Assignment1_YilunDai

```r
setwd("~/Documents/Documents/Fall2018/StatsComputingA/assignment-1-yilundai")
```

Note: All the exercise problems, except for problems from 4.4.1, are from the 2016 edition. Problems under 4.4.1 are from the 2007 edition.

Section 4.1.1

2.

- (1)

```r
Ratios <- numeric(29)
for (i in 2:29)
  Ratios[i-1] <- Fibonacci[i] / Fibonacci[i-1]
```

```r
Ratios
```

```
##  [1] 1.000000 2.000000 1.500000 1.666667 1.600000 1.625000 1.615385
##  [8] 1.619048 1.617647 1.618182 1.617978 1.618056 1.618026 1.618037
## [15] 1.618033 1.618034 1.618034 1.618034 1.618034 1.618034 1.618034
## [22] 1.618034 1.618034 1.618034 1.618034 1.618034 1.618034 1.618034
## [29] 0.000000
```

- Yes, it appears to converge.
- (2)

```r
golden_ratio <- (1 + sqrt(5)) / 2
golden_ratio
```

```
## [1] 1.618034
```

- Yes, the sequence converges to the golden ratio.
- proof
- proof using for loop

```r
difference <- NULL
for (i in 1:29) {
  difference <- c(difference, abs(Ratios[i] - golden_ratio))
}
print(difference)
```

```
##  [1] 6.180340e-01 3.819660e-01 1.180340e-01 4.863268e-02 1.803399e-02
##  [6] 6.966011e-03 2.649373e-03 1.013630e-03 3.869299e-04 1.478294e-04
## [11] 5.646066e-05 2.156681e-05 8.237677e-06 3.146529e-06 1.201865e-06
## [16] 4.590718e-07 1.753498e-07 6.697766e-08 2.558319e-08 9.771908e-09
## [21] 3.732537e-09 1.425702e-09 5.445699e-10 2.080072e-10 7.945178e-11
## [26] 3.034772e-11 1.159184e-11 4.427569e-12 1.618034e+00
```

As we can see, the absolute difference converges to zero

- proof using while loop

```r
i = 3
fib = c(1,1)
```

```r
while(abs(fib[i-1]/fib[i-2]-golden_ratio) > 0.00001){
  fib[i] <- fib[i-2] + fib[i-1]
  i <- i + 1
}
print(fib)
```

```
##  [1]   1   1   2   3   5   8  13  21  34  55  89 144 233 377
```

```r
length(fib)
```

```
## [1] 14
```

3.

- (a)

answer = 15.

check in R:

```r
answer <- 0
for (j in 1:5)
  answer <- answer + j

print(answer)
```

```
## [1] 15
```

- (b)

answer = 1 2 3 4 5

check:

```r
answer <- NULL
for (j in 1:5)
  answer <- c(answer, j)

print(answer)
```

```
## [1] 1 2 3 4 5
```

- (c)

answer = 0 1 2 3 4 5

check:

```r
answer <- 0
for (j in 1:5)
  answer <- c(answer, j)

print(answer)
```

```
## [1] 0 1 2 3 4 5
```

- (d)

answer = 120

check:

```r
answer <- 1
for (j in 1:5)
  answer <- answer * j

print(answer)
```

```
## [1] 120
```

Section 4.1.2

4.

```r
gic_return <- function(P, n) {
  i <- 0
  if(n <= 3) {
    i <- 0.03
  } else {
    i <- 0.05
  }
  I <- P * ((1 + i)^n - 1)
  return(I)
}

gic_return(1, 2)
```

```
## [1] 0.0609
```

5.

```r
mortgage <- function(n, P, open){
  if(open == TRUE){
    i <- 0.05
  } else {
    i <- 0.04
  }
  R <- P * i / (1 - (1 + i)^(-n))
  return(R)

}

mortgage(1, 2, TRUE)
```

```
## [1] 2.1
```

Section 4.1.3

2.

```r
Fibonacci <- c(1, 1)
while (Fibonacci[length(Fibonacci)]+
                 Fibonacci[length(Fibonacci)-1] < 300) {
  Fibonacci <- c(Fibonacci, Fibonacci[length(Fibonacci)] +
                   Fibonacci[length(Fibonacci)-1])

}

Fibonacci
```

```
##  [1]   1   1   2   3   5   8  13  21  34  55  89 144 233
```

