Assignment 1

Section 4.1.1

2

a

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

```
## [1] 1
## [1] 2
## [1] 1.5
## [1] 1.666667
## [1] 1.6
## [1] 1.625
## [1] 1.615385
## [1] 1.619048
## [1] 1.617647
## [1] 1.618182
## [1] 1.617978
## [1] 1.618056
## [1] 1.618026
## [1] 1.618037
## [1] 1.618033
## [1] 1.618034
## [1] 1.618034
## [1] 1.618034
## [1] 1.618034
## [1] 1.618034
## [1] 1.618034
## [1] 1.618034
## [1] 1.618034
## [1] 1.618034
## [1] 1.618034
## [1] 1.618034
## [1] 1.618034
## [1] 1.618034
## [1] 1.618034
```

It seems that this sequence appears to be converging.

b

```
print((1+sqrt(5))/2)
```

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

It shows that the sequence converging to the golden ratio.
Proof:
$F_i + F_{i+1} = F_{i+2}$
$\frac{F_i}{F_{i+1}} + 1 = \frac{F_{i+2}}{F_{i+1}}$
As we can see that this sequence appears to be converging, then we let $\frac{F_{i+1}}{F_i} \to x$
then the upper equation can be like
$x = 1 + \frac{1}{x}$
and we know that $x > 0$
by solving this equation, we can get that
$x = \frac{1+\sqrt{5}}{2}$

3

a

answer=15

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

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

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
```

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
```

d

answer=120

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

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

e

answer=(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
```

I think it may repeat the first 14 sequence, like it is 14 elements a loop and repeat over and over again.

Section 4.1.2

4

```
interest<-function(amount,periods){
  I=0
  if(periods<=3){
    I=amount*((1+0.04)^periods-1)
  } else{
    I=amount*((1+0.05)^periods-1)
  }
  return(I)
}
interest(10000,3)
```

```
## [1] 1248.64
```

```
interest(10000,5)
```

```
## [1] 2762.816
```

5

```
mortgage<-function(n,p,open){
  R=0
  if(open==TRUE){
    i=0.005
    R=p*i/(1-(1+i)^(-n))
  } else{
    i=0.004
    R=p*i/(1-(1+i)^(-n))
  }
  return(R)
```

```
}
mortgage(12,10000,TRUE)
```

```
## [1] 860.6643
```

```
mortgage(12,10000,FALSE)
```

```
## [1] 855.1586
```

Section 4.1.3

2

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

```
##  [1]   1   1   2   3   5   8  13  21  34  55  89 144 233 377
```

4

```
i1=0
i2=0.006
while(abs(i2-i1)>=0.000001){
  i1=i2
  i2=(1-(1+i2)^(-20))/19
}
i2
```

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

If I try another starting gusses, like i=0.1

```
i1=0
i2=0.1
while(abs(i2-i1)>=0.000001){
  i1=i2
  i2=(1-(1+i2)^(-20))/19
}
i2
```

```
## [1] 0.004953499
```

It seems that the result will be a little different after the 0.000001 part. But almost doesn't change.

5

```
i1=0
i2=0.006
n=0
while(abs(i2-i1)>=0.000001){
  n=n+1
  i1=i2
  i2=(1-(1+i2)^(-20))/19
```

```
}
i2
```

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

```
n
```

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

Section 4.1.5

2

(a)

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

This function is used to generate all the prime numbers between 2 and n. It works because it will remove all the non-prime numbers in the sequence and add prime number one by one in the loop.

(b)

```
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]
      if(p>=sqrt(n)){
        print(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(100)
```

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

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

We can see that once p >= sqrt(n), all the remaining entries in sieve are prime (comparing the results we get in (a), we can see that this is true)

(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]
      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(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
```

Section 4.2.1

2

(a)

```
compound.interest<-function(P,i.r,n){
  n=P*(1+i.r)^n
  return(n)
}
compound.interest(1000,0.04,3)
```

```
## [1] 1124.864
```

(b)

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

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

We can see that Mr.Ng will have $1347.849 at the end of 30 months.

