STAT37810 Assignment1

Yanfei Zhou (UCID:12207462)

Question 4.1.1

2a)

```
Fibonacci <- numeric(30)
Fibonacci[1] <- Fibonacci[2] <- 1  #assign first two element sin Fibonacci as 1,
for (i in 3:30){
Fibonacci[i] <- Fibonacci[i - 2] + Fibonacci[i - 1]
}
Fibonacci
```

```
##  [1]      1      1      2      3      5      8     13     21     34     55
## [11]     89    144    233    377    610    987   1597   2584   4181   6765
## [21]  10946  17711  28657  46368  75025 121393 196418 317811 514229 832040
```

```
Fib_ratio <-numeric(29)
for ( i in 2: 30){
  Fib_ratio[i] = Fibonacci[i]/Fibonacci[i-1]
}
Fib_ratio
```

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

The sequence appears to be converging to 1.618034

2 b)

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

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

The Fibonacci sequence converges to this ratio:

Let $\psi = \frac{1+\sqrt{5}}{2}$, notice $\psi = 1 + \frac{1}{\psi}$. Let $\{F_n\}$ with n=1,2,3,… be Fibonacci sequence. Let $R_n = \frac{F_{n+1}}{F_n}$, to prove:

$$\lim_{n\to\infty} R_n = \psi = \frac{1+\sqrt{5}}{2}$$

$$\begin{aligned}
R_n &= \frac{F_{n+1}}{F_n} \\
&= \frac{F_n + F_{n-1}}{F_n} \\
&= 1 + \frac{1}{R_{n-1}} \\
|R_n - \psi| &= |1 + \frac{1}{R_{n-1}} - 1 - \frac{1}{\psi}| \\
&= |\frac{\psi - R_{n-1}}{R_{n-1}\psi}| \\
&= \frac{1}{\psi}|\frac{\psi - R_{n-1}}{R_{n-1}}| \\
&\leq \frac{1}{\psi}|\psi - R_{n-1}| \qquad (as\ R_n > 1) \\
&= \frac{1}{\psi}|R_{n-1} - \psi| \\
&\leq (\frac{1}{\psi})^n |R_1 - \psi| \\
\lim_{n\to\infty} |R_n - \psi| &\leq \lim_{n\to\infty} (\frac{1}{\psi})^n |R_1 - \psi| \\
&\to 0 \quad (as\ \frac{1}{\psi} < 1)
\end{aligned}$$

Hence we have

$$\lim_{n\to\infty} |R_n| = \psi$$

Question 4.1.1.

3 a)

answer = 15

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

## [1] 15
```

3 b)

answer = 1 2 3 4 5

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

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

3 c)

answer = 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
```

3 d)

answer = 120

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

## [1] 120
```

3 e)

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 sequence 3 21 23 6 11 15 12 22 30 24 13 29 17 26 27 will repeat when second 3 appears.

Question 4.1.2

4

```
GIC<-function(P,t){
  if (t>3){
    return (P*((1 + 0.05)^t - 1))
  } else {
    return (P*((1 + 0.04)^t - 1))
```

```
  }}
GIC(100,3)

## [1] 12.4864
```

5

```
Mortgage <-function(n,P,open){
  if (open==TRUE){
    i = 0.005
  } else{
    i=0.004
  }
  return (P*i/(1-(1+i)^(-n)))
}
Mortgage(3,100,FALSE)

## [1] 33.60035
```

Question 4.1.3

2

```
i=1
Fibonacci <- c(1, 1)
while (i<30) {
  Fibonacci <- c(Fibonacci, Fibonacci[i-1]+Fibonacci[i])
  i = i+1
}
Fibonacci

##  [1]      1      1      2      3      5      8     13     21     34     55
## [11]     89    144    233    377    610    987   1597   2584   4181   6765
## [21]  10946  17711  28657  46368  75025 121393 196418 317811 514229 832040
```

4

```
i <- 0.006
b = 1
a = 0
while (abs(b - a)>=0.000001) {
  b <- i
  i <- (1 - (1 + i)^(-20)) / 19
  a <- i
}
#trying other starting values
i

