# Homework 1

*Kenneth Ruiter (12206909)*

*October 16, 2018*

## For loops

### Exercise 2

(a) The first 29 elements of the sequence of ratios of the form $f_n/f_{n-1}$ is given by:

```
##  [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
```

Note that the sequence seems to be converging to about 1.618034.

(b) The golden ratio is $\frac{1+\sqrt{5}}{2} \approx 1.618034$. This does seem to be the value that the sequence in part (a) is converging to. We will prove that this is the case.

Let $n > 4$. Then
$$\frac{f_n}{f_{n-1}} = \frac{f_{n-1}+f_{n-2}}{f_{n-1}} = 1 + \frac{f_{n-2}}{f_{n-2}+f_{n-3}} = 1 + \frac{1}{\frac{f_{n-2}+f_{n-3}}{f_{n-2}}}.$$

Now note that we can rewrite the denominator of the final fraction, $\frac{f_{n-2}+f_{n-3}}{f_{n-2}}$, the same way as the second fraction, $\frac{f_{n-1}+f_{n-2}}{f_{n-1}}$, and as long as $n$ is big enough we can continue doing this. Thus if we are looking at the limit for $n$ to $\infty$, we can express the ratio as an "infinite fraction".

$$\frac{f_n}{f_{n-1}} = \frac{1}{1+\frac{1}{1+\frac{1}{\cdots}}}$$

Now all that remains to be show is that this infinite fraction is equal to the golden ratio. To do this, let us first assign the value $x$ to it, such that $x = \frac{1}{1+\frac{1}{1+\frac{1}{\cdots}}}$. However, then $1 + \frac{1}{x} = \frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{\cdots}}}} = x$. Now we can simply compute $x$ from this quadratic equation.

$$1 + \frac{1}{x} = x \implies x^2 - x - 1 = 0 \implies x = \frac{1 \pm \sqrt{1-4*1*-1}}{2*1} = \frac{1 \pm \sqrt{5}}{2}.$$

Noting that $x$ should be positive, finally proves that indeed $\frac{f_n}{f_{n-1}} \to \frac{1+\sqrt{5}}{2}$, as $n \to \infty$.

As a fun sidenote, nowhere in the proof were the starting values of the Fibonnaci series used. Thus not just this series, but any series that stars with two integers (at least one unequal 0), and the rest of the elements made with the same rules as the Fibonnaci series, will have this same property of the ratio between succeeding numbers converging to the golden ratio.

### Exercise 3

(a) We are taking 0 and adding the numbers $1, 2, 3, 4$ and 5 to it to obtain $0 + 1 + 2 + 3 + 4 + 5 = 15$.

Checking in R gives us that answer = 15.

(b) We begin with an empty set and then add the values 1 to 5 as elements to it, leaving us with the vector $(1, 2, 3, 4, 5)$.

Checking in R gives us that answer = (1, 2, 3, 4, 5).

(c) Instead of beginning with an empty set, we now start with the value 0. Adding the values 1 to 5 as elements results in the vector $(0, 1, 2, 3, 4, 5)$.

Checking in R gives us that answer = (0, 1, 2, 3, 4, 5).

(d) This time we are multiplying the answer by 1 to 5, so that the result is $1 * 1 * 2 * 3 * 4 * 5 = 120$.

Checking in R gives us that answer = 120.

(e) Now we are creating a vector of 16 elements, starting with 3, where each next element is given by 7∗(the last element) mod 31. This gives the vector $(3, 21, 23, 6, 11, 15, 12, 22, 30, 24, 13, 29, 17, 26, 27, 3)$.

Checking in R gives us that answer = (3, 21, 23, 6, 11, 15, 12, 22, 30, 24, 13, 29, 17, 26, 27, 3). Note here that the last element of the vector is the same as the first. By just inspecting the sequence of numbers, the pattern is not easy to see, however if we were to know (or guess) that the next element in the sequence is only dependent on the element preceding it, then we can predict that the successive element of the sequence is 21, as this is also the value of the second element.

