Stat 378 Assignment 1

By Yuhan Li

The homework is based on the 2016 edition. I added the 4.4.1 part of the 2007 edition at the end of the file.

Section 4.1.1

Ex2

a)

```
x=rep(0,30)
y=rep(0,29)
x[1]=1
x[2]=1
for(i in 3:30){
  x[i]=x[i-1]+x[i-2]
}
for(i in 1:29){
  y[i]=x[i+1]/x[i]
}
y
```

```
##  [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] 1.618034
```

The sequence appears to be converging.

b)

```
(1+sqrt(5))/2
```

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

The sequnce converges to this value.

Proof: Suppose the sequence converges to a certain value x_0.
Then, let three consecutive elements of this sequence is $a, b, a+b$.
Since the sequence is convergence, the rate between two consecutive elements stays the same when the sequence comes to infinite,
then $b/a = (a+b)/b = x_0$.
Thus $b^2 = a^2 + ab$, which leads to $b = (1+\sqrt{5})/2 * a$.
Thus the rate is $x_0 = (1+\sqrt{5})/2$.

Ex3

a) The value of answer is 15.

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

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

b) The value of answer is the vector (1,2,3,4,5).

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

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

c) The value of answer is the vector (0,1,2,3,4,5).

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

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

d) The value of answer is 120.

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

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

e) The value of answer is the vector (3,21,23,6,11,15,12,22,30,24,13,29,17,26,27,3)

```
answer <- 3
for (j in 1:15) answer <- c(answer, (7 * answer[j]) %% 31)
answer
```

```
##  [1]  3 21 23  6 11 15 12 22 30 24 13 29 17 26 27  3
```

The next element of this sequence should be 21.

Section 4.1.2

Ex4

```
interest<-function(P,i){
  if (i<=3){I=P*((1+0.04)^i-1)}
  else {I=P*((1+0.05)^i-1)}
  print(I)
}
```

Ex5

```
rate<-function(n,P,open){
  if(open==TRUE){R=P*0.005/(1-(1+0.005)^(-n))}
  else{R=P*0.004/(1-(1+0.004)^(-n))}
  R
}
```

Section 4.1.3

Ex2.

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

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

Ex4

```r
i=0.006
old_i=0
while(abs(old_i-i)>=0.000001){
  new_i=(1-(1+i)^(-20))/19
  old_i=i
  i=new_i
}
i
```

```
## [1] 0.004954139
```

When trying other starting values, the result won't change.

Ex5

```r
i=0.006
old_i=0
j=0
while(abs(old_i-i)>=0.000001){
  new_i=(1-(1+i)^(-20))/19
  old_i=i
  i=new_i
  j=j+1
}
j
```

```
## [1] 74
```

It takes 74 iterations to reach the final result, when the start guessing is 0.006. When trying different starting values, the number of iterations would differ.

Section 4.1.5

Ex2 a)

```r
Eratosthenes <- function(n) {
  # Print prime numbers up to n (based on the sieve of Eratosthenes)
  if (n >= 2) {
    sieve <- seq(2, n)
    primes <- c()
    while (length(sieve) > 0) {
      p <- sieve[1]
      primes <- c(primes, p)
      # add the prime to the sequence
      sieve = sieve[(sieve %% p) != 0]
      # remove all integers in the sequence which can be divided by this prime number
```

```
    }
    return(primes)
  } else {
    stop("Input value of n should be at least 2")
  }
}
```

b)Suppose $n > p > \sqrt{n}$ remaining in the *sieve* is not a prime.
Then there exists primes $m_1, m_2 ... m_k < p$, such that $p = m_1 * m_2 * ... m_k$
In that case, however, p should be removed when the prime m_1 is added to the sequence of prime, according to the process of the algorithm.
Therefore, p shouldn't be a remaining element in the *sieve*, which leads to a contradiction.
Thus, p should be a prime number.

c)

```
Eratosthenes <- function(n) {
 # Print prime numbers up to n (based on the sieve of Eratosthenes)
  if (n >= 2) {
    sieve <- seq(2, n)
    primes <- c()
    while (length(sieve) > 0) {
      p <- sieve[1]
      primes <- c(primes, p)
      sieve = sieve[(sieve %% p) != 0]
      if (p >= sqrt(n)){
        primes = c(primes, sieve);
        break
      }
    }
    return(primes)
  } else {
    stop("Input value of n should be at least 2")
  }
}
#test
Eratosthenes(30)
```

```
##  [1]  2  3  5  7 11 13 17 19 23 29
```

Section 4.2.1

Ex2 a)

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

b)

```
compound.interest(1000,0.01,30)
```

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

Mr.Ng will have $1347.849.