## [1] 0.004954139

i <- 0.005
b = 1
```

```
a = 0
while (abs(b - a)>=0.000001) {
  b <- i
  i <- (1 - (1 + i)^(-20)) / 19
  a <- i
}
i

## [1] 0.004953866
```

When using another starting guess, the point estimation becomes slightly different.

5

```
i <- 0.006
b = 1
a = 0
n = 0
while (abs(b - a)>=0.000001) {
  b <- i
  i <- (1 - (1 + i)^(-20)) / 19
  a <- i
  n = n+1}
n

## [1] 74
```

Question 4.1.5

2 a)

```
Eratosthenes<- function(n) {
  #find all the primes between 2 to n
  if (n>=2){
    sieve <- seq(2,n) #create a consecutive integer sequence from 2 up to n
    primes <- c() #create an empty list to store primes later
    while(length(sieve)>0){
      p<-sieve[1] #store the first element remaining in sieve as p
      primes<-c(primes,p) #add p to prime list
      sieve<-sieve[(sieve%%p)!=0] #remove all the elements in remaining sieve
sequence that can be divided by p
    }
    return (primes) #return prime list
  } else{
    stop("Input value of n should be at least 2.") }
}
Eratosthenes(11)

## [1]  2  3  5  7 11
```

2 b)

```r
Eratosthenes<- function(n) {
  if (n>=2){
    sieve <- seq(2,n)
    primes <- c()
    while(length(sieve)>0){
      p<-sieve[1]
      if (p>=sqrt(n)){
        print(c("sieve: ",sieve))
        break}
      primes<-c(primes,p)
      sieve<-sieve[(sieve%%p)!=0]
    }
    return(primes)
  } else{
    stop("Input value of n should be at least 2.") }
}
Eratosthenes(11)

## [1] "sieve: " "5"       "7"       "11"

## [1] 2 3
```

Hence once $p \geq \sqrt{n}$, all the remaining entries in sieve are primes.

2 c)

```r
Eratosthenes<- function(n) {
  if (n>=2){
    sieve <- seq(2,n)
    primes <- c()
    while(length(sieve)>0){
      p<-sieve[1]
      if(p>=sqrt(n)){
        primes<-c(primes,sieve)
        break}
      primes<-c(primes,p)
      sieve<-sieve[(sieve%%p)!=0]
    }
    return (primes)
  } else{
    stop("Input value of n should be at least 2.") }
}
Eratosthenes(11)

## [1]  2  3  5  7 11
```

Question 4.2.1.

2 a)

```r
compound.interest <- function(P, n, ir){
  return (P*((1+ir)^n))}
compound.interest(100,3,0.04)

## [1] 112.4864
```

2 b)

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

## [1] 1347.849
```

He will have $1347.849 in the bank at the end of 30 months.

Question 4.2.1.

3

```r
bisection <- function(f, a, b, nmax=100, tol=1e-8){
  # check if f(a) and f(b) has opposite signs
  if(sign(f(a))==sign(f(b))) {
    stop('Oops! f(a) and f(b) have same signs. Need to change a or b to have f(a), f(b) opposite signs!')
    }
  n = 1
  while(n < nmax){
    # find midpoint
    c <- (a+b)/2
    # if c is the root or the midpoint is below tolerance, output c
    if ((f(c)==0) | (b - a) / 2 < tol) return (c)
    #check the signs of f(a) with f(c), and f(c) with f(b), reassign a or b accordingly as the value of c to be used in the next iteration.
    else if (sign(f(a))!=sign(f(c))) b <- c
    else a <- c
    n = n+1
  }
  #when root is not found after nmax iteration
  print("method failed, try more iterations or check function!")
}
f1 <- function(x) {
  x^3 - 2 * x - 5
}
bisection(f1,a=-5,b=3)

## [1] 2.094551
```

Question 4.4.1. (1)