## If statements

### Exercise 4

```r
interest <- function(P, periods){
  if(periods <= 3){
    i <- 0.04 #Annual interest for a term 3 years or less
  } else {
    i <- 0.05 #Annual interest for a term more than 3 years
  }

  I <- P*((1+i)^periods - 1)
  return(I)
}
```

### Exercise 5

```r
mortgage <- function(n, P, open){
  if(open){
    i <- 0.005 #Interest rate when the morgage term is open
  } else {
    i <- 0.004 #Interest rate when the morgage term is closed
  }

  R <- P*i/(1-(1+i)^(-n))
  return(R)
}
```

## While statements

### Exercise 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])
}
```

### Exercise 4

```r
i <- 0.006
iOld <- -1
while(abs(i - iOld) > 0.000001){
  iOld <- i
  i <- (1-(1+i)^(-20))/19
}
```

Using a starting guess of $i = 0.006$ results in an iteratively calculated value of $i = 0.0049541$. The result will be about the same (with a maximal difference of 0.0000001), for any starting value that is not close to 0. This is the case as 0 is also a fixed point, however it is unstable, unlike the value we find for $i$. Because of our initial choice of $iOld = -1$, a starting value close to $-1$ would also not give a desired result, however a negative guess for $i$ would not make sense for this example anyway.

### Exercise 5

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

Again starting with $i = 0.006$, we find our desired estimate of $i$ in 74 iterations.

# Break statements

## Exercise 2

(a) Done.

(b) Let $p \geq \sqrt{n}$. Now consider any integer $q$, such that $p \leq q \leq n$.
If $q$ is prime, then it is still in *sieve* as there cannot have been any number $\tilde{p}$ in *sieve* smaller than $q$, such that $q = 0 \mod \tilde{p}$, by the definition of a prime number.
If $q$ is not prime, then there must exist integers $a, b \neq 1$ such that $q = a * b$. Now suppose $a, b > \sqrt{n}$. Then $q = a * b > \sqrt{n} * \sqrt{n} = n$, which contradicts our assumption, meaning that at least one of $a$ and $b$ must be smaller than or equal to $\sqrt{n}$. Without loss of generality, assume that $a \leq \sqrt{n}$. Now if $a$ is prime, then at some point we had that $p = a$, at which point then $q = 0 \mod p$, so $q$ can currently not be an element of *sieve* anymore. If $a$ is not prime, then there must have been some point where $p = \tilde{a}$ where $a = 0 \mod p$. But then also $q = a * b = 0 \mod p$, meaning $q$ can currently still not be an element of *sieve*. Therefore all elements left in *sieve* once $p \geq \sqrt{n}$ must be prime.

(c)
```r
Erastosthenes <- function(n) {
  if(n >= 2) {
    sieve <- seq(2,n)
    primes <- c()
    repeat{
      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.")
  }
}
```

## Functions

### Exercise 2

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

(b) In this case we have that $P = 1000, i.r = 0.01$ and $n = 30$. Plugging these values into the function defined in part (a) gives a total amount of money in the bank of \$1347.85.

### Exercise 3

```r
bisection.zero <- function(f, x1, x2, tolerance = 10^-8){
  if(abs(f(x1)) <= tolerance){ #Checking if f(x1) is already close enough to 0
    return(x1)
  } else if(abs(f(x2)) <= tolerance){ #Checking if f(x2) is already close enough to 0
    return(x2)
  } else{
    if(sign(f(x1)) != sign(f(x2))){ #Checking if f(x1) and f(x2) have opposite signs
      x3 <- (x1 + x2)/2
      while(abs(f(x3)) > tolerance) {
        if(sign(f(x1)) != sign(f(x3))){
          x2 <- x3
        } else { #Now f(x2) and f(x3) must have opposite signs
          x1 <- x3
        }
        x3 <- (x1 + x2)/2
      }
      return(x3)
    } else {
      stop("The initial starting values must give opposite signs.")
    }
  }
}
```

## Putting it all together

### Exercise 1

