PC4b - Roots

Question 1a

The derivatives are given by

```
D(expression(acos(1/2*sin(x))), "x")
```

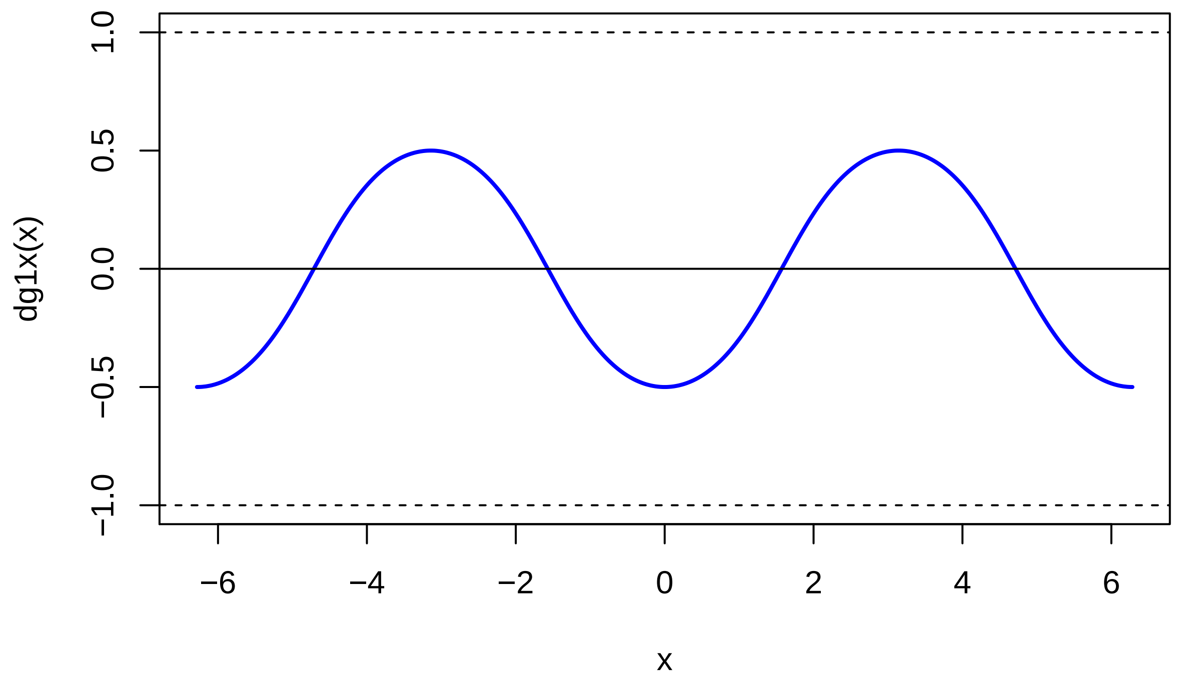
```
## -(1/2 * cos(x)/sqrt(1 - (1/2 * sin(x))^2))
```

```
D(expression(asin(2*cos(x))), "x")
```

```
## -(2 * sin(x)/sqrt(1 - (2 * cos(x))^2))
```

The code below defines the two derivatives as R-functions and shows them in a plot

