mtcars,
                     MARGIN = 2,
                     FUN = mad)

madDf
```

```
##         mpg         cyl        disp          hp        drat          wt
##   5.4114900   2.9652000 140.4763500  77.0952000   0.7042350   0.7672455
##        qsec          vs          am        gear        carb
##   1.4158830   0.0000000   0.0000000   1.4826000   1.4826000
```

b.

```r
madDirect <- function(x) {
  median <- median(x, na.rm = TRUE)
  deviations <- abs(sweep(x = as.matrix(x),
                          MARGIN = 2,
                          STATS = median)
                   )
  mad <- median(deviations, na.rm = TRUE)
  1.4826 * mad
}

madDdf <- apply(X = mtcars,
              MARGIN = 2,
              FUN = madDirect)

madDdf
```

```
##         mpg         cyl        disp          hp        drat          wt 
##   5.4114900   2.9652000 140.4763500  77.0952000   0.7042350   0.7672455 
##        qsec          vs          am        gear        carb 
##   1.4158830   0.0000000   0.0000000   1.4826000   1.4826000
```

## Q3

Q:

Let $h(x,n) = 1 + x + x^2 + ... + x^n = \sum_{i=0}^{\infty} x^i$ Answer the following questiuons

a.

Write code to do the above in a for loop.

b.

Rewrite the code that you wrote to use a while loop.

c.

For each of the $x = 0.3, 1.01$ evaluate the performance for $n = 500, 5000$ in terms of time taken by the software. To do so, wrap the code around the R function system.time() with the same code as above inside the parentheses. Report the values returned in the output as per the user time field.

d.

Compare the above with results obtained avoiding loops.

A:

a.

```r
forfunc <- function(x, n) {
  result <- 0
  for (i in 0:n) {
    result <- result + x^i
  }
  result
}
```

b.

```r
whilefunc <- function(x, n) {
  result <- 0
  i <- 0
  while (i <= n) {
    result <- result + x^i
    i <- i + 1
  }
  result
}
```

c.

$x = 0.3, 1.01$ evaluate the performance for $n = 500, 5000$

```
# because life
set.seed(42)
```

```
xVal <- 0.3
nVal <- 500

system.time(expr =
"result <- 0
  for (i in 0:nVal) {
    result <- result + xVal^i
  }
result"
)
```

```
##    user  system elapsed
##       0       0       0
```

```
system.time(expr =
"result <- 0
  i <- 0
  while (i <= nVal) {
    result <- result + xVal^i
    i <- i + 1
  }
result"
)
```

```
##    user  system elapsed
##       0       0       0
```

```
xVal <- 0.3
nVal <- 500

system.time(expr =
"forRes <- forfunc(xVal, nVal)
forRes"
)
```

```
##    user  system elapsed
##       0       0       0
```

```
system.time(
"whielRes <- whilefunc(xVal, xVal)
whielRes"
)
```

```
##    user  system elapsed
##       0       0       0
```

```r
xVal <- 0.3
nVal <- 5000

system.time(expr =
"forRes <- forfunc(xVal, nVal)
forRes"
)
```

```
##    user  system elapsed
##       0       0       0
```

```r
system.time(
"whielRes <- whilefunc(xVal, xVal)
whielRes"
)
```

```
##    user  system elapsed
##       0       0       0
```

```r
xVal <- 1.01
nVal <- 500

system.time(expr =
"forRes <- forfunc(xVal, nVal)
forRes"
)
```

```
##    user  system elapsed
##       0       0       0
```

```r
system.time(
"whielRes <- whilefunc(xVal, xVal)
whielRes"
)
```

```
##    user  system elapsed
##       0       0       0
```

```r
xVal <- 1.01
nVal <- 5000

system.time(expr =
"forRes <- forfunc(xVal, nVal)
forRes"
)
```

```
##    user  system elapsed
##       0       0       0
```

```r
system.time(
"whielRes <- whilefunc(xVal, xVal)
whielRes"
)
```

```
##    user  system elapsed
##       0       0       0
```

```r
xVal <- 0.3
nVal <- 500

t1 <- Sys.time()
forRes <- forfunc(xVal, nVal)
forRes
```

```
## [1] 1.428571
```

```r
t2 <- Sys.time()
t2 - t1
```

```
## Time difference of 0.003913879 secs
```

```r
t1 <- Sys.time()
whileRes <- whilefunc(xVal, xVal)
whileRes
```

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

