MAS115: R programming
Lecture 5: Writing your own functions

Sofia Karadimitriou

The University of Sheffield
School of Mathematics and Statistics

Aims

Introduce ideas of creating your own functions in **R** to help solve problems

- Basic syntax of functions
- Why functions are useful

More pseudo-code to solve some big problems:

- Newton-Raphson algorithm

Functions in **R**

- You have seen lots of functions already in **R** e.g.

  `sum()`, `rnorm()`, `sample()`, `plot()`, `hist()`
- Sometimes, if you are planning on using the same code several times you might want to write your own.
- Functions simply act as wrappers for lines of code.
- Once created, you can then just call your function like you do the built-in ones
- Helps clarity of your coding and you to think about how to split a problem up into smaller parts.

An example function

```
# Surface area and volume of a cylinder
# Inputs: r radius of cylinder, l length of cylinder
# Outputs: List of area and volume of cylinder
areacyl.func <- function(r, l = 1)
{
vol <- pi*r^2*l
area <- 2*pi*r^2 + (2*pi*r)* l
retlist <- list(area = area, vol = vol)
return(retlist)
}
```

This function will work out the area and volume of a cylinder and return both values in a list.

An example function - Explanation

- `areacyl.func`
  Can give a function any name that you wish, can then run the function using this name.
- `<- function(r, l = 1)`
  Tells **R** you are creating a function. Need to list the arguments (*inputs*) that you are going to pass to the function. These may have default values
- `{....}`
  The wrapper where you write the block of commands that you wish the function to perform.
- `return()`
  The output that you would like to be returned by the function. Can be a single number or a list if we want to return multiple objects.
  Formally **R** *will return the last evaluation so the* `return()` *is not needed but keeping it makes understanding simpler.*

Why use functions?

- Using functions helps you break up your big problem into smaller problems.
- Each sub-problem might have it's own function to solve it
- These sub-problems might be split up into even smaller problems with their own functions
- Eventually, by breaking down your problem iteratively, you get down to really simple tasks which use the built in functions.
- If any function is longer than about 1/2 page then probably should have split it up into smaller sub-functions

What's the best way to write them?

- To write functions it is often simplest to write the code you wish to implement as an R script. You can check it works as you would wish. You can then add the function wrapper afterwards with the required arguments

Pseudo-Code
The Newton-Raphson Algorithm

Newton-Raphson Algorithm

Newton-Raphson Algorithm is an iterative scheme to find the root of an equation i.e. solve