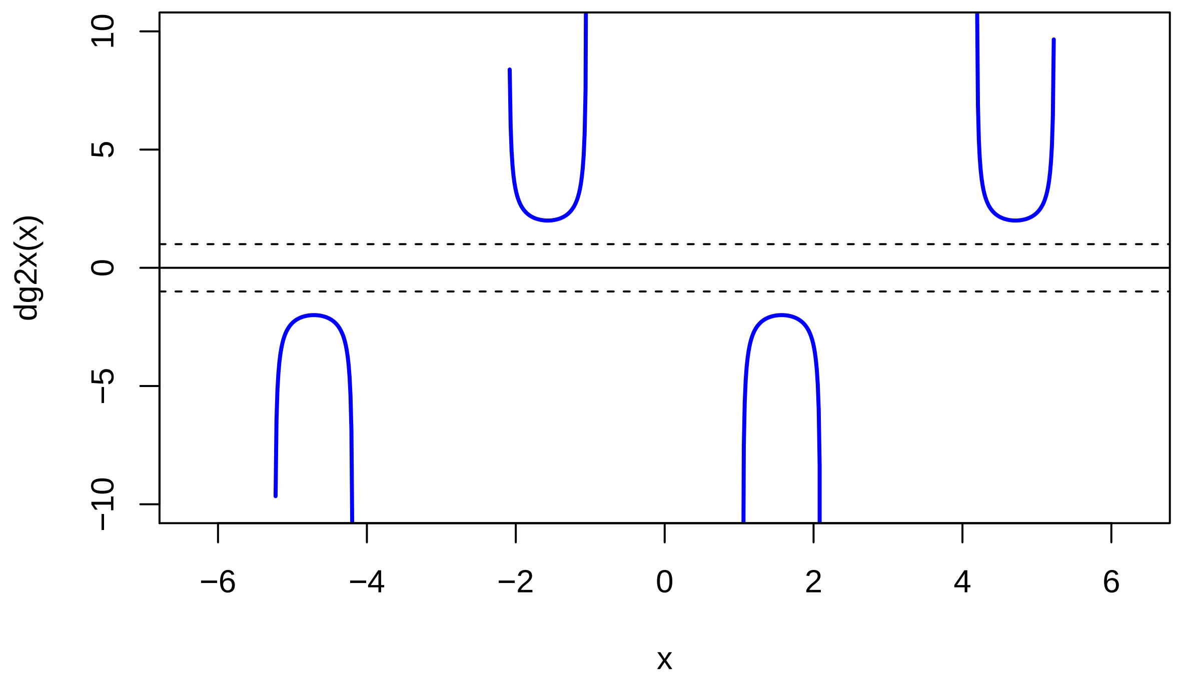
```
dg1x<-function(x){
  -(1/2 * cos(x)/sqrt(1 - (1/2 * sin(x))^2))
}
x=seq(-2*pi,2*pi,length=1e3)
plot(x,dg1x(x),type="l",col="blue",lwd=2,ylim=c(-1,1))
abline(0,0); abline(-1,0,lty=2); abline(+1,0,lty=2)
```



```r
dg2x<-function(x){
  -(2 * sin(x)/sqrt(1 - (2 * cos(x))^2))
}
x=seq(-2*pi,2*pi,length=1e3)
plot(x,dg2x(x),type="l",col="blue",lwd=2,ylim=c(-10,10))
```

```
## Warning in sqrt(1 - (2 * cos(x))^2): NaNs produced
```

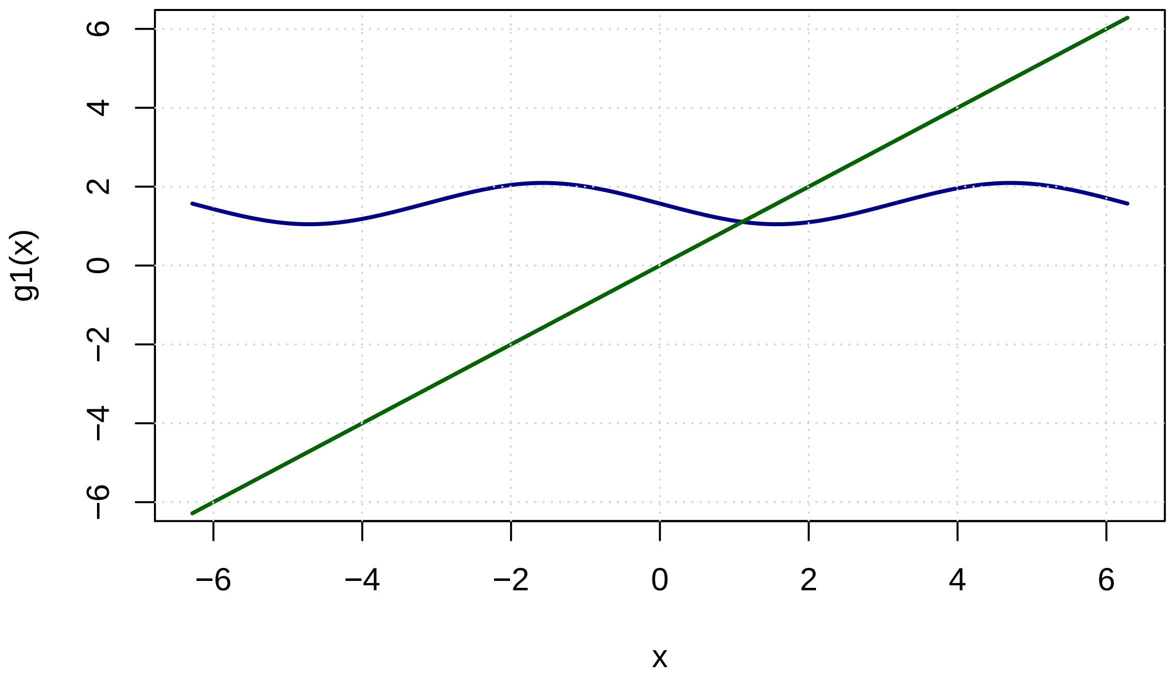
```r
abline(0,0); abline(-1,0,lty=2); abline(+1,0,lty=2)
```



The fixed-point method only converges if the absolute value of the derivative is smaller than one. From the graphs, we see that $|g_1'(x)| < 1$, while $|g_2'(x)| > 1$ for this interval, so only $g_1(x)$ will converge.