```r
t2 <- Sys.time()
t2 - t1
```

```
## Time difference of 0.005202055 secs
```

```r
xVal <- 0.3
nVal <- 5000

t1 <- Sys.time()
forRes <- forfunc(xVal, nVal)
forRes
```

```
## [1] 1.428571
```

```r
t2 <- Sys.time()
t2 - t1
```

```
## Time difference of 0.002883911 secs
```

```
t1 <- Sys.time()
whileRes <- whilefunc(xVal, xVal)
whileRes
```

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

```
t2 <- Sys.time()
t2 - t1
```

```
## Time difference of 0.002098799 secs
```

```
xVal <- 1.01
nVal <- 500

t1 <- Sys.time()
forRes <- forfunc(xVal, nVal)
forRes
```

```
## [1] 14522.05
```

```
t2 <- Sys.time()
t2 - t1
```

```
## Time difference of 0.002146959 secs
```

```
t1 <- Sys.time()
whileRes <- whilefunc(xVal, xVal)
whileRes
```

```
## [1] 2.01
```

```
t2 <- Sys.time()
t2 - t1
```

```
## Time difference of 0.002203941 secs
```

```
xVal <- 1.01
nVal <- 5000

t1 <- Sys.time()
forRes <- forfunc(xVal, nVal)
forRes
```

```
## [1] 4.084983e+23
```

```
t2 <- Sys.time()
t2 - t1
```

```
## Time difference of 0.003015995 secs
```

```r
t1 <- Sys.time()
whileRes <- whilefunc(xVal, xVal)
whileRes
```

```
## [1] 2.01
```

```r
t2 <- Sys.time()
t2 - t1
```

```
## Time difference of 0.002220154 secs
```

```r
xVal <- 1.01
nVal <- 5000

t1 <- Sys.time()
result <- 0
  for (i in 0:nVal) {
    result <- result + xVal^i
  }
result
```

```
## [1] 4.084983e+23
```

```r
t2 <- Sys.time()
t2 - t1
```

```
## Time difference of 0.004695177 secs
```

```r
t1 <- Sys.time()
result <- 0
  i <- 0
  while (i <= nVal) {
    result <- result + xVal^i
    i <- i + 1
  }
result
```