Ex3

```
solution=function(f,a,b,tolerance=10^(-5)){
  if(abs(f(a))<=tolerance){
    return(a)
  }
  if(abs(f(b))<=tolerance){
    return(b)
  }
  else{
     m =(b+a)/2
  if (b-a<=tolerance)
    return (m)
   if (f(a)*f(m)<0){
     return (solution(f,a,m,tolerance))
   }
  else(return (solution(f,m,b,tolerance)))
  }
}
#test
f=function(x){x^2+3*x-4}
solution(f,-2,3)
```

```
## [1] 1.000002
```

Section 4.4.1

Ex1

```
factorial(10)
```

```
## [1] 3628800
```

```
factorial(50)
```

```
## [1] 3.041409e+64
```

```
factorial(100)
```

```
## [1] 9.332622e+157
```

```
factorial(1000)
```

```
## Warning in factorial(1000): value out of range in 'gammafn'
```

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

1000! is so large that causes an error in computation.

Ex2 a)

```
binom.coefficient <- function(n, m) {
  return (factorial(n)/(factorial(m)* factorial(n - m)))
}
```

b)

```
binom.coefficient(4,2)
```

```
## [1] 6
```

```
binom.coefficient(50,20)
```

```
## [1] 4.712921e+13
```

```
binom.coefficient(5000,2000)
```

```
## Warning in factorial(n): value out of range in 'gammafn'

## Warning in factorial(m): value out of range in 'gammafn'

## Warning in factorial(n - m): value out of range in 'gammafn'

## [1] NaN
```

The last value is too large to be obtained.

c)

```
improved.coefficient <- function(n, m) {
  x1=log(1:n)
  x2=log(1:m)
  x3=log(1:(n-m))
  coefficient=exp(sum(x1)-sum(x2)-sum(x3))
  return(coefficient)
}
```

d)

```
improved.coefficient(4,2)
```

```
## [1] 6
```

```
improved.coefficient(50,20)
```

```
## [1] 4.712921e+13
```

```
improved.coefficient(5000,2000)
```

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

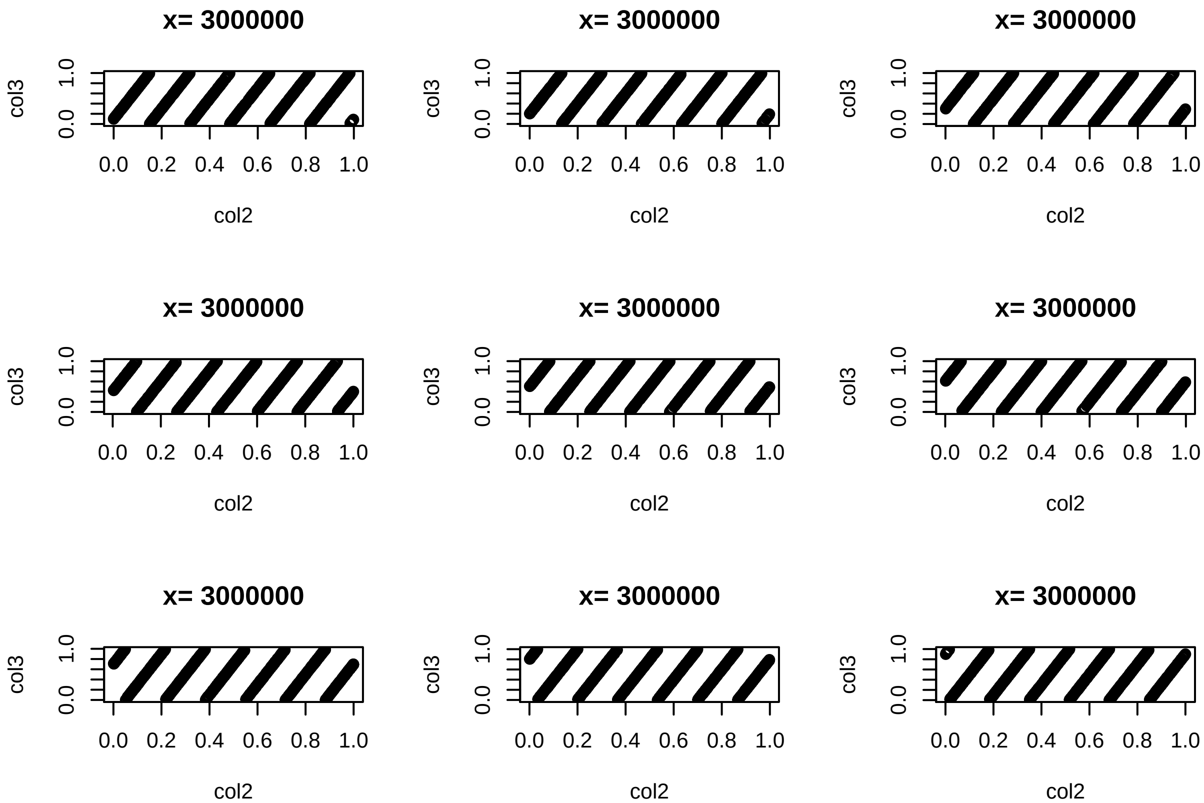
$\binom{5000}{2000}$ is still too large to be calculated.