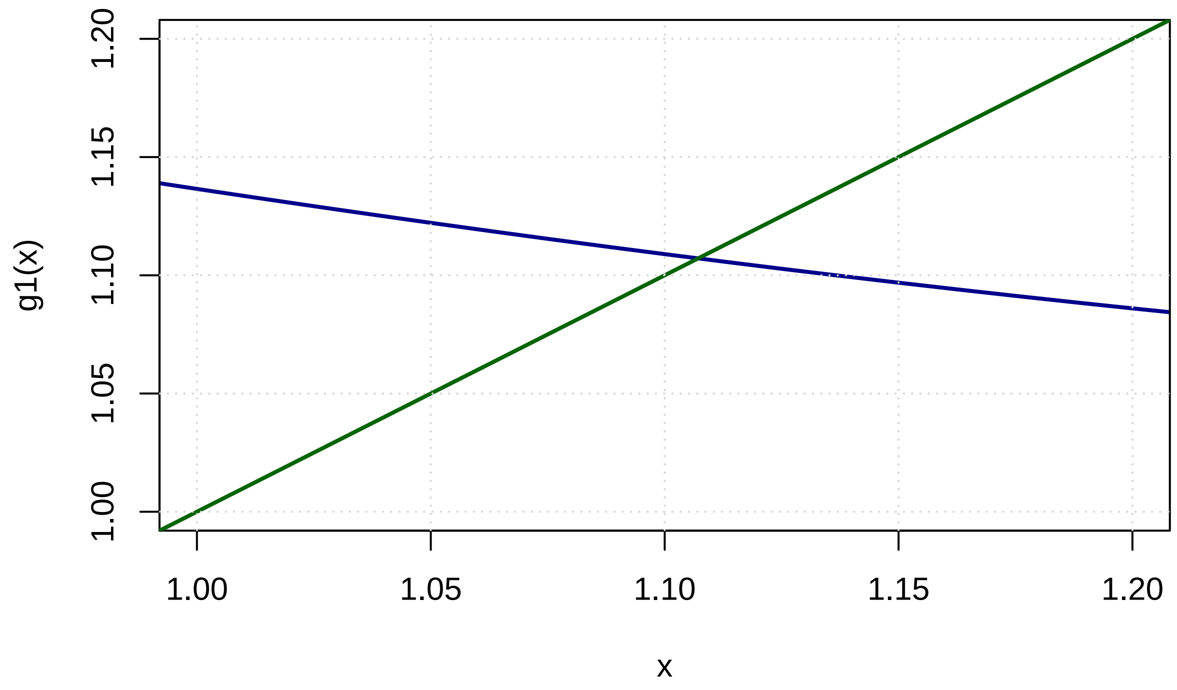
Question 1b

```r
g1<-function(x){
  acos(1/2*sin(x))
}
plot(x,g1(x),type="l",col="darkblue",lwd=2,ylim=c(-6,6))
lines(x,x,col="darkgreen",lwd=2)
grid()
```



There is only one intersection around $x = 1$. The next graph zooms in.

```
plot(x,g1(x),type="l",col="darkblue",lwd=2,xlim=c(1,1.2),ylim=c(1,1.2))
lines(x,x,col="darkgreen",lwd=2)
grid()
```



The fixed point seems to be approximately 1.11. Next, we consider the fixed-point method for different starting values. For this, we first modify the code of the function `fixedpoint()` do that it returns a vector `c(xnew,iter)`

```
FP <- function(ftn, x0, tol = 1e-9, max.iter = 100) {
  xold <- x0
  xnew <- ftn(xold)
  iter <- 1
  while ((abs(xnew-xold) > tol) && (iter < max.iter)) {
    xold <- xnew;
    xnew <- ftn(xold);
    iter <- iter + 1
  }
  if (abs(xnew-xold) > tol) {
    return(NULL)
  } else {
    return(c(xnew,iter))
  }
}
```

The code below generates the desired table. Not that if `results==NULL`, the fixed-point algorithm didn't converge

```
for(x0 in (-5:5)){
  results=FP(g1,x0)
  if(is.null(results)){
    cat(sprintf("x0=%2d: no root found\n"),x0)
  } else {
    cat(sprintf("x0=%2d: Root=%.5f found in %d iterations\n",x0,results[1],results[2]))
  }
}
```

```
## x0=-5: Root=1.10715 found in 15 iterations
## x0=-4: Root=1.10715 found in 16 iterations
## x0=-3: Root=1.10715 found in 17 iterations
## x0=-2: Root=1.10715 found in 14 iterations
## x0=-1: Root=1.10715 found in 15 iterations
## x0= 0: Root=1.10715 found in 17 iterations
## x0= 1: Root=1.10715 found in 15 iterations
## x0= 2: Root=1.10715 found in 14 iterations
## x0= 3: Root=1.10715 found in 17 iterations
## x0= 4: Root=1.10715 found in 16 iterations
## x0= 5: Root=1.10715 found in 15 iterations
```