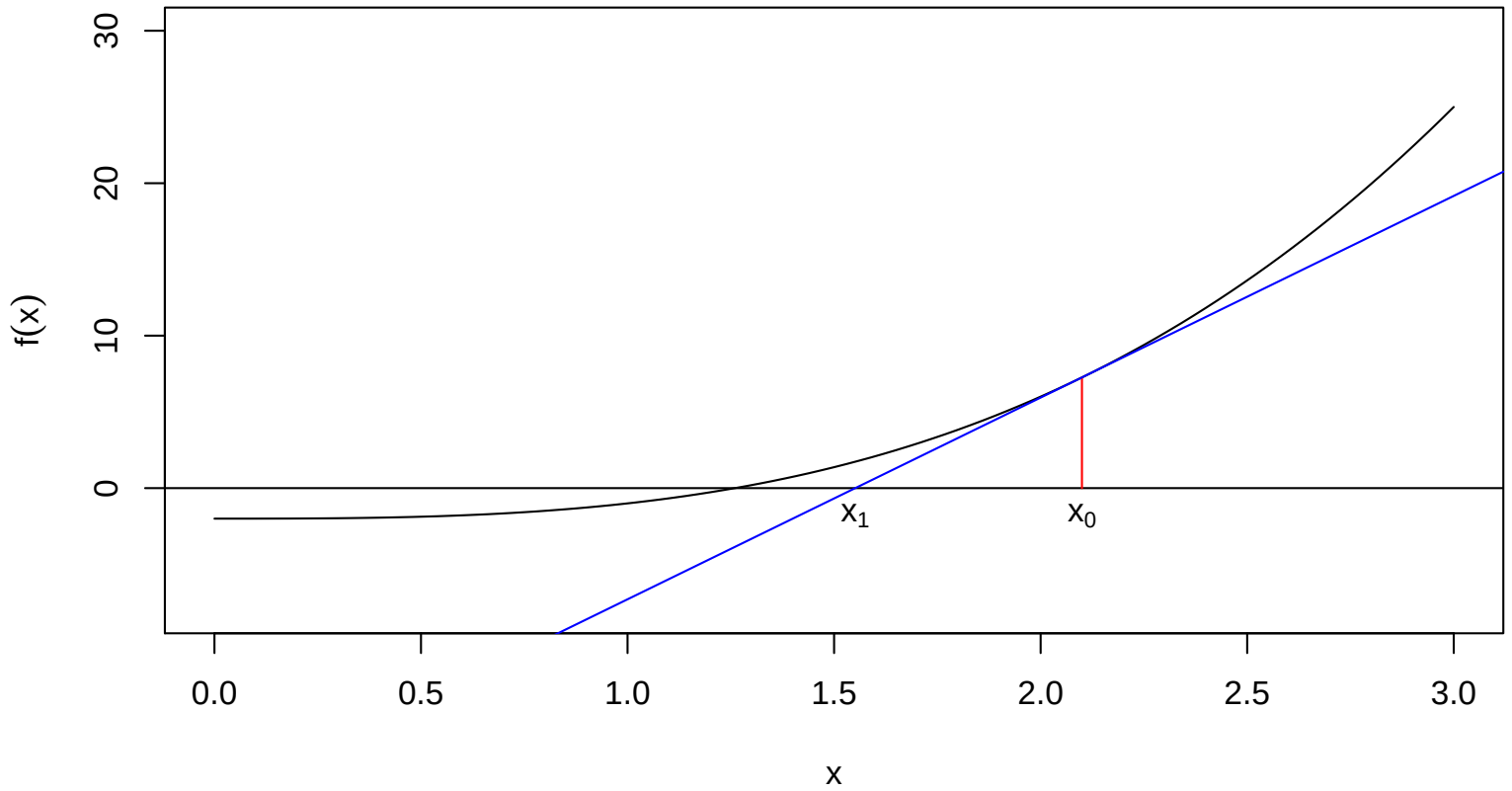
$$f(x) = 0.$$

Given an initial starting point x_0 it generates a sequence of numbers iteratively using the formula

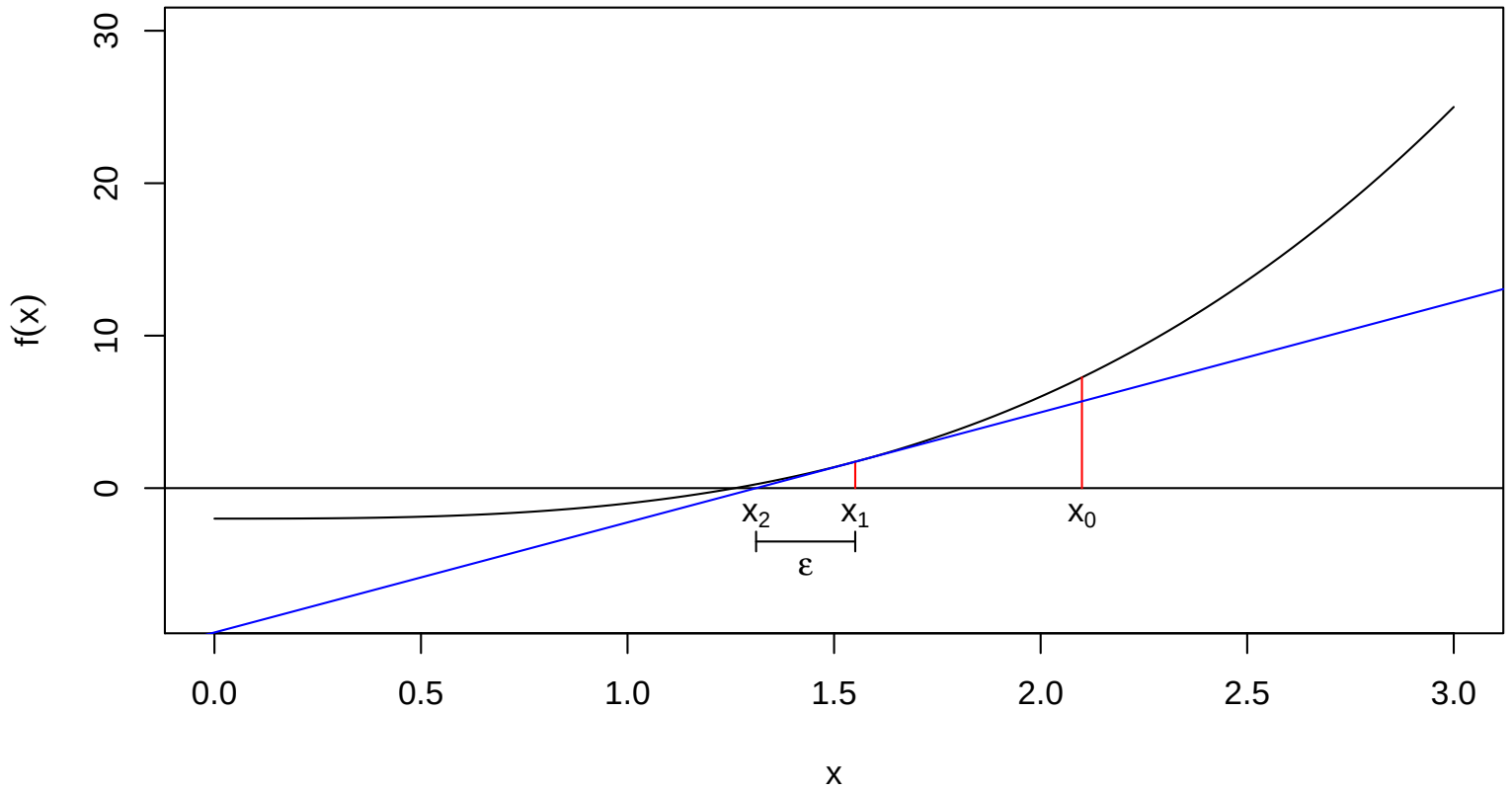
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

