Assignment #1

Kim Ting Li

October 3, 2017

4.1.1 #2

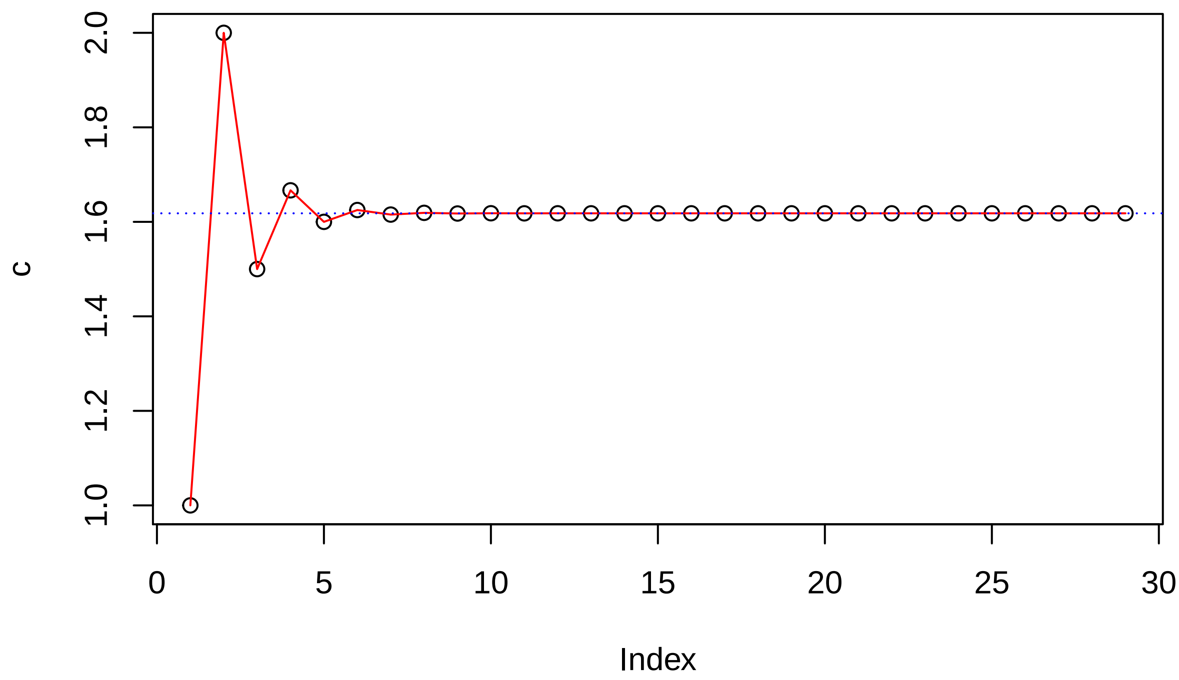
```r
#Yes, the sequence appear to be converging
fn <- numeric(30)
fn[1] <- fn[2] <- 1
for (i in 3:30) fn[i] <- fn[i-2] + fn[i-1]
c<-numeric(29)
for (j in 2:30) c[j-1]<-(fn[j]/fn[j-1])
print(c)
```

```
##  [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
```

```r
#result for golden ratio
a<-((1+sqrt(5))/2)
a
```

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

```r
# We can show the convergence by plotting the sequence and show
#it goes towards the horizontal line that's the golden ratio
plot(c)
lines(c,col='red')
abline(h=((1+sqrt(5))/2), col='blue',lty=3)
```



4.1.1 #3

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

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

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

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

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

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

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

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

```
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
```

4.1.2 #4

```
GICreturn<-function(P,n){
  if(n<=3) return(P*((1+0.04)^n-1))
  else return(P*((1+0.05)^n-1))
}
```

4.1.2 #5

```
#when it is an open mortgage, put TRUE for the OPEN argument
R<-function(p,n,OPEN){
  if (OPEN==TRUE) i=0.005
  else i=0.004
  R=(p*i/(1-(1+i)^(-n)))
  return (R)
}
```

4.1.3 #2

```
Fibonacci<-NULL
Fibonacci[1]<-1
Fibonacci[2]<-1
Fibonacci<-c(Fibonacci[1],Fibonacci[2])
while ((Fibonacci[length(Fibonacci)]+Fibonacci[length(Fibonacci)-1])<300) {
  Fibonacci<-c(Fibonacci,(Fibonacci[length(Fibonacci)]+Fibonacci[length(Fibonacci)-1]))
}
print(Fibonacci)
```

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

4.1.3 #4

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

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

4.1.3 #5

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

```r
  i<-(1-(1+i)^(-20))/19
  a=a+1}
print(a)
```

```
## [1] 73
```

```r
print(i)
```