Question 2

From the graph of $f(x) = 2cos(x) - sin(x)$, we observe that there are four roots in the neighborhood of $-5, -2, 1$ and 4.5. Hence, we consider intervals of width 1 around these visual guesses!

```
f<-function(x){
  2*cos(x)-sin(x)
}

#install.packages("spuRs")
library(spuRs)
```

```
## Loading required package: MASS
```

```
## Loading required package: lattice
```

```
bisection(f,-5.5,-4.5,1e-4)
```

```
## at iteration 1 the root lies between -5.5 and -5
## at iteration 2 the root lies between -5.25 and -5
## at iteration 3 the root lies between -5.25 and -5.125
## at iteration 4 the root lies between -5.1875 and -5.125
## at iteration 5 the root lies between -5.1875 and -5.15625
## at iteration 6 the root lies between -5.1875 and -5.171875
## at iteration 7 the root lies between -5.179688 and -5.171875
## at iteration 8 the root lies between -5.179688 and -5.175781
## at iteration 9 the root lies between -5.177734 and -5.175781
## at iteration 10 the root lies between -5.176758 and -5.175781
## at iteration 11 the root lies between -5.17627 and -5.175781
## at iteration 12 the root lies between -5.17627 and -5.176025
## at iteration 13 the root lies between -5.176147 and -5.176025
## at iteration 14 the root lies between -5.176086 and -5.176025
```

```
## [1] -5.176056
```

```
bisection(f,-2.5,-1.5,1e-4)
```

```
## at iteration 1 the root lies between -2.5 and -2
## at iteration 2 the root lies between -2.25 and -2
## at iteration 3 the root lies between -2.125 and -2
## at iteration 4 the root lies between -2.0625 and -2
## at iteration 5 the root lies between -2.0625 and -2.03125
## at iteration 6 the root lies between -2.046875 and -2.03125
## at iteration 7 the root lies between -2.039062 and -2.03125
## at iteration 8 the root lies between -2.035156 and -2.03125
## at iteration 9 the root lies between -2.035156 and -2.033203
## at iteration 10 the root lies between -2.035156 and -2.03418
## at iteration 11 the root lies between -2.034668 and -2.03418
## at iteration 12 the root lies between -2.034668 and -2.034424
## at iteration 13 the root lies between -2.034546 and -2.034424
## at iteration 14 the root lies between -2.034485 and -2.034424
```

```
## [1] -2.034454
```

```
bisection(f, 0.5, 1.5,1e-4)
```

```
## at iteration 1 the root lies between 1 and 1.5
## at iteration 2 the root lies between 1 and 1.25
## at iteration 3 the root lies between 1 and 1.125
## at iteration 4 the root lies between 1.0625 and 1.125
## at iteration 5 the root lies between 1.09375 and 1.125
## at iteration 6 the root lies between 1.09375 and 1.109375
## at iteration 7 the root lies between 1.101562 and 1.109375
## at iteration 8 the root lies between 1.105469 and 1.109375
## at iteration 9 the root lies between 1.105469 and 1.107422
## at iteration 10 the root lies between 1.106445 and 1.107422
## at iteration 11 the root lies between 1.106934 and 1.107422
## at iteration 12 the root lies between 1.106934 and 1.107178
## at iteration 13 the root lies between 1.107056 and 1.107178
## at iteration 14 the root lies between 1.107117 and 1.107178
```

```
## [1] 1.107147
```

```
bisection(f, 4.0, 5.0,1e-4)
```