Chapter Exercises

Ex1

```
results<-numeric(3000000)
x<-123
for (i in 1:3000000){
  x <- (65539*x) %% (2^31)
  results[i]<-x/(2^31)
}
results.round<-round(results,3)
results.matrix<-matrix(results.round,ncol=3,byrow=T)
par(mfrow = c(3,3))
for(j in 1:9){
  x=j/10
  new.rowx <- results.matrix[results.matrix[,1] == x,]
```

```
  plot(new.rowx[,2],new.rowx[,3],xlab = "col2", ylab = "col3",main=paste("x=",i))
}
```



Ex2

```
directpoly=function(C,x){
  value=rep(0,length(C))
  for(i in 1:length(C)){
    value[i]=C[i]*x^(i-1)
  }
 answer=sum(value)
 print(answer)
}
#test
directpoly(c(1,2,3,4),5)
```

```
## [1] 586
```

Ex3

```
hornerpoly=function(C,x){
  answer=numeric(length(x))
  k=0
  for(y in x){
  n=length(C)
  a=c(numeric(n-1),C[n])
  for (i in (n-1):1) {
    a[i]<-a[i+1]*y+C[i]
  }
  k=k+1
```

```r
  answer[k]=a[1]
  }
  return(answer)
}
#test
hornerpoly(c(1,2,3,4),5)
```

```
## [1] 586
```

```r
hornerpoly(c(1,2,3,4),c(1,2,3))
```

```
## [1]  10  49 142
```

Section 4.4.1 (2007 Edition)

Ex1

```r
mergesort <- function(x, decreasing) {
  len <- length(x)
  if (len < 2) result <- x
  else {
    y <- x[1: floor(len / 2)]
    z <- x[floor(len / 2 + 1): len]
    y <- mergesort(y, decreasing)
    z <- mergesort(z, decreasing)
    result <- c()
    while (min(length(y), length(z)) > 0) {
      if (decreasing) {
        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 (length(z) > 0)
    result <- c(result, z)
```

```
  }

  return (result)
}

#test
mergesort(c(3,4,56,7,23),TRUE)
```

```
## [1] 56 23  7  4  3
```

```
mergesort(c(3,4,56,7,23),FALSE)
```

```
## [1]  3  4  7 23 56
```

Ex2

a)

```
Newton=function(f,g,tol=10^(-6),x0,y0){
  h = 10^(-6)
  i = 1
  while(i<= 100){
    f.dx = (f(x0+h,y0)-f(x0,y0))/h
    f.dy = (f(x0,y0+h)-f(x0,y0))/h
    g.dx = (g(x0+h,y0)-g(x0,y0))/h
    g.dy = (g(x0,y0+h)-g(x0,y0))/h
    x1 = x0-(g.dy*f(x0,y0)-f.dy*g(x0,y0))/(f.dx*g.dy - f.dy*g.dx)
    y1 = y0-(f.dx*g(x0,y0)-g.dx*f(x0,y0))/(f.dx*g.dy - f.dy*g.dx)
    i = i+1
    if (abs(x1-x0)<tol && abs(y1-y0)<tol) break
    x0=x1
    y0=y1
  }
  return(c(x1,y1))
}
```

b)

```
f=function(x,y) x^2+2*y^2-2
g=function(x,y) x+y
Newton(f,g,tol=10^(-6),x0=1,y0=1)
```

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

Analytically, solution to this system is $x = \sqrt{\frac{2}{3}}$, $y = -\sqrt{\frac{2}{3}}$; or $x = -\sqrt{\frac{2}{3}}$, $y = \sqrt{\frac{2}{3}}$. The result we gained from the above function is consistent with one of the solution.