Assignment 1

Jinjie Zhang

October 4, 2017

Section 4.1.1

Exercise 2

(a)

```
Fibonacci=numeric(31)
Fibonacci[1]=Fibonacci[2]=1
for(i in 3:31){
  Fibonacci[i]=Fibonacci[i-1]+Fibonacci[i-2]
}

 ratio=numeric(30)              #construct the sequence of ratios
 for (i in 1:30){
   ratio[i]=Fibonacci[i+1]/Fibonacci[i]
 }
 ratio
```

```
##  [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 1.618034
```

When n is large, the value of f_n/f_{n-1} is close to 1.618034. It seems that the sequence $\{f_n/f_{n-1}\}$ is convergent.

(b)

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

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

```
abs(G_ratio-ratio[30])
```

```
## [1] 6.459278e-13
```

We find that the f_n/f_{n-1} is pretty close to $\frac{1+\sqrt{5}}{2}$ as n goes to infinity. In fact, we can prove this result rigorously, see this.

Exercise 3

(a) The answer should be $\sum_{j=1}^{5} j = 15$.

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

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

(b) The answer should be the collection $(1, 2, 3, 4, 5)$.

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

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

(c) The answer should be the collection $(0, 1, 2, 3, 4, 5)$

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

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

(d) The answer should be the factorial $5! = 120$.

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

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

(e)

```r
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
```

Yes. We can predict successive numbers accroding to its periodicity. The following elements should be $21, 23, 6, 11 \ldots$

Section 4.1.2

Exercise 4

```r
interest=function(P,n){
  #P is the initial investment amount; n is the number of interest periods(years);
  #I is the amount of interest earned
  if (n%%1!=0|n<0){
    print('the year should be a positive integer!')
  }
  else if(n<=3){
    I=P*((1+0.04)^n-1)
    return(I)
  }
  else if(n>3){
    I=P*((1+0.05)^n-1)
    return(I)
  }
}
```

Exercise 5

```r
mortgage=function(P,n,open){
  if (n%%1!=0|n<0){
```

```
    print('the year shoud be a positive integer!')
  }
  else if(open==TRUE){
    i=0.005
    R=P*i/(1-(1+i)^(-n))
    return(R)
  }
  else if(open==FALSE){
    i=0.004
    R=P*i/(1-(1+i)^(-n))
    return(R)
  }
}
```

Section 4.1.3

Exercise 2

```
Fibonacci=c(1,1)
while(sum(tail(Fibonacci,2)) < 300){
  Fibonacci=c(Fibonacci,sum(tail(Fibonacci,n=2)))
}
Fibonacci
```

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

Exercise 4

```
error=1e-6
value_old=0.006
value_new=0
count=0
while(abs(value_new-value_old)>error){
  if(count>0){
    value_old=value_new
  }
  value_new=(1-(1+value_old)^(-20))/19
  count=count+1
}
value_new
```

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

```
count  #the number of iterations
```

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

If the initial value is 0, then the output is 0. If we try other proper initial value (around 0.01), the results may have small difference.

Section 4.1.5

Exercise 2

(a) This algorithm finds all prime number from 2 to n. In each step, we remove the numbers that is divided by the smallest remaining number in sequence 'sieve'. We classify the smallest remaining number as a prime and continue this process until all numbers in 'sieve' are removed.

(b) Suppose that integer $m \in [2, n]$ has decompostion $n = pq$ with $p > \sqrt{n}$. Then we must have $q < \sqrt{n}$ and thus m should be removed before p increases to current value. According to this kind of symmetry, we only need to check the case where $p \leq \sqrt{n}$.