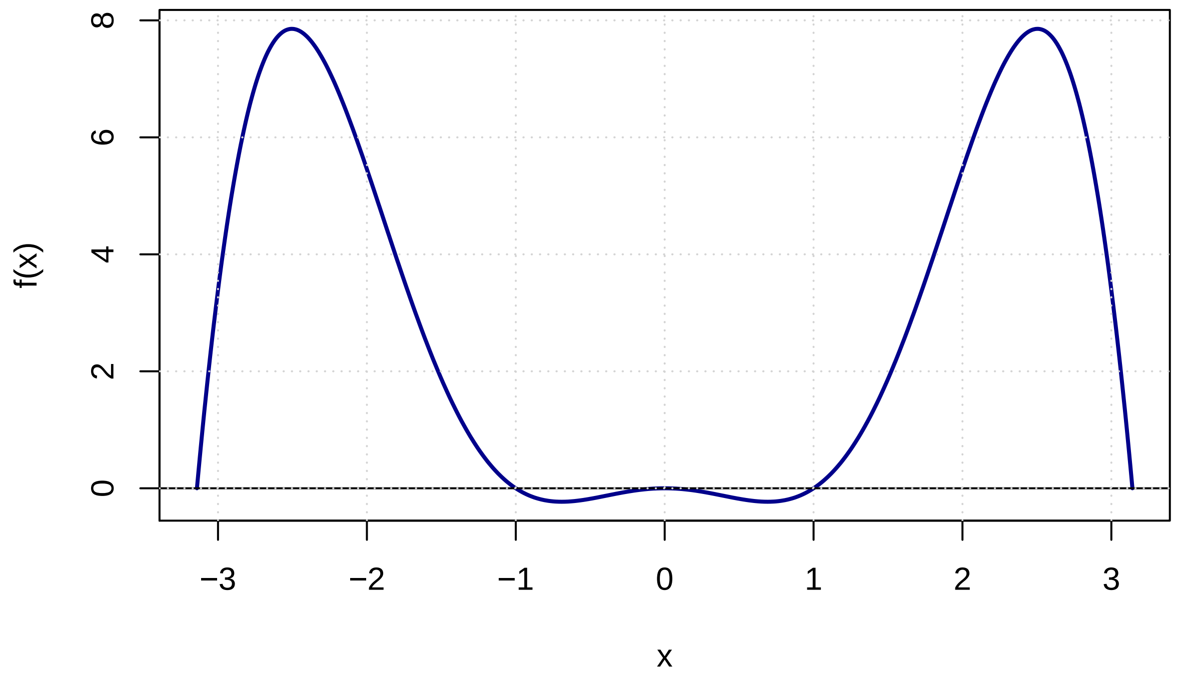
```
## at iteration 1 the root lies between 4 and 4.5
## at iteration 2 the root lies between 4 and 4.25
## at iteration 3 the root lies between 4.125 and 4.25
## at iteration 4 the root lies between 4.1875 and 4.25
## at iteration 5 the root lies between 4.21875 and 4.25
## at iteration 6 the root lies between 4.234375 and 4.25
## at iteration 7 the root lies between 4.242188 and 4.25
## at iteration 8 the root lies between 4.246094 and 4.25
## at iteration 9 the root lies between 4.248047 and 4.25
## at iteration 10 the root lies between 4.248047 and 4.249023
## at iteration 11 the root lies between 4.248535 and 4.249023
## at iteration 12 the root lies between 4.248535 and 4.248779
## at iteration 13 the root lies between 4.248657 and 4.248779
## at iteration 14 the root lies between 4.248718 and 4.248779
```

```
## [1] 4.248749
```

Question 3a

A graph of $f(x) = (x^3 - x)sin(x)$ for $x \in [\pi, \pi]$ is shown below.

```
f<-function(x){
  (x^3-x)*sin(x)
}
x=seq(-pi,pi,length=1e3)
plot(x,f(x),type="l",col="darkblue",lwd=2)
abline(0,0)
grid()
```



Next, the derivative $f'(x)$ and the R-function `fNR` that returns `c(f(x),dfx(x))` are defined.

```
dfx<-function(x){
  (3*x^2-1)*sin(x)+(x^3-x)*cos(x)  # product rule of differentiation
}

fNR<-function(x){
  return(c(f(x),dfx(x)))
}
```

Finally, we initialize a vector with starting values and roots. First, `newtonraphson` is called to fill the vector with roots and later the roots are shown to the user

```
x0=-3:3
root=rep(NA,length(x0))
for(i in 1:length(x0)){
  root[i]=newtonraphson(fNR,x0[i],1e-6)
}
```

```
## At iteration 1 value of x is: -3.16858
## At iteration 2 value of x is: -3.142307
## At iteration 3 value of x is: -3.141593
## At iteration 4 value of x is: -3.141593
## Algorithm converged
## At iteration 1 value of x is: -1.273085
## At iteration 2 value of x is: -1.080559
## At iteration 3 value of x is: -1.010887
## At iteration 4 value of x is: -1.000245
## At iteration 5 value of x is: -1
## Algorithm converged
## Algorithm converged
## Algorithm converged
## Algorithm converged
## At iteration 1 value of x is: 1.273085
## At iteration 2 value of x is: 1.080559
## At iteration 3 value of x is: 1.010887
## At iteration 4 value of x is: 1.000245
## At iteration 5 value of x is: 1
## Algorithm converged
## At iteration 1 value of x is: 3.16858
## At iteration 2 value of x is: 3.142307
## At iteration 3 value of x is: 3.141593
## At iteration 4 value of x is: 3.141593
## Algorithm converged
```

