Assignment 1

Wenjing Xu

October 7, 2018

For loops: Section 4.1.1

Exercise 2

(a)

```
f <- numeric(30)
f[1] <- f[2] <- 1
r <- numeric(30)
r[1] <- r[2] <- 1
for (i in 3:30){
  f[i] <- f[i-1] + f[i-2]
  r[i] <- f[i]/f[i-1]
}
r
```

```
##  [1] 1.000000 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 ratio sequence appears to converge to 1.618034.

(b)

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

```
##  [1] -6.180340e-01 -6.180340e-01  3.819660e-01 -1.180340e-01  4.863268e-02
##  [6] -1.803399e-02  6.966011e-03 -2.649373e-03  1.013630e-03 -3.869299e-04
## [11]  1.478294e-04 -5.646066e-05  2.156681e-05 -8.237677e-06  3.146529e-06
## [16] -1.201865e-06  4.590718e-07 -1.753498e-07  6.697766e-08 -2.558319e-08
## [21]  9.771908e-09 -3.732537e-09  1.425702e-09 -5.445699e-10  2.080072e-10
## [26] -7.945178e-11  3.034772e-11 -1.159184e-11  4.427569e-12 -1.691314e-12
```

Since the difference between the element of the ratio sequence and the value of golden ratio seems to converge to 0, the sequence converges to this golden ratio.

The proof is as follow. $\lim_{n\to+\infty}\frac{F_{n+1}}{F_n}=\lim_{n\to+\infty}\frac{F_{n-1}+F_n}{F_n}=1+\frac{1}{L}$

So there comes $1+\frac{1}{L}=L$,

Solving this and we get $L=\frac{1+\sqrt{5}}{2}$ (the negative solution is omitted because the fraction should be positive)

Exercise 3

(a)

The final value of *answer* should be 15

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

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

(b)

The final value of *answer* should be a sequence (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 final value of *answer* should be a sequence (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 final value of *answer* should be 120

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

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

(e)

The final value of *answer* should be a sequence (3,21,23,6,11,15,12,22,30,24,13,29,17,26,27,3)

```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
```

Since the sequence enters a cycle when the value becomes 3 agin at the 16th element, it can be predicted that the successive elements will loop and they are 21, 23,6. . . .

If Statements: Section 4.1.2

Exercise 4

```r
Interest <- function(P,n){
# Return the amount of interest earned over the term of GIC.
# n takes the value of interest periods in years. SO n should be postive integer.
  if (n<=3) i <- 0.04 else i <- 0.05
  return(P*((1+i)^n-1))
}
```

Exercise 5

```
R <- function(n, P, open){
# Return a monthly mortgage payment.
# n is the length of the term in months, so it should be integer greater than 0.
# open demonstrate whether the mortgage term is open or closed.
  if (open == T) i <- 0.005 else i <- 0.004
  return(P*i/(1-(1+i)^(-n)))
}
```

While statements: Section 4.1.3

Exercise 2

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

Exercise 4

```
i_0 <- 0.006
i <- 0
while (abs(i-i_0)>=0.000001) {
  i <- i_0
  i_0 <- (1-(1+i)^(-20))/19
}
i
```

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

If try another starting point of 0.005

```
i_0 <- 0.005
i <- 0
while (abs(i-i_0)>=0.000001) {
  i <- i_0
  i_0 <- (1-(1+i)^(-20))/19
}
i
```

```
## [1] 0.004954847
```

The interest rates which satisfies the fixed-point equation are a little bit different using different start points. But the difference is very small.

Exercise 5

```
i_0 <- 0.006
i <- 0
n <- 0
while (abs(i-i_0)>=0.000001) {
  i <- i_0
  i_0 <- (1-(1+i)^(-20))/19
  n <- n+1
```

```
}
i
```

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

```
n
```

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

n here represents the number of iterations required to get the interest rate, and it is 74 in this case.

Break statements: Section 4.1.5

Exercise 2

(b)