3

```r
bisec<-function(Fun,i1,i2){
  if(Fun(i1)==0){
    return(i1)
  }
  if(Fun(i2)==0){
    return(i2)
  }
  while(abs(Fun(n))>=0.000001){
    n=(i1+i2)/2
    if(Fun(i1)*Fun(n)<0){
      i2=n
    } else if(Fun(i2)*Fun(n)<0){
      i1=n
    } else{
      print("we don't know whether it has a zero point" )
      return(0)
    }
  }
  return(n)
}
Fun<-function(x){
  return(x^3-0.5)
}
bisec(Fun,-1,1)
```

```
## [1] 0.7937002
```

We assume that the function user supplied is monotonic in the domain they give ([i1,i2]), and there is a zero point in the domain ([i1,i2]).
Or if the function is not monotonic, then at the middle of the domain (n=(i1+i2)/2), the function must have a different sign (which means Fun(i1)*Fun(n)<0 or Fun(i2)*Fun(n)<0). Otherwise we don't know whether it has a zero point in the domain.

Section 4.4.1

1

```r
mergesort<-function(x,decreasing){
  len<-length(x)
  if(len<2) result<-x
  else if(decreasing==FALSE){
    y<-x[1:(len/2)]
    z<-x[(len/2+1):len]
    y<-mergesort(y,decreasing)
    z<-mergesort(z,decreasing)
    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{
    y<-x[1:(len/2)]
    z<-x[(len/2+1):len]
    y<-mergesort(y,decreasing)
    z<-mergesort(z,decreasing)
    result<-c()
    while(min(length(y),length(z))>0){
      if(y[1]<z[1]){
        result<-c(result,z[1])
        z<-z[-1]
      } else{
        result<-c(result,y[1])
        y<-y[-1]
      }
    }
    if(length(y)>0)
      result<-c(result,y)
    else
      result<-c(result,z)
  }
  return(result)
}
mergesort(c(2,5,14,8,5,13,10,9,7),TRUE)
```

```
## [1] 14 13 10  9  8  5  5  2
```

```
mergesort(c(2,5,14,8,5,13,10,9,7),FALSE)
```

```
## [1]  2  5  5  8  9 10 13 14
```