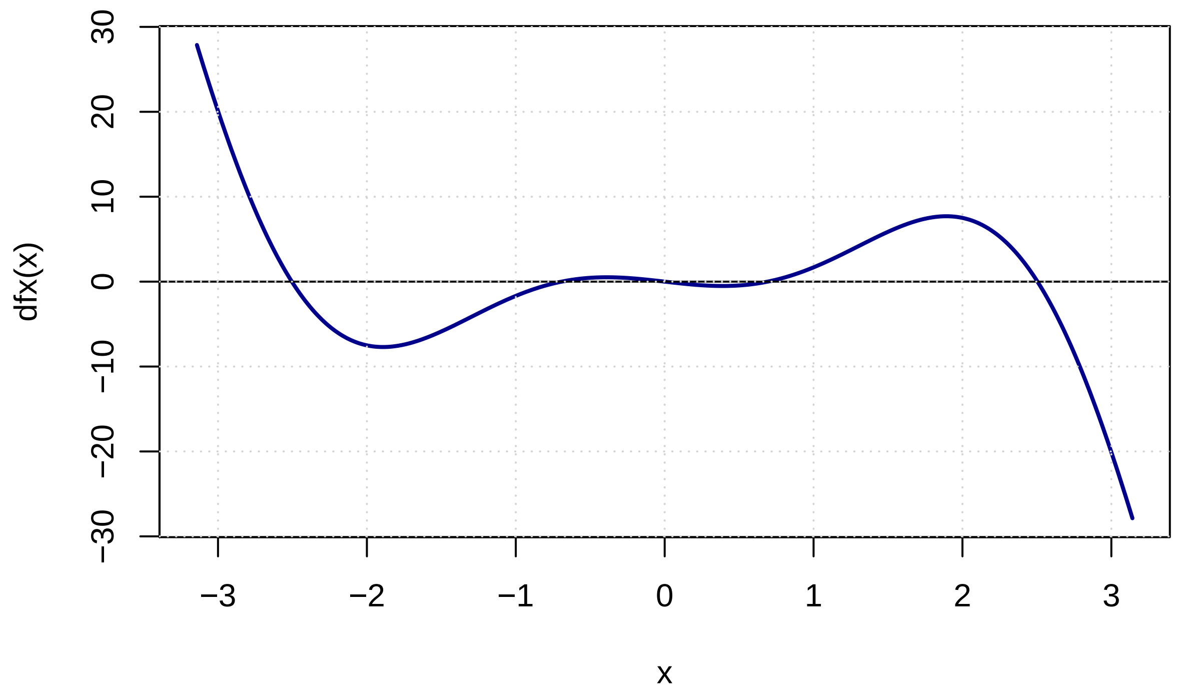
```r
for(i in 1:length(x0)){
  if(is.null(root[i])){
    cat(sprintf("x0=%2d: no root found\n"),x0[i])
  } else {
    cat(sprintf("x0=%2d: Root=%8.5f found\n",x0[i],root[i]))
  }
}
```

```
## x0=-3: Root=-3.14159 found
## x0=-2: Root=-1.00000 found
## x0=-1: Root=-1.00000 found
## x0= 0: Root= 0.00000 found
## x0= 1: Root= 1.00000 found
## x0= 2: Root= 1.00000 found
## x0= 3: Root= 3.14159 found
```

Question 3b

First, a graph of $f'(x)$ is created for the interval $[-\pi, \pi]$.

```
x=seq(-pi,pi,length=1e3)
plot(x,dfx(x),type="l",col="darkblue",lwd=2)
abline(0,0)
grid()
```



Next, *all* roots are determined by the function `uniroot.all`. This results in the values: $0.0000000, -2.5039365, -0.6916089, 0.6916089$ and 2.5039365.

```
#install.packages("rootSolve")
library("rootSolve")
uniroot.all(dfx, c(-4,4))
```

```
## [1]  0.0000000 -2.5039365 -0.6916089  0.6916089  2.5039365
```