```
## [1] 4.084983e+23
```

```r
t2 <- Sys.time()
t2 - t1
```

```
## Time difference of 0.005520105 secs
```

d.

The only noticeable difference I found was when using Sys.time() instead of system.time, likely due to rounding on part of the latter and precision on part of the former. Comparing across the different values of x, n, even in fractions of a second we tend to observe similar computation times. As the results are very similar across values of x and n, I believe this is more indicative of a lack of evidence that one is more efficient than the other (not an indication they are equally efficient, which apriori I do not believe).

## Q4

Q:

The Lotka-Volterra model for a predator-prey system assumes that x(t) is the number of prey animals at the start of year t and that y(t) is the number of predators at the start of year t. Then the number of prey animals and predators at the end of the following year is given by:

$$x(t+1) = x(t) + b_x x(t) - d_x x(t) y(t)$$

$$y(t+1) = y(t) + b_y d_x x(t) y(t) - d_y y(t)$$

Where $b_x$, $b_y$, $d_x$ and $d_y$ are as follows: - $b_x$ is the natural birth rate of the prey animals in the absence of predation - $d_x$ is the death rate of prey animal in an encounter with the predator. - $d_y$ is the natural death rate of the predators in the absence of food (prey animals) - $b_y$ is the efficiency of turning predated animals into predators

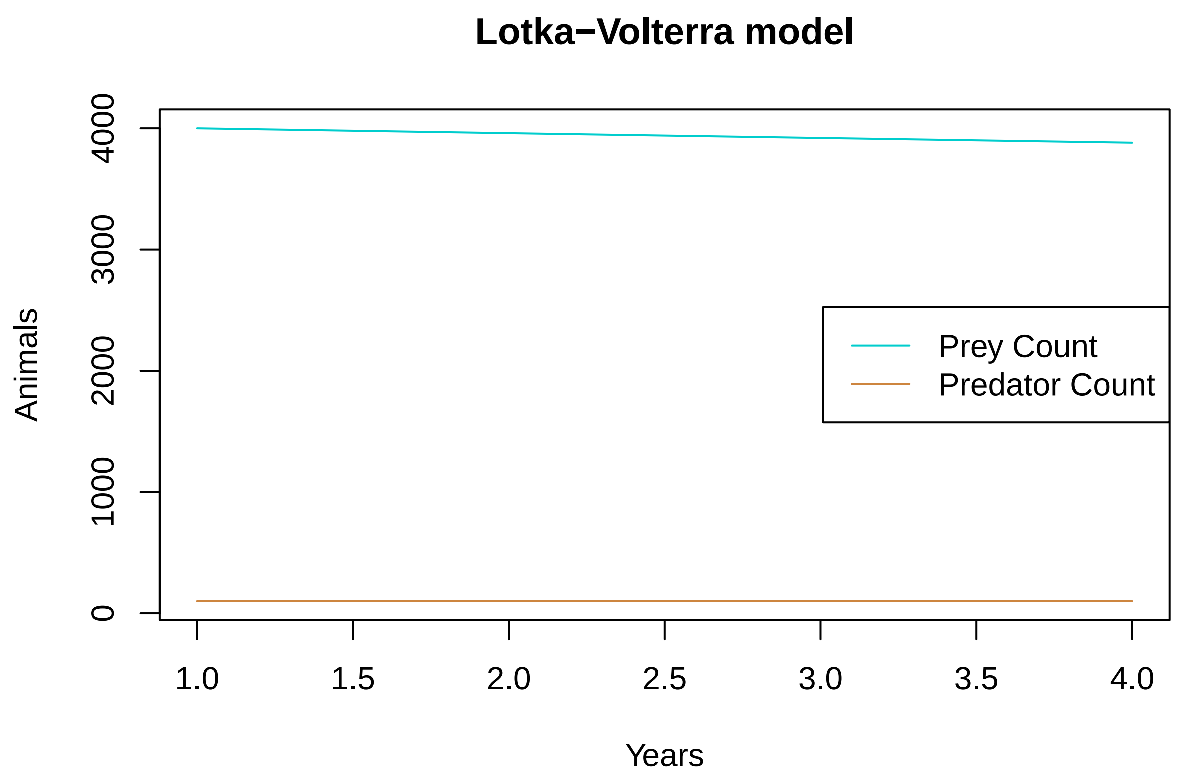
Let $b_x = 0.04$, $d_x = 0.0005$, $b_y = 0.1$, and $d_y = 0.2$.

Suppose that there were 4000 animal of the prey variety at the beginning of the time period. Also, suppose that there were only 100 predators. Write a while loop to show the predator-prey system as long as there are over 3900 prey animals. Save the prey/predator output in a list and plot them using lines on the same plot.

A:

```r
bx <- 0.04
dx <- 0.0005
by <- 0.1
dy <- 0.2

prey <- 4000
predator <- 100

preyList <- c(prey)
predatorList <- c(predator)

while (prey > 3900) {
  newPrey <- prey + bx * prey - dx * prey * predator
  newPredator <- predator + by * dx * prey * predator - dy * predator
  # this is a lot like recursion, which
  # is a lot like recursion, which
  # is a lot like...
  preyList <- c(preyList, newPrey)
  predatorList <- c(predatorList, newPredator)

  prey <- newPrey
  predator <- newPredator
}
```

```r
plot(preyList,
     type="l",
     col="cyan3",
     ylim=c(min(c(preyList, predatorList)), max(c(preyList, predatorList))),
     xlab="Years",
     ylab="Animals",
     main="Lotka-Volterra model")
lines(predatorList,
      col="tan3")
legend("right",
       inset=0,
       legend=c("Prey Count", "Predator Count"),
       col=c("cyan3", "tan3"),
       lty=1)
```



## Q5

Q:

The game of craps is played as follows: first, Player 1 rolls two six-sided die; let x be the sum of the die on the first roll. If x = 7 or x = 11, then Player 1 wins, otherwise the player continues rolling until (s)he gets x again, in which case also Player 1 wins, or until (s)he gets 7 or 11, in which case (s)he loses. Write R code to simulate the game of craps. You can simulate the roll of a fair die using the sample() function in R.

A:

Figure 1: Saw what I did there?

```r
ohCrap <- function() {
  roll1 <- sum(sample(1:6, 2, replace = TRUE))
  if (roll1 == 7 || roll1 == 11) {
    return("Winner Winner, Chicken Dinner. And on the first roll! Lucky duck!")
  } else {
    point <- roll1
    repeat {
      rollN <- sum(sample(1:6, 2, replace = TRUE))
      if (rollN == point) {
        return("Fortune smiles upon you. We have a winner!")
      }
      if (rollN == 7 || rollN == 11) {
        return("You lost! Better luck next time. Remember, the house wins in the long run, and this tim
      }
    }
  }
}