```
Eratosthenes<- function(n){
  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]
    }
    return(primes)
  } else {
     stop("Input value of n should be at least 2.")
  }
}

numbers_greater_than_sqrt <- function(n){
  sieve <- seq(2,n)
  repeat{
    p <- sieve[1]
    sieve <- sieve[(sieve %% p) !=0]
    if (p >= sqrt(n)) break
  }
  return(sieve)
}

Eratosthenes(100)
```

```
##  [1]  2  3  5  7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83
## [24] 89 97
```

```
numbers_greater_than_sqrt(100)
```

```
##  [1] 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

The results indicate that once $p \geq \sqrt{n}$ all remaining entries in *sieve* are prime.

(c)

```
Eratosthenes <- function(n) {
  if (n >= 2) {
    sieve <- seq(2,n)
```

```r
    primes <- c()
    repeat {
      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.")
  }
}
```

Functions: Section 4.2.1

Exercise 2

(a)

```r
compound.interest <- function(P,n,i.r){
# Returns total amount if interest rate is compounded for n periods.
  P*(1+i.r)^n
}
```

(b)

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

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

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

Exercise 3

```r
bisection <- function(f,x1,x2) {
  repeat {
    f1 <- f(x1)
    f2 <- f(x2)
    x3 <- (x1+x2)/2
    f3 <- f(x3)
    if (f3 == 0) {
      return(x3)
      break
    } else {
      if (f1*f3>0) {
        x1 <- x3
      } else {
        x2 <- x3
      }
    }
  }
}
```

Puting it all together, checking: Section 4.4.1

Exercise 1

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

```
}
```

Test the validity of this function.

```r
mergesort(c(1,4,6,2,5,12,41,13,11), decreasing = TRUE)
```

```
## [1] 41 13 12 11  6  5  4  2  1
```

```r
mergesort(c(1,4,6,2,5,12,41,13,11), decreasing = FALSE)
```

```
## [1]  1  2  4  5  6 11 12 13 41
```

Exercise 2

(a)

```r
System_Newton <- function(f,g,x_0, y_0) {
  x <- x_0
  y <- y_0
  tolerance <- 0.000001
  while (abs(eval(f)) > tolerance | abs(eval(g)) > tolerance) {
    f_x <- eval(D(f,"x"))
    f_y <- eval(D(f,"y"))
    g_x <- eval(D(g,"x"))
    g_y <- eval(D(g,"y"))
    d <- f_x*g_y-f_y*g_x
    x <- x - (g_y*eval(f)-f_y*eval(g))/d
    y <- y - (f_x*eval(g)-g_x*eval(f))/d
  }
  print(c(x,y))
}
```

(b)

```r
System_Newton(expression(x+y),expression(x^2+2*y^2-2),0.8,0.8)
```

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

Chapter 4 Exercise

Exercise 1

```r
directpoly <- function(x,coefficient) {
  n <- length(coefficient)
  P <- rep(0,length(x))
  for(i in 1:n) {
    P <- P + coefficient[i]*(x^(i-1))
  }
  return(P)
}
```

Exercise 2

```r
hornerpoly <- function(x,coefficient) {
  n <- length(coefficient)
  a <- matrix(0,length(x),n)
```

```
  a[ ,n] <- coefficient[n]
  for (i in (n-1) : 1) {
    a[ ,i] <- a[ ,i+1]*x + coefficient[i]
  }
  return(a[ ,1])
}
```

Exercise 3

(a)

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

```
##    user  system elapsed
##    1.28    0.08    1.36
```

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

```
##    user  system elapsed
##    0.25    0.20    0.45
```

From the results, we can see that the speed of *hornerpoly* is much faster than *directpoly* when the number of x is large.

(b)

```
system.time(directpoly(x=1,c(-3,17,2)))
```

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

```
system.time(hornerpoly(x=1,c(-3,17,2)))
```

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

From the results, we can see that the speed of these two algorithms are the same when the number of polynomial coefficients is smaller.