4.

```
i <- c(0.006, (1-(1+0.006)^-20)/19)
while (abs(i[length(i)] - (1-(1+i[length(i)])^-20)/19) >= 0.000001){
  i <- c(i, (1-(1+i[length(i)])^-20)/19)
}
i_value <- i[length(i)]
i_value
```

```
## [1] 0.004955135
```

5.

```
i <- c(0.006, (1-(1+0.006)^-20)/19)
num_iter <- 0
while (abs(i[length(i)] - (1-(1+i[length(i)])^-20)/19) >= 0.000001){
  i <- c(i, (1-(1+i[length(i)])^-20)/19)
  num_iter <- num_iter + 1
}
i_value <- i[length(i)]
i_value
```

```
## [1] 0.004955135
```

```
print(num_iter)
```

```
## [1] 72
```

Section 4.1.5

2.

- (a)

If a number is prime, then only 1 and itself are its factors. Therefore, the modulus of this number and another number cannot be zero. The first loop removes every number in the sequence whose smallest prime factor is 2. The following loops remove every number in the sequence whose smallest factor is p. If a number is a prime, then it will not be removed until p equals this number. If a number is not prime, then it will be removed when p is smaller than this number. Since p is the first number of the remaining sieve sequence, p cannot be a non-prime number.

- (b)

Suppose p >= sqrt(n) and p <= n. Suppose p is not prime, so p = ab, where a and b are integers that are greater than 1 and smaller than p. Since n is the greatest in the sequence, p is greater than square of any other number in that sequence. Therefore, the square root of any other number has been removed. If a = b, then a = b = sqrt(n). However, if p = n = sqrt(n) * sqrt(n), then it should have been removed in the loop when p = sqrt(n). If a != b, then p should have been removed in the loop when p = a (if a < b), or when p = b (if a > b). Therefore, p must be prime.

- (c)

```
Eratosthenes <- function(n) {
 if (n >= 2) {
   sieve <- seq(2, n)
   primes <- c()
   repeat{
     if(length(sieve) <= 0) break
     p <- sieve[1]
     primes <- c(primes, p)
     sieve <- sieve[(sieve %% p) != 0]
   }
   return(primes)
 } else {
     stop("Input value of n should be at least 2.") }
}

Eratosthenes(10)
```

```
## [1] 2 3 5 7
```

Section 4.2.1

2.

- (a)

```
compound.interest <- function(P, i.r, n){
  interest <- P * (1 + i.r)^n
  return(interest)
}
```

- (b)

```
Ng <- compound.interest(1000, 0.01, 30)
print(Ng)
```

```
## [1] 1347.849
```

He will have $1347.849

3.

```
find_zero <- function(a, b, f){
  if(f(a) * f(b) < 0){
    while((a != b)){
      mid <- (a+b)/2
      if(f(a) * f(mid) < 0){
        b <- mid
      } else {
        a <- mid
      }
    }
    return(a)
  } else if(f(a) == 0) {
    return(a)
  } else if(f(b) == 0) {
    return(b)
  } else {
    stop("f(a) and f(b) should be of opposite sign.")
  }
}
```

Example:

```
f1 <- function(x){
  y <- x^2 -1
  return(y)
}

find_zero(0, 2, f = f1)
```

```
## [1] 2
```

Section 4.4.1

1.

```
mergesort <- function (x, decrease) {
  len <- length(x)
  if(len < 2){
    result <- x
  } else {
    y <- x[1:(len %/% 2)]
    z <-x[(len%/%2 + 1):len]
    y <- mergesort(y, decrease)
    z <- mergesort(z, decrease)
    result <- c()
    while (min(length(y), length(z)) > 0) {
      if (decrease == TRUE) {
        if (y[1] > z[1]) {
          result <- c(result, y[1])
          y <- y[-1]
        } else {
          result <- c(result, z[1])
          z <- z[-1]
        }

      } else {
        if (y[1] < z[1]) {
          result <- c(result, y[1])
          y <- y[-1]
        } else {
          result <- c(result, z[1])
          z <- z[-1]
        }
      }
    }
  if (length(y) > 0) {
    result <- c(result, y)
  } else {
    result <- c(result, z)
```

```r
  }
  }
  return(result)
}
```

- Testing

```r
mergesort(c(1, 2, 3, 4, 5, 6, 7) , TRUE)
```

```
## [1] 7 6 5 4 3 2 1
```

2.

