Exercises

1. Write down the binary expansion of 0.55, 0.2, 0.35, 19, 1024, 105.3.

2. Find a function and starting values such that Newton's method and Secant method don't converge.

3. The amount of time taken by an algorithm is an important consideration. Consider for example the computation of the determinant of a 100×100 matrix by expanding along the first row and evaluating determinants of 99×99 matrices, each of which are then expanded along their first rows and determinants of 98×98 matrices are evaluated, etc. If each multiplication of two numbers and addition of two numbers is an *operation*, roughly how many operations are needed for computing the matrix. Considering that today's fastest parallel computers are capable of at most 10^{15} operations per second, how much time would it take to compute the determinant of the matrix?

4. Let `x<-c(1,2,3)` and `y<-c(0,0,0)`. Try `z<-y+2*x^2`.

5. We shall examine 3 algorithms to evaluate polynomials. We shall represent a polynomial by the vector of their coefficients. For example the vector `c(30,-19,-15,3,1)` represents the polynomial $x^4 + 3x^3 - 15x^2 - 19x + 30$. We would like to evaluate polynomials at different points given in a vector `x`. Consider the following function:

```
naivepoly<-function(x,coefs){
  y<-rep(0, length(x))
  for (i in 1:length(coefs)) {
    y<-y +coefs[i]* (x^ (i-1))
  }
  return(y)
}
```

How many operations (that is, additions and mulitiplications) does the above function perform?

Next consider the following different algorithm:

```
betterpoly<-function(x, coefs){
  y<-rep(0, length(x))
  cached.x<-1
  for (i in 1:length(coefs)) {
    y<-y +coefs[i]* cached.x
    cached.x<-cached.x *x
  }
  return(y)
}
```

What does the above function do? How many operations does the function perform?

Now consider Horner's method for evaluating polynomials. This involves the following iteration:

$$\begin{aligned} f(x) &= a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} + a_n x^n \\ &= a_0 + x(a_1 + \cdots + a_{n-1}x^{n-2} + a_n x^{n-1}) \\ &= a_0 + x(a_1 + \cdots + x(a_{n-1} + x(a_n)) \cdots) \end{aligned}$$

This can be implemented as follows:

```
horner<-function(x, coefs){
  y<-rep(0, length(x))
  for (i in length(coefs):1) {
    y<-x*y +coefs[i]
  return(y)
}
```

How many operations does the above function perform?

6. For a sequence (a_n) of reals and another sequence (b_n) of positive reals, we say $a_n = O(b_n)$ if there is a $C > 0$ such that $|a_n| \leq Cb_n$. The O notation is used very often to express how fast an algorithm is. If an algorithm has $2n^2$ operations (n here is some parameter) then we would say that it takes $O(n^2)$ time. The constant 2 is not very important; in any case the actual time it takes to implement the algorithm will also depend on the speed of the computer.

   Similarly for $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}_+$ we say $f(x) = O(g(x))$ as $x \to \infty$ if there is $C > 0$ and $M > 0$ such that $|f(x)| \leq Cg(x)$ for all $x > M$. For fixed $a \in \mathbb{R}$ we say $f(x) = O(g(x))$ as $x \to a$ if there is $C > 0$ and $\delta > 0$ such that $|f(x)| \leq Cg(x)$ for all $x \in (a - \delta, a + \delta)$.