(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)){
   break
 }
 }
 primes <- c(primes, sieve)
 return(primes)
}
  else {
  stop("Input value of n should be at least 2.")
 }
}

#Example
Eratosthenes(30)
```

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

Section 4.2.1

Exercise 2

(a)

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

(b)

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

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

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

Exercise 3

```r
#Find root of f(x)=0 using bisection method
Bisection=function (f, a, b, num, eps = 1e-05)
{
    h = abs(b - a)/num
    i = 0
    j = 0
    a1 = b1 = 0
    while (i <= num) {
        a1 = a + i * h
        b1 = a1 + h
        if (f(a1) == 0) {
            print(a1)
            print(f(a1))
        }
        else if (f(b1) == 0) {
            print(b1)
            print(f(b1))
        }
        else if (f(a1) * f(b1) < 0) {
            repeat {
                if (abs(b1 - a1) < eps)
                  break
                x <- (a1 + b1)/2
                if (f(a1) * f(x) < 0)
                  b1 <- x
                else a1 <- x
            }
            j = j + 1
            print((a1 + b1)/2)
        }
        i = i + 1
    }
    if (j == 0)
        print("finding root is fail")
    else print("finding root is successful")
  }


#Example: let f(x)=x^2-e^x
f=function(x){
  value=x^2-exp(x)
  return(value)
}
Bisection(f,-2,2,20)
```

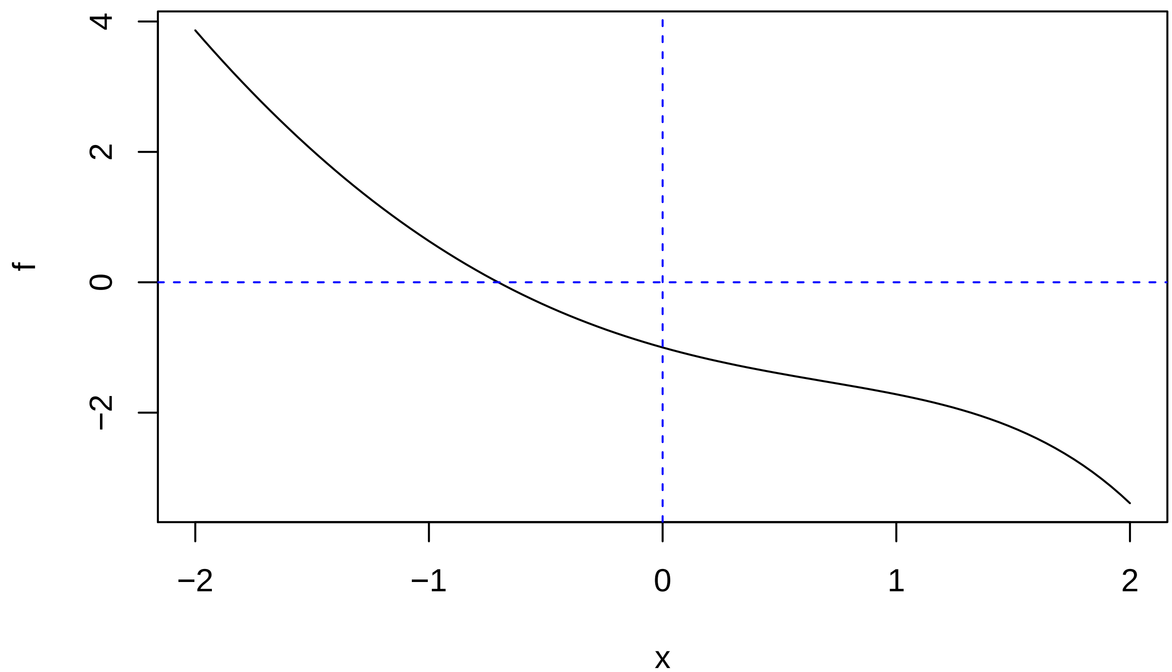
```
## [1] -0.7034698
## [1] "finding root is successful"
```

```r
uniroot(f,c(-1,1))$root #compare our result with the true value
```

```
## [1] -0.7034616
```

We find that our result is pretty close to the true value and we can plot the graph of this function on interval $[-2, 2]$.