- (a)

```r
grad <- function(expr){
  result <- attr(expr, "gradient")[1]
  return(result)
}
solvexy <- function(x0, y0, f, g, delta_val){

  fx <- deriv(f, "x")
  fy <- deriv(f, "y")
  gx <- deriv(g, "x")
  gy <- deriv(g, "y")
  xvec <- c(x0)
  yvec <- c(y0)
  x <- x0
  y <- y0
  fn <- eval(fx)[1]
  gn <- eval(gx)[1]
  fxn <- grad(eval(fx))
  fyn <- grad(eval(fy))
  gxn <- grad(eval(gx))
  gyn <- grad(eval(gy))
  dn <- fxn * gyn - fyn * gxn


  while((abs(fn) > delta_val) || (abs(gn) > delta_val)){
    xn <- x - (gyn * fn  - fyn * gn) / dn
    yn <- y - (fxn * gn - gxn * fn) / dn
    x <- xn
    y <- yn
    fn <- eval(fx)[1]
    gn <- eval(gx)[1]
    fxn <- grad(eval(fx))
    fyn <- grad(eval(fy))
    gxn <- grad(eval(gx))
    gyn <- grad(eval(gy))
    dn <- fxn * gyn - fyn * gxn
  }

  return(c(x, y))


}
```

- (b)

```
f <- z ~ x + y
g <- z ~ x^2 + 2 * y^2 - 2
solvexy(-1, 1, f, g, 0.0001)
```

```
## [1] -0.8164966  0.8164966
```

- Find Solution

x + y = 0, so x = - y Then x^2 = y ^2, and the second equation becomes 3x^2 = 2. so x and y should be sqrt(2/3) and -sqrt(2/3). From the R code below, we can see that it is the same as the two answers found with the function.

```
sqrt(2/3)
```

```
## [1] 0.8164966
```

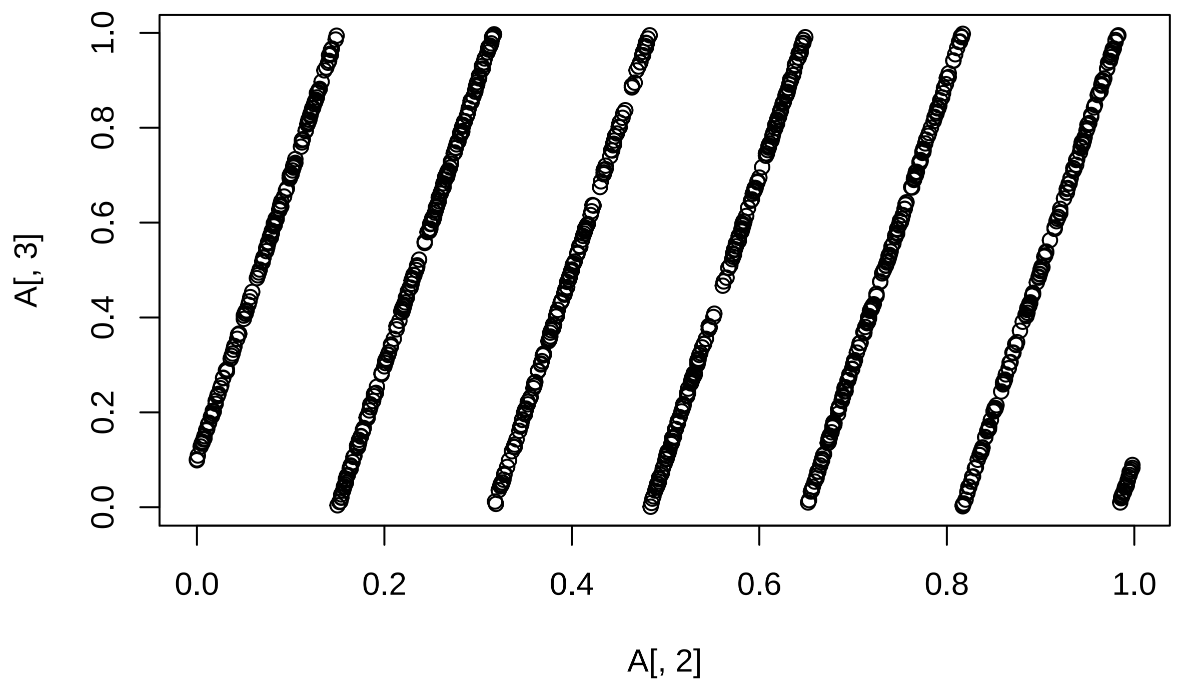
Chapter 4

1.

```
results <- numeric(3000000)
y <- 123
for (i in 1:3000000) {
   y <- (65539*y) %% (2^31)
   results[i] <- round(y / (2^31), 3)
 }
```

```
M <- matrix(results, ncol = 3, byrow = TRUE)
```