2

(a)

```
f<-function(x,y,coef1,coef2,coef12,dimx,dimy,cons){
  n1=length(coef1)
  n2=length(coef2)
  n12=length(coef12)
  p=0
  for(i in 1:n1){
```

```
    p=p+coef1[i]*x^i
  }
  for(i in 1:n2){
    p=p+coef2[i]*y^i
  }
  for(i in 1:n12){
    p=p+coef12[i]*x^dimx[i]*y^dimy[i]
  }
  p=p+cons
  return(p)
}
dx<-function(x,y,coef1,coef2,coef12,dimx,dimy,cons){
  n1=length(coef1)
  n2=length(coef2)
  n12=length(coef12)
  p=0
  for(i in 1:n1){
    p=p+i*coef1[i]*x^(i-1)
  }
  for(i in 1:n12){
    p=p+dimx[i]*coef12[i]*x^(dimx[i]-1)*y^dimy[i]
  }
  return(p)
}
dy<-function(x,y,coef1,coef2,coef12,dimx,dimy,cons){
  n1=length(coef1)
  n2=length(coef2)
  n12=length(coef12)
  p=0
  for(i in 1:n2){
    p=p+i*coef2[i]*y^(i-1)
  }
  for(i in 1:n12){
    p=p+dimy[i]*coef12[i]*x^dimx[i]*y^(dimy[i]-1)
  }
  return(p)
}
newtown<-function(x0=1,y0=1,coef1f=0,coef2f=0,coef12f=0,dimxf=0,dimyf=0,consf=0,coef1g=0,coef2g=0,coef1
  #x0=5
  #y0=5
  fx0=dx(x0,y0,coef1f,coef2f,coef12f,dimxf,dimyf,consf)
  fy0=dy(x0,y0,coef1f,coef2f,coef12f,dimxf,dimyf,consf)
  gx0=dx(x0,y0,coef1g,coef2g,coef12g,dimxg,dimyg,consg)
  gy0=dy(x0,y0,coef1g,coef2g,coef12g,dimxg,dimyg,consg)
  f0=f(x0,y0,coef1f,coef2f,coef12f,dimxf,dimyf,consf)
  g0=f(x0,y0,coef1g,coef2g,coef12g,dimxg,dimyg,consg)
  d0=fx0*gy0-fy0*gx0
  x1=x0-(gy0*f0-fy0*g0)/d0
  y1=y0-(fx0*g0-gx0*f0)/d0
  while(max(abs(x1-x0),abs(y1-y0))>=0.0000001){
    x0=x1
    y0=y1
    fx0=dx(x0,y0,coef1f,coef2f,coef12f,dimxf,dimyf,consf)
```

```
    fy0=dy(x0,y0,coef1f,coef2f,coef12f,dimxf,dimyf,consf)
    gx0=dx(x0,y0,coef1g,coef2g,coef12g,dimxg,dimyg,consg)
    gy0=dy(x0,y0,coef1g,coef2g,coef12g,dimxg,dimyg,consg)
    f0=f(x0,y0,coef1f,coef2f,coef12f,dimxf,dimyf,consf)
    g0=f(x0,y0,coef1g,coef2g,coef12g,dimxg,dimyg,consg)
    d0=fx0*gy0-fy0*gx0
    x1=x0-(gy0*f0-fy0*g0)/d0
    y1=y0-(fx0*g0-gx0*f0)/d0
  }
  return(c(x1,y1))
}
newtown(coef1f=c(1,1,3),coef2f=c(0,2),coef12f=1,dimxf=1,dimyf=2,consf=1,coef1g=c(1,1),coef2g=1)
```

```
## [1] -0.6549137  0.2260017
```

We can see that the function can solve these two equation functions and it applys to any format of the expression, including the $x^a y^b$

(b)

```
s1=newtown(x0=1,y0=1,coef1f=1,coef2f=1,coef1g=c(0,1),coef2g=c(0,2),consg=-2)
s2=newtown(x0=-1,y0=-1,coef1f=1,coef2f=1,coef1g=c(0,1),coef2g=c(0,2),consg=-2)
```

So we can see that the two solutions are (-0.8164966,0.8164966) and (0.8164966,-0.8164966)

Chapter 4 Exercise

1

```
directpoly<-function(x,coef){
  n=length(coef)
  p=0
  for(i in 1:n){
    p=p+coef[i]*x^(i-1)
  }
  return(p)
}
directpoly(4,c(2,3,1,1))
```

```
## [1] 94
```

2

```
hornerpoly<-function(x,coef){
  n=length(coef)
  m=length(x)
  a<-matrix(nrow=n,ncol=m)
  a[n,]<-coef[n]
  i=n-1
  while(i >= 1){
    a[i,]=a[i+1,]*x+coef[i]
    i=i-1
  }
  return(a[1,])
```

```r
}
hornerpoly(4,c(2,3,1,1))
```

```
## [1] 94
```

```r
hornerpoly(c(3,4,5),c(2,3,1,1))
```

```
## [1]  47  94 167
```

3

(a)

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

```
##    user  system elapsed
##    1.48    0.05    1.53
```

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

```
##    user  system elapsed
##    0.63    0.13    0.75
```

So we can see that the hornerpoly method is much more efficient when the n is large.

(b)

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

```
##    user  system elapsed
##    0.33    0.00    0.32
```

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

```
##    user  system elapsed
##    0.12    0.11    0.23
```

It seems when the n is small, the difference in timings is not that big, although the horner method is more efficient.