print("Let's play a game!")
```

```
## [1] "Let’s play a game!"
```

```r
ohCrap()
```

```
## [1] "Fortune smiles upon you. We have a winner!"
```

Orange you glad I went with the (jig)Saw joke instead of a poop joke?

## Q6

Q:

Suppose that $(x(t), y(t))$ has polar coordinates given by $(\sqrt{t}, 2\pi t)$. Write code to plot the curve $(x(t), y(t))$ for $t \in [0, 1]$.

A:

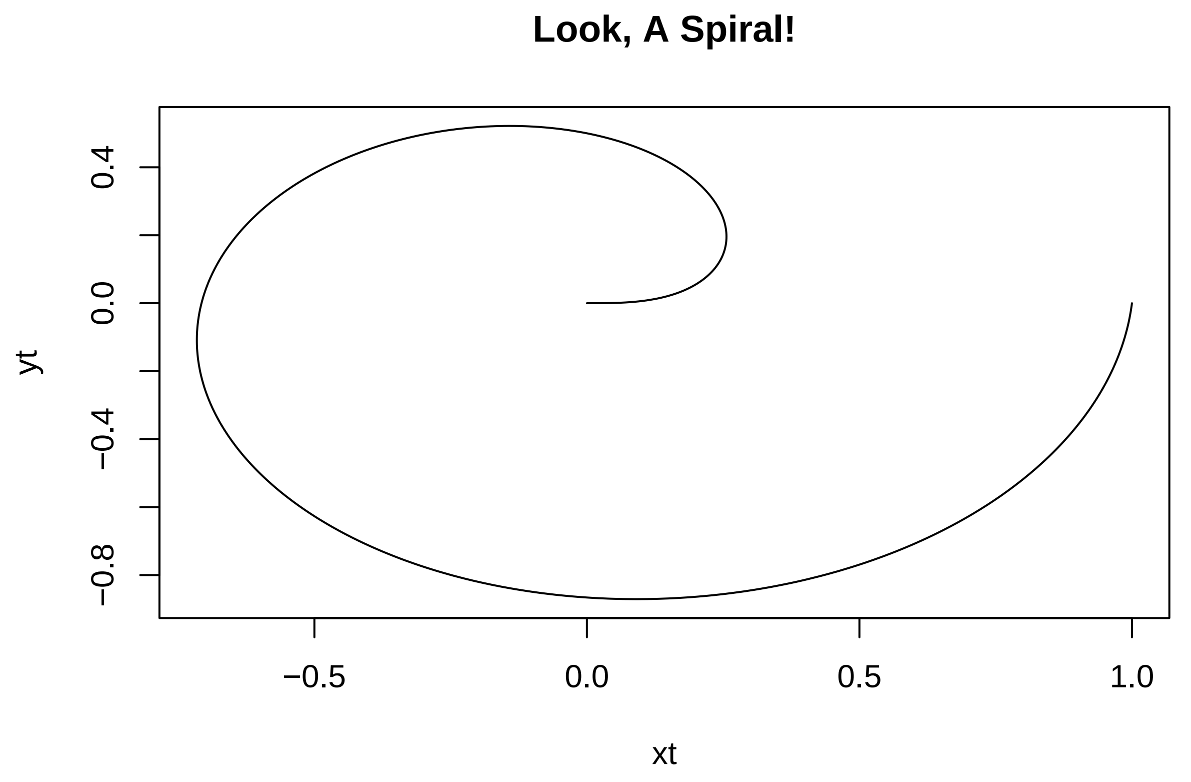
Polar to Cartesian coordinates, r radius x = r cos $\theta$ y = r sin $\theta$

Where $r = \sqrt{t}$ Add $\theta = 2\pi t$

Utilizing this conversion, we then have:

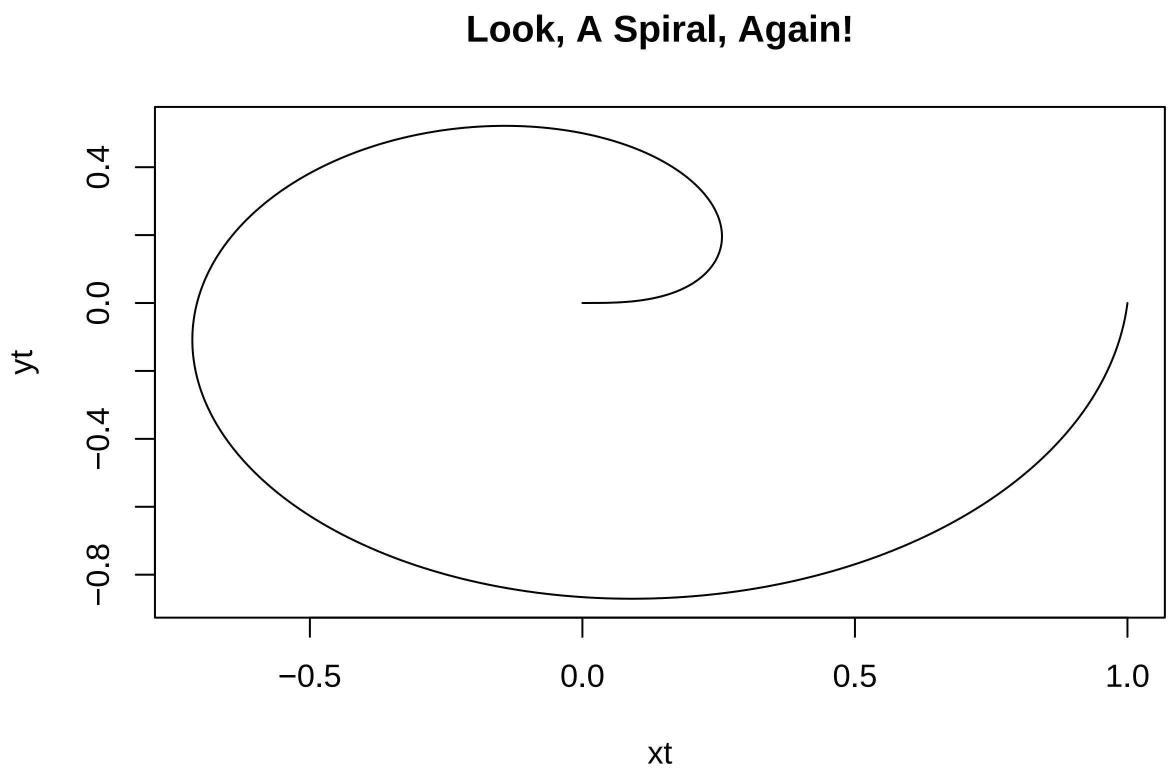
$x(t) = \sqrt{t} \cdot \cos(2\pi t)$ $y(t) = \sqrt{t} \cdot \sin(2\pi t)$