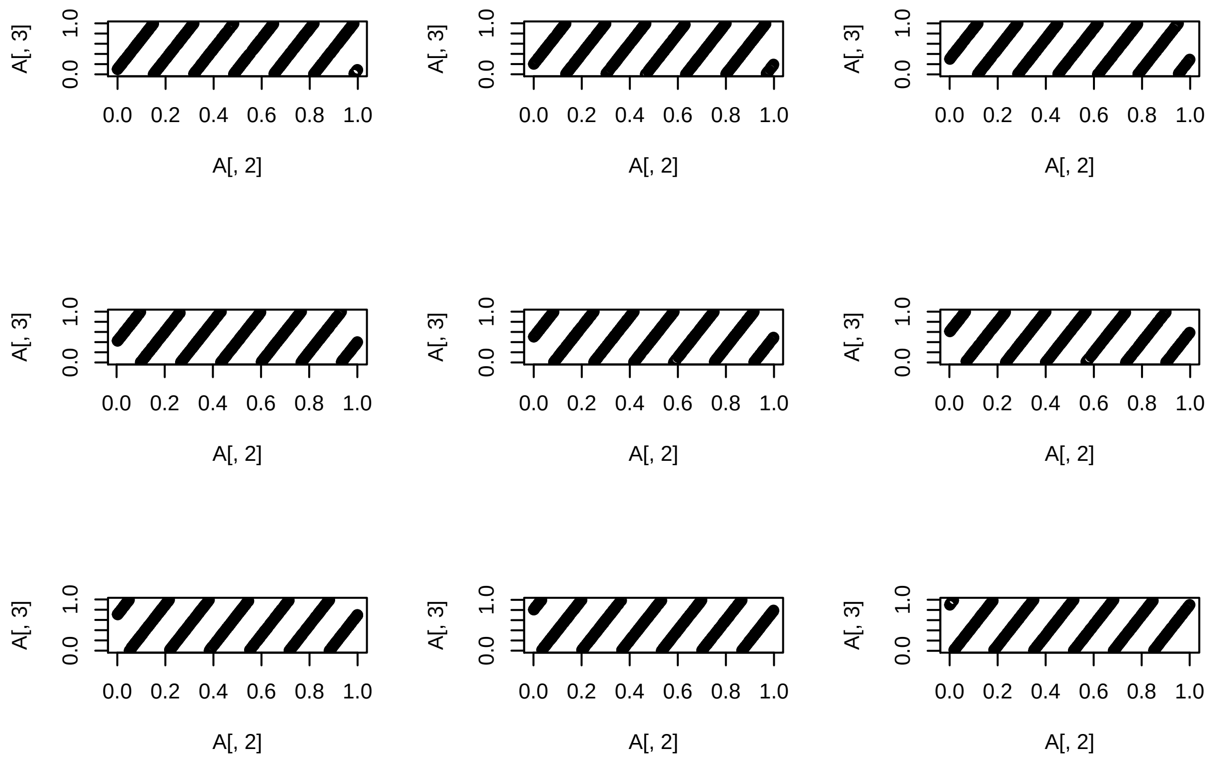
```
x <- 0.1
A = M[M[,1] == x, ]
A2A3 <- plot(A[,2], A[,3])
```



```
A2A3
```

```
## NULL
```

```
xvals <- seq(1,9)
xvals <- xvals/10
par(mfrow=c(3,3))
for(i in 1:9){
  A <- M[M[,1] == xvals[i], ]
  plot(A[,2], A[,3])
}
```



2.

```r
directpoly <- function(x, coefficients){
  n <- length(coefficients)
  P <- 0
  i <- 1
  while(i <= n){
    P <- P + coefficients[i] * (x^(i-1))
    i <- i + 1
  }
  return(P)
}
```

- testing

```r
directpoly(3, seq(1,3))
```

```
## [1] 34
```

3.

- hornerpoly function

```r
hornerpoly <- function(x, coefficients){
  a <- coefficients[length(coefficients)]
  n = length(coefficients) - 1
  while(n >= 1){
    a <- a*x + coefficients[n]
    n <- n - 1
  }
  return(a)

}
```

- testing

```r
hornerpoly(c(2,2,3), seq(1,3))
```

```
## [1] 17 17 34
```