With a good initial choice for x_0 this sequence will converge to a fixed point which solves $f(x) = 0$.

Newton-Raphson Visually



Newton-Raphson Visually



Newton-Raphson Formally — Taylor Series

We can expand $f(x)$ around the point $x_0 + \epsilon$ as

$$f(x_0 + \epsilon) = f(x_0) + f'(x_0)\epsilon + \frac{1}{2}f''(x_0)\epsilon^2 + \dots$$

or to first order

$$f(x_0 + \epsilon) \approx f(x_0) + f'(x_0)\epsilon. \tag{1}$$

Now if we want a better estimate for the root we set $f(x_0 + \epsilon) = 0$ in (1) and rearrange to find

$$\epsilon = -f(x_0)/f'(x_0)$$

We can then update our estimate for the root to

$$x_1 = x_0 + \epsilon = x_0 - f(x_0)/f'(x_0)$$

Newton-Raphson: When to stop

If we choose a sensible initial starting point x_0 then algorithm will converge to the root of interest. However can be unstable if we start near a horizontal asymptote or a local extremum.
Normally run the algorithm until one of two things happens:

- Judged to have converged i.e. $|x_{i+1} - x_i| = |\epsilon_i| < \texttt{tol}$.
- Have done more than `maxit` iterations through the algorithm.

If the latter condition happens first then we say that the algorithm has not converged.
Would typically choose the tolerance `tol` to be very small and `maxit` to be quite large.

A problem

Given a function $f(\cdot)$ and pre-specified values for:

- `x` — the starting point for the algorithm,
- `maxit` — the maximum number of iterations to perform,
- `tol` — the tolerance used to judge convergence,

write pseudo-code to perform the Newton-Raphson algorithm until one of the two stopping conditions is reached.

Hint: Which kind of loop would be most suitable for this algorithm?

Pseudo-code for the Newton-Raphson algorithm

We can write the following pseudo-code:

```
# Pseudo-code for Newton-Raphson
Initialise with value x
Set iteration = 0
Set eps = infty
WHILE (iteration < maxit and |eps| > tol)
      eps = f(x)/f'(x)
      x = x - eps
      iteration = iteration + 1
ENDWHILE
IF(iteration not equal to maxit)
      THEN converged
      ELSE has not converged
ENDIF
```

Conclusion

- Functions allow us to create wrappers for parts of code that we may want to reuse
- Very useful in helping us to split up big problems into smaller ones:
  - Each sub-problem might have a separate function to solve it
  - These sub-problems might themselves be broken into smaller problems each with their own function
- Newton-Raphson algorithm to find roots of equations and how to implement using pseudo-code.

In the lab class today we will be learning more about the functions and using the Newton-Raphson algorithm to find an estimate of $\sqrt{2}$.