When the starting value is 0.7, Newton-Raphson converges to 4π. When the starting value is 2.5, Newton-Raphson converges to -23π.

```
newtonraphson(fNR,0.7,1e-6)
```

```
## At iteration 1 value of x is: 8.434863
## At iteration 2 value of x is: 11.79755
## At iteration 3 value of x is: 13.08257
## At iteration 4 value of x is: 12.58062
## At iteration 5 value of x is: 12.56642
## At iteration 6 value of x is: 12.56637
## At iteration 7 value of x is: 12.56637
## Algorithm converged

## [1] 12.56637
```

```r
newtonraphson(fNR,2.5,1e-6)
```

```
## At iteration 1 value of x is: -70.31823
## At iteration 2 value of x is: -67.98055
## At iteration 3 value of x is: -70.34941
## At iteration 4 value of x is: -67.80078
## At iteration 5 value of x is: -72.38659
## At iteration 6 value of x is: -72.2566
## At iteration 7 value of x is: -72.25663
## At iteration 8 value of x is: -72.25663
## Algorithm converged

## [1] -72.25663
```

These ‘far away’ roots are due to the fact that

- the function has infinitely many roots and
- Newton-Raphson uses $1/f'(x)$; if $f'(x)$ is close to zero, then $1/f'(x)$ becomes very large, such that the next value is very far from the starting value (overshooting).

Question 4

```r
secant <- function(ftn, x0, x1, tol = 1e-9, max.iter = 100) {
  # secant algorithm for solving ftn(x) == 0
  # we assume that ftn is a function of a single variable that returns
  # the function value
  #
  # x0 and x1 are two initial guesses at the root
  # the algorithm terminates when the function value is within distance
  # tol of 0, or the number of iterations exceeds max.iter
  # initialise
  f0 <- ftn(x0)
  f1 <- ftn(x1)
  iter <-  0

  # continue iterating until stopping conditions are met
  while ((abs(f1) > tol) && (iter < max.iter)) {
    if (f0 == f1) {
      cat("Algorithm failed with f0 == f1\n")
      return(NULL)
    }
    x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
    x0 <- x1
    f0 <- f1
    x1 <- x2
    f1 <- ftn(x1)
    iter <-  iter + 1
    cat("At iteration", iter, "approximation is:", x1, "\n")
  }

  # output depends on success of algorithm
  if (abs(f1) > tol) {
    cat("Algorithm failed to converge\n")
    return(NULL)
  } else {
    cat("Algorithm converged\n")
    return(x1)
  }
}
```

Comparing the various methods for finding the roots of $f(x) = cos(x) - x$.

```r
f1 <- function(x){ cos(x)-x }
f1NR <- function(x){ c(cos(x)-x, -sin(x)-1) }
# root of cos(x)-x
secant(f1, 0, 1)
```

```
## At iteration 1 approximation is: 0.6850734
## At iteration 2 approximation is: 0.736299
## At iteration 3 approximation is: 0.7391194
## At iteration 4 approximation is: 0.7390851
## At iteration 5 approximation is: 0.7390851
## Algorithm converged
## [1] 0.7390851
```

```
newtonraphson(f1NR, 0.5)
```

```
## At iteration 1 value of x is: 0.7552224
## At iteration 2 value of x is: 0.7391417
## At iteration 3 value of x is: 0.7390851
## At iteration 4 value of x is: 0.7390851
## Algorithm converged
```

```
## [1] 0.7390851
```

```
bisection(f1, 0, 1)
```

```
## at iteration 1 the root lies between 0.5 and 1
## at iteration 2 the root lies between 0.5 and 0.75
## at iteration 3 the root lies between 0.625 and 0.75
## at iteration 4 the root lies between 0.6875 and 0.75
## at iteration 5 the root lies between 0.71875 and 0.75
## at iteration 6 the root lies between 0.734375 and 0.75
## at iteration 7 the root lies between 0.734375 and 0.7421875
## at iteration 8 the root lies between 0.7382812 and 0.7421875
## at iteration 9 the root lies between 0.7382812 and 0.7402344
## at iteration 10 the root lies between 0.7382812 and 0.7392578
## at iteration 11 the root lies between 0.7387695 and 0.7392578
## at iteration 12 the root lies between 0.7390137 and 0.7392578
## at iteration 13 the root lies between 0.7390137 and 0.7391357
## at iteration 14 the root lies between 0.7390747 and 0.7391357
## at iteration 15 the root lies between 0.7390747 and 0.7391052
## at iteration 16 the root lies between 0.7390747 and 0.73909
## at iteration 17 the root lies between 0.7390823 and 0.73909
## at iteration 18 the root lies between 0.7390823 and 0.7390862
## at iteration 19 the root lies between 0.7390842 and 0.7390862
## at iteration 20 the root lies between 0.7390842 and 0.7390852
## at iteration 21 the root lies between 0.7390847 and 0.7390852
## at iteration 22 the root lies between 0.739085 and 0.7390852
## at iteration 23 the root lies between 0.7390851 and 0.7390852
## at iteration 24 the root lies between 0.7390851 and 0.7390851
## at iteration 25 the root lies between 0.7390851 and 0.7390851
## at iteration 26 the root lies between 0.7390851 and 0.7390851
## at iteration 27 the root lies between 0.7390851 and 0.7390851
## at iteration 28 the root lies between 0.7390851 and 0.7390851
## at iteration 29 the root lies between 0.7390851 and 0.7390851
## at iteration 30 the root lies between 0.7390851 and 0.7390851
## [1] 0.7390851
```