```r
mergesort <- function (x, decreasing = FALSE) {
  # Check for a vector that doesn't need sorting
  len <-length(x)
  if (len < 2) result <- x
  else {
    # 2: sort x into result
    # 2.1: split x in half
    y <- x[1:(len %/% 2)]
    z <- x[(len %/% 2 + 1):len]
    # 2.2: sort y and z
    y <- mergesort(y)
    z <- mergesort(z)
    # 2.3: merge y and z into a sorted result
    result <- c()
    # 2.3.1: while (some are left in both piles)
    while (min(length(y), length(z)) > 0) {
      # 2.3.2: put the smallest first element on the end
      # 2.3.3: remove it from y or z
      if (y[1] < z[1]) {
        result <- c(result, y[1])
        y <- y[-1]
      } else {
        result <- c(result, z[1])
        z <- z[-1]
      }
    }
    # 2.3.4: put the leftovers onto the end of result
    if (length(y) > 0)
      result <- c(result, y)
    else
      result <- c(result, z)
  }
  # Modification to include decreasing
  if(decreasing){
    decreasingresult <- rep(0,length(result))
    for(i in 1:length(result)){
      decreasingresult[i] <- result[length(result)+1-i]
    }
    result <- decreasingresult
  }
  return(result)
}
```

### Exercise 2

(a)
```r
findzero <- function(f, g, fx, fy, gx, gy, x0, y0) {
  x <- x0
  y <- y0
  tolerance <- 0.000001
  while(abs(f(x,y)) > tolerance || abs(g(x,y)) > tolerance) {
    d <- fx(x,y)*gy(x,y) - fy(x,y)*gx(x,y)
    if(d == 0){
      stop("Try different starting values")
    }
    newx <- x - (gy(x,y)*f(x,y) - fy(x,y)*g(x,y))/d
    newy <- y - (fx(x,y)*g(x,y) - gx(x,y)*f(x,y))/d
    x <- newx
    y <- newy
  }
  return(c(x,y))
}
```

(b) Applying the function to the system

$$\begin{aligned} x + y &= 0 \\ x^2 + 2y^2 - 2 &= 0, \end{aligned}$$

with initial guess $(x_0, y_0) = (1, 1)$ gives $(x, y) = (-0.8164966, 0.8164966)$. We can analytically solve this system of equations to check wether this is the correct answer.

$$\begin{aligned} x + y = 0 &\implies y = -x \\ x^2 + 2y^2 - 2 = 0 &\implies x^2 + 2(-x)^2 - 2 \\ &\implies 3x^2 = 2 \\ &\implies x = \pm\sqrt{\frac{2}{3}} \\ y = -x &\implies y = \mp\sqrt{\frac{2}{3}} \end{aligned}$$

The numerically found answer corresponds with the answer $(x, y) = (-\sqrt{\frac{2}{3}}, \sqrt{\frac{2}{3}}) \approx (-0.8164966, 0.8164966)$.

# Chapter 4 exercises

## Exercise 1

```
directpoly <- function(x, vec.coef){
  n <- length(vec.coef)
  m <- length(x)
  powers <- matrix(rep(0,n*m), nrow = m)
  evaluation <- rep(0,m)
  for (i in 1:m){
    for(j in 1:n){
     powers[i,j] <- x[i]^(j-1)
    }
    evaluation[i] <- sum(vec.coef * powers[i,])
  }
  return(evaluation)
}
```

## Exercise 2

```
hornerpoly <- function(x, vec.coef){
  n <- length(vec.coef)
  m <- length(x)
  a <- matrix(rep(c(numeric(n-1), vec.coef[n]),m), nrow = m, byrow = TRUE)
  for(i in 1:m){
    for (j in (n-1):1) {
      a[i,j] <- a[i,j+1]*x[i] + vec.coef[j]
    }
  }
  return(a[,1])
}
```

To test the function, we try the polynomial $P(x) = 3x^2 + 2x + 1$ evaluated at $x = 2$. This should result in $3 * 2^2 + 2 * 2 + 1 = 17$. Indeed, the function $hornerpoly(2, c(1, 2, 3))$ does output 17.

## Exercise 3

(a) The *hornerpoly* function seems to be more efficient than the *directpoly* function.

```
##    user  system elapsed
##   11.54    0.28   11.83
```

```
##    user  system elapsed
##    5.83    0.17    6.00
```

(b) The total time for both methods significantly decreases for smaller polynomials.

```
##    user  system elapsed
##    6.98    0.13    7.11
```

```
##    user  system elapsed
##    2.39    0.13    2.51
```