1

```
mergesort <- function (x, descending) {
  len <-length(x)
  if (len < 2) result <- x
  else {
    y <- x[1:(len / 2)]
    z <- x[(len / 2 + 1):len]
    y <- mergesort(y, descending)
    z <- mergesort(z, descending)
    result <- c()
    if (descending==FALSE){
      while (min(length(y), length(z)) > 0) {
        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)
      return (result)
    } else if (descending==TRUE){
      while (min(length(y), length(z)) > 0) {
        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)
      return (result)
    }
  }
}
mergesort(c(2,6,1,7,5,22),descending = TRUE)

## [1] 7 6 5 2

mergesort(c(2,6,1,7,3,22),descending = FALSE)

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

2 a)

```
newton<-function(f,g, tol=1e-8,nmax=100,x0,y0){ #define function with
argument: function f, function g, minimum tolerance, maximum number of
iterations nmax, set initial values x0, y0
  h = 1e-8
```

```
  i = 1
  while( i <= nmax){
    #find x1, y1 from the definition of derivatives and rules of Newton
method
    df.dx = (f(x0+h,y0)-f(x0,y0))/h
    df.dy = (f(x0,y0+h)-f(x0,y0))/h
    dg.dx = (g(x0+h,y0)-g(x0,y0))/h
    dg.dy = (g(x0,y0+h)-g(x0,y0))/h
    x1 = x0 - (dg.dy*f(x0,y0) - df.dy*g(x0,y0))/(df.dx*dg.dy - df.dy*dg.dx)
    y1 = y0 - (df.dx*g(x0,y0) - dg.dx*f(x0,y0))/(df.dx*dg.dy - df.dy*dg.dx)
    i = i+1
    #if the tolerance between two iteration is smaller than threshold we set,
then break and output x1, y1
    if (abs(x1-x0)<tol && abs(y1-y0)<tol) break
    #if the tolerance is not small enough, continue to next iteration
    x0 = x1
    y0 = y1
  }
  return(c(x1,y1))
}
```

2 b)

```
f<-function(x,y) x^2+2*y^2-2
g<-function(x,y) x+y
newton(f=f,g=g,x0=1,y0=1)

## [1] -0.8164966  0.8164966

newton(f=f,g=g,x0=-1,y0=-1)

## [1]  0.8164966 -0.8164966
```

Analytical solutions:

$$\begin{cases} x+y=0 \\ x^2+2y^2-2=0 \end{cases}$$

From the first equation we have $x=-y$, substitute into second equation, we get $y^2+2y^2-2=0$, which gives $y=\sqrt{\frac{2}{3}}$, one root coincides with the numerical solution we found above.

Chapter 4

1

```
directpoly<-function(c,x){
  P=0
  for (i in 1:length(c)){
    P = P + c[i]*(x**(i-1))
```

```r
  }
  return(P)
}
directpoly(c(1,5,2,6,2),c(1,2))

## [1] 16 99
```

2

```r
hornerpoly<-function(c,x){
  a<-matrix(c,0,nrow=length(c),ncol=length(x))
  for (i in (length(c)-1):1){
    a[i,]=a[i+1,]*x+c[i]
  }
  return(a[1,])
}
hornerpoly(c(1,5,2,6,2),c(1,2))

## [1] 16 99
```

3 a)

```r
system.time(directpoly(c=c(1,-2,2,3,4,6,7),x=seq(-10,10,length=5000000)))

##    user  system elapsed
##   0.926   0.085   1.055

system.time(hornerpoly(c=c(1,-2,2,3,4,6,7),x=seq(-10,10,length=5000000)))

##    user  system elapsed
##   0.613   0.164   0.826
```

Hornerpoly is more efficient when the number of polynomial coefficients is large.

3 b)

```r
system.time(directpoly(c=c(-3,17,2),x=seq(-10,10,length=5000000)))

##    user  system elapsed
##   0.164   0.025   0.208

system.time(hornerpoly(c=c(-3,17,2),x=seq(-10,10,length=5000000)))

##    user  system elapsed
##   0.181   0.071   0.257
```

When the number of polynomial coefficient is small, directpoly turns to be more efficient.