Newton-Raphson is the fastest method using only 4 iterations, while the Secant method needs 1 additional iteration. As expected, the bisection is the slowest method requiring 30 iterations.

```r
f2 <- function(x){ log(x) - exp(-x) }
f2NR <- function(x){ c(log(x)-exp(-x), x^(-1)+exp(-x)) }
# root of log(x)-exp(-x)
secant(f2, 1, 2)
```

```
## At iteration 1 approximation is: 1.39741
## At iteration 2 approximation is: 1.285476
## At iteration 3 approximation is: 1.310677
## At iteration 4 approximation is: 1.309808
## At iteration 5 approximation is: 1.3098
## At iteration 6 approximation is: 1.3098
## Algorithm converged
```

```
## [1] 1.3098
```

```r
newtonraphson(f2NR, 1.5)
```

```
## At iteration 1 value of x is: 1.295082
## At iteration 2 value of x is: 1.30971
## At iteration 3 value of x is: 1.3098
## At iteration 4 value of x is: 1.3098
## Algorithm converged
```

```
## [1] 1.3098
```

```r
bisection(f2, 1, 2)
```

```
## at iteration 1 the root lies between 1 and 1.5
## at iteration 2 the root lies between 1.25 and 1.5
## at iteration 3 the root lies between 1.25 and 1.375
## at iteration 4 the root lies between 1.25 and 1.3125
## at iteration 5 the root lies between 1.28125 and 1.3125
## at iteration 6 the root lies between 1.296875 and 1.3125
## at iteration 7 the root lies between 1.304688 and 1.3125
## at iteration 8 the root lies between 1.308594 and 1.3125
## at iteration 9 the root lies between 1.308594 and 1.310547
## at iteration 10 the root lies between 1.30957 and 1.310547
## at iteration 11 the root lies between 1.30957 and 1.310059
## at iteration 12 the root lies between 1.30957 and 1.309814
## at iteration 13 the root lies between 1.309692 and 1.309814
## at iteration 14 the root lies between 1.309753 and 1.309814
## at iteration 15 the root lies between 1.309784 and 1.309814
## at iteration 16 the root lies between 1.309799 and 1.309814
## at iteration 17 the root lies between 1.309799 and 1.309807
## at iteration 18 the root lies between 1.309799 and 1.309803
## at iteration 19 the root lies between 1.309799 and 1.309801
## at iteration 20 the root lies between 1.309799 and 1.3098
## at iteration 21 the root lies between 1.309799 and 1.3098
## at iteration 22 the root lies between 1.309799 and 1.3098
## at iteration 23 the root lies between 1.3098 and 1.3098
## at iteration 24 the root lies between 1.3098 and 1.3098
## at iteration 25 the root lies between 1.3098 and 1.3098
## at iteration 26 the root lies between 1.3098 and 1.3098
## at iteration 27 the root lies between 1.3098 and 1.3098
## at iteration 28 the root lies between 1.3098 and 1.3098
## at iteration 29 the root lies between 1.3098 and 1.3098
## at iteration 30 the root lies between 1.3098 and 1.3098
```

```
## [1] 1.3098
```

For $f(x) = log(x) - exp(-x)$, the Secant methods needs 2 more iterations than Newton-Raphson. As before, the bisection method requires 30 iterations, which was to be expected since the tolerance (1e-09) and the width of the starting interval (1) are the same!

Question 5

The roots can be determine using:

```
polyroot(c(1,1,1,1))
```

```
## [1]  0+1i -1+0i  0-1i
```

```
polyroot(c(-4,-2,1,0,0,1))
```

```
## [1]  0.381244+1.419562i -1.067248+0.458702i -1.067248-0.458702i
## [4]  1.372006-0.000000i  0.381244-1.419562i
```

Hence, we can factorize the two polynomials as

$$x^3 + x^2 + 1 = (x - \mathbb{i})(x + \mathbb{i})(x + 1)$$

and

$$\begin{aligned} x^5 + x^2 - 2x - 4 = & (x - 1.372006)(x + 1.067248 + 0.458702\mathbb{i})(x + 1.0672483 - 0.458702\mathbb{i}) \\ & (x - 0.381244 + 1.419562\mathbb{i})(x - 0.381244 - 1.419562\mathbb{i}) \end{aligned}$$

with $\mathbb{i}^2 = -1$.