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

4.1.5 #2

```r
#Results for the original function and the modified function
#with BREAK statement are listed below, they are the same.
#Therefore, question b is demonstrated.
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
}
}
Eratosthenes(200)
```

```
##  [1]   2   3   5   7  11  13  17  19  23  29  31  37  41  43  47  53  59
## [18]  61  67  71  73  79  83  89  97 101 103 107 109 113 127 131 137 139
## [35] 149 151 157 163 167 173 179 181 191 193 197 199
```

```r
ModifiedE <- function(n) {
if (n >= 2) {
  sieve <- seq(2, n)
  primes <- c()
  while (length(sieve) > 0) {
    p <- sieve[1]
    if (p>=sqrt(n)) break
    primes <- c(primes, p)
    sieve <- sieve[(sieve %% p) != 0]

  }
  primes<-c(primes, sieve)
return(primes)
} else {
stop
}
}
ModifiedE(200)
```

```
##  [1]   2   3   5   7  11  13  17  19  23  29  31  37  41  43  47  53  59
## [18]  61  67  71  73  79  83  89  97 101 103 107 109 113 127 131 137 139
## [35] 149 151 157 163 167 173 179 181 191 193 197 199
```

```
Eratosthenes(200)==ModifiedE(200)
```

```
##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [15] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [29] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [43] TRUE TRUE TRUE TRUE
```

4.2.1 #2

```
compound.interest<-function(P,ir,n){
  x<-(P*(1+ir)^n)
  return(x)
}
compound.interest(1000,0.01,30)
```

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

4.2.1 #3

```
#to give an answer for the precision of 6 decimal places
bisection<-function(f,x1,x2){
  repeat{
  f1<-f(x1)
  f2<-f(x2)
  x3<-(x1+x2)/2
  f3<-f(x3)
  if (f3==0){
    print(x3)
    break
  }
  else {
    if (f3*f1<0){
      x1<-x3
    } else{
      x2<-x3
    }
  }
  if (abs(x1-x2)<0.000001){
    print((x1+x2)/2)
    break
  }
  }
}
```

4.4.1 #1

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

    }
  }else{
    len <-length(x)
  if (len < 2) result <- x
  else {
    y <- x[1:(len / 2)]
    z <- x[(len / 2 + 1):len]
    y <- mergesort(y,FALSE)
    z <- mergesort(z,FALSE)
    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)

}

a<-c(2,5,1,6,7,11,8,10)
b<-c(2,5,1,6,7,11,8,10)
mergesort(a,TRUE)
```

```
## [1] 11 10  8  7  6  5  2  1
```

```
mergesort(b,FALSE)
```

```
## [1]  1  2  5  6  7  8 10 11
```

4.4.1 #2 !!!!!

```
  f<-function(x,y) (x+y)
  g<-function(x,y) (x^2+2*y^2-2)

answer<-function(f,g,x,y){

  dfx<-function(x,y){}
  dfy<-function(x,y){}
  dgx<-function(x,y){}
  dgy<-function(x,y){}

  body(dfx)=D(body(f),'x')
  body(dfy)=D(body(f),'y')
  body(dgx)=D(body(g),'x')
  body(dgy)=D(body(g),'y')

  tolerance<-0.000001

  while (f(x,y) > tolerance){
    x <- x-(dgy(x,y)*f(x,y)-dfy(x,y)*g(x,y))/(dfx(x,y)*dgy(x,y)-dfy(x,y)*dgx(x,y))
    y <- y-(dfx(x,y)*g(x,y)-dgx(x,y)*f(x,y))/(dfx(x,y)*dgy(x,y)-dfy(x,y)*dgx(x,y))
  }
  print(x)
  print(y)
}

answer(f,g,100,100)
```

```
## [1] -150.01
## [1] 60.71969
```

Chapter 4 #1

```r
directpoly<-function(x,c){
   p<-0
    for(a in 1:length(c)){
    p1<-(c[a]*x^(a-1))
    p=p+p1
  }
  return(p)
}

directpoly(2,c(4,6,2,9))
```

```
## [1] 96
```

```r
directpoly(c(2,5),c(4,6,2,9))
```

```
## [1]   96 1209
```

Chapter 4 #2

```r
hornerpoly<-function(x,c){
  p<-0
  a<-matrix(0,length(x),length(c))
  a[,length(c)]<-c[length(c)]
  for (i in (length(c)-1):1){
    a[,i]<-a[,i+1]*x+c[i]
  }
  return(a[,1])
}
hornerpoly(2,c(4,6,2,9))
```

```
## [1] 96
```

```r
hornerpoly(c(2,5),c(4,6,2,9))
```

```
## [1]   96 1209
```

Chapter 4 #3

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

```
##    user  system elapsed
##    1.84    0.17    2.02
```

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

```
##    user  system elapsed
##    0.49    0.27    0.75
```

```r
#the first algorithm is slower in user time and elapsed time,
#but the system time is similar.
```

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

```
##    user  system elapsed
##    0.35    0.01    0.36
```

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

```
##    user  system elapsed
##    0.22    0.03    0.25
```

```r
#when the number of coefficients is smaller, it takes less time to execute
#the code in general. The two algorithms also run at very similar speed
#when the number of coefficient is small.
```