```
plot(f,-2,2)
abline(h=0,v=0,col="blue",lty=2)
```



section 4.4.1

Exercises 1

The modified code is as follows.

```
mergesort = function (x, decreasing=FALSE) {
n = length(x)
if(n < 2){
seq = x
}
else{
m = floor(n/2)
y = x[1:m]
z = x[(m+1):n]
y = mergesort(y)
z = mergesort(z)
seq = c()
```

```
while(min(length(y),length(z)) > 0) {
if(y[1] < z[1]) {
seq = c(seq, y[1])
y = y[-1]
} else {
seq = c(seq, z[1])
z = z[-1]
}
}
if(length(y) > 0)
seq = c(seq, y)
else
seq = c(seq, z)
}
if(decreasing==TRUE)
return(rev(seq))
else
return(seq)
}

#We give an example
x=rnorm(5)
x
```

```
## [1]  0.59130795 -0.17574429  0.04662949  2.32628830 -1.79849049
```

```
mergesort(x,FALSE)   #sort in increasing order
```

```
## [1] -1.79849049 -0.17574429  0.04662949  0.59130795  2.32628830
```

```
mergesort(x,TRUE) #sort in decreasing order
```

```
## [1]  2.32628830  0.59130795  0.04662949 -0.17574429 -1.79849049
```

Exercise 2

(a) Use Newton's method to solve equations.

```
Newton_method= function(x_initial, y_initial, f, g, tol=1e-5) {
df = deriv(f, c("x","y"))
dg = deriv(g, c("x","y"))  #find the partial derivatives
x = x_initial
y = y_initial
while(abs(eval(f)) > tol | abs(eval(g)) > tol) {
evdf = eval(df)
evdg = eval(dg)
f = evdf[1]
fx = attr(evdf,"gradient")[1]
fy = attr(evdf,"gradient")[2]
g = evdg[1]
gx = attr(evdg,"gradient")[1]
gy = attr(evdg,"gradient")[2]
d = fx * gy - fy * gx
x = x - (gy*f - fy*g)/d  #get new x and new y
y = y - (fx*g - gx*f)/d
```

```
}
return(c(x,y))

}
```

(b)

```
f = expression(x + y)
g = expression(x^2 + 2*y^2 - 2)
Newton_method(0.5,-0.5,f,g)
```

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

```
Newton_method(-0.5,0.5,f,g)
```

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

In fact, the analytic solution of this equation system is $(\sqrt{\frac{2}{3}}, -\sqrt{\frac{2}{3}})$ and $(-\sqrt{\frac{2}{3}}, \sqrt{\frac{2}{3}})$. Our result is pretty close to the true value.

Chapter 4

Exercises 1

```
directpoly=function(x,coef){
  degree=length(coef)
  value=0
  for(i in 1:degree){
    value=value+coef[i]*x^(i-1)
  }
  return(value)
}
```

Exercise 2

```
hornerpoly = function(x,coef) {
n = length(coef)
a = matrix(0, nrow=length(x), ncol=n)  #construc a matrix
a[,n] = coef[n]
for(i in (n-1):1) {
a[,i] = a[,i+1] * x + coef[i]
}
return(a[,1])
}
#Give an example
hornerpoly(c(2,4,6),c(10,2,3,4,5))
```

```
## [1]  138 1602 7474
```

Exercise 3

(a)

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

```
##    user  system elapsed
##    1.81    0.03    1.87
```

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

```
##    user  system elapsed
##    0.56    0.09    0.65
```

We notice that 'hornerpoly' function computes faster than 'directpoly'.

(b)

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

```
##    user  system elapsed
##    0.39    0.00    0.39
```

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

```
##    user  system elapsed
##    0.20    0.03    0.23
```

Remark: The difference of time consumption is smaller and 'hornerpoly' function is still faster than 'directpoly'.