```r
t <- seq(from = 0, to = 1, by = 0.001)

# xt <- sqrt(t)
# yt <- 2 * pi * t

xt <- sqrt(t) * cos(2 * pi * t)
yt <- sqrt(t) * sin(2 * pi * t)

plot(x = xt,
     y = yt,
     type = "l",
     main = "Look, A Spiral!")
```



Here's a for loop of it.

```r
t <- seq(from = 0, to = 1, by = 0.001)

xt <- numeric(length(t))
yt <- numeric(length(t))

for (i in 1:length(t)) {
  xt[i] <- sqrt(t[i]) * cos(2 * pi * t[i])
  yt[i] <- sqrt(t[i]) * sin(2 * pi * t[i])
}

plot(x = xt,
     y = yt,
     type = "l",
     main = "Look, A Spiral, Again!")
```


Look, A Spiral, Again!
yt
xt
0.4
0.0
−0.4
−0.8
−0.5
0.0
0.5
1.0

## Q7

Q:

Consider the following code:

```r
x <- matrix(rnorm(n = 500), ncol = 5)
varx <- var(x)
```

Starting with varx, use two applications of the sweep() function, one dividing each row of the matrix and the other dividing each column, of a covariance matrix to obtain R, the correlation matrix.

A:

```r
# start
x <- matrix(rnorm(n = 500), ncol = 5)
varx <- var(x)

# the problem
std_devs <- sqrt(diag(varx))

rowSweep <- sweep(x = varx,
                  MARGIN = 1,
                  STATS = std_devs,
                  FUN = "/")

# 5x5 matrix
corR <- sweep(x = rowSweep,
              MARGIN = 2,
              STATS = std_devs,
              FUN = "/")
corR
```

```
##              [,1]         [,2]         [,3]        [,4]         [,5]
## [1,]  1.000000000 -0.136597758  0.004551206 -0.17161980 -0.129268609
## [2,] -0.136597758  1.000000000  0.064840380  0.02641392  0.001435963
## [3,]  0.004551206  0.064840380  1.000000000 -0.02191198  0.079069916
## [4,] -0.171619796  0.026413918 -0.021911977  1.00000000  0.078293577
## [5,] -0.129268609  0.001435963  0.079069916  0.07829358  1.000000000
```