9. Writing Functions

Ken Rice
Tim Thornton

University of Washington

Seattle, July 2013

In this session

One of the most powerful features of R is the user's ability to expand existing functions and write custom functions. We will give an introduction to writing functions in R.

- Structure of a function
- Creating your own function
- Examples and applications of functions

Introduction

Functions are an important part of R because they allow the user to customize and extend the language.

- Functions allow for reproducible code without copious/error prone retyping
- Organizing code into functions for performing specified tasks makes complex programs tractable
- Often necessary to develop your own algorithms or take existing functions and modify them to meet your needs

Structure of a function

Functions are created using the function() directive and are stored as R objects.

Functions are defined by

1. a function name with assignment to the function() directive (function names can be almost anything. However, the usage of names of existing functions should be avoided.)

2. the declaration of arguments/variables

3. and the definition of operations (the function body) that perform computations on the provided arguments.

Structure of a function

The basic structure of a function is:

```
my.func <- function(arg1,arg2,arg3, ...) {
 commands;
 return(output);
}
```

Function arguments (arg1, arg2, ...) are what is passed to the function and used by the function's code to perform calculations.

Calling a function

Functions are called by their name followed by parentheses containing possible argument names.

A call to the function generally takes the form

```
my.func(arg1=expr1,arg2=expr2,arg3=exp3, ...)
```

or

```
my.func(expr1,expr2,exp3, ...)
```

Arguments can be "matched" by name or by position.

A function can also take no arguments, and in this case the function is called with the name of the function with empty parenthesis () after the function name.

Typing just the function name without parentheses will print the definition of a function.

Function body

The actual expressions (commands/operations) are defined in the body of the function.

The function body appears within {curly brackets}. The brackets {} are not required for functions with just one expression.

Individual commands/operations are separated by new lines or semicolons.

An object is returned by a function with the `return()` command, where the object to be returned appears inside the parentheses.

If the end of a function is reached without calling `return`, the value of the last evaluated expression will be returned by the function.

Variables that are created inside the function body exist only for the lifetime of the function. Thus, they are not accessible outside of the function in an R session.

Example: returning a single value

Below is a function for calculating the coefficient of variation (the ratio of the standard deviation to the mean) for a vector.

```
coef.of.var <- function(x){
   meanval <- mean(x,na.rm=TRUE)
   sdval   <- sd(x,na.rm=TRUE)
   return(sdval/meanval)
   }
```

We can apply this function to obtain the coefficient of variation for the daily ozone concentrations in New York, summer 1973:

```
data(airquality)
coef.of.var(airquality$Ozone)
```

```
> coef.of.var(airquality$Ozone)
[1] 0.7830151
```

Example: returning multiple values

A function can return multiple objects/values by using `list()` – which collects objects of (potentially) different types.

The function below calculates and returns the maximum likelihood estimates for the mean and standard deviation for a numeric vector under a normal distribution assumption

```
gaussian.mle <- function(x) {
n <- length(x)
mean.est <- mean(x,na.rm=TRUE)
var.est <- var(x,na.rm=TRUE)*(n-1)/n
est <- list(mean=mean.est, sd=sqrt(var.est))
return(est)}
```

Example: returning multiple values

We can apply the `gaussian.mle` function to the daily ozone concentrations in New York data:

```
> results<-gaussian.mle(airquality$Ozone)
> attributes(results)  #list the attributes of the object returned
$names
[1] "mean" "sd"
> results$mean
[1] 42.12931
> results$sd
[1] 32.8799
```

Elements of lists can also be obtained using *double* square brackets, i.e. `results[[1]]`.

Declaring functions within functions

Functions can be declared and used inside a function.

```
square.plus.cube <- function(y) {
   square <- function(x) { return(x*x) }
   cube <- function(x) { return(x^3) }
   return(square(y) + cube(y))
 }
```

```
> square.plus.cube(4)
[1] 80
```

Example: function returning a function

A function can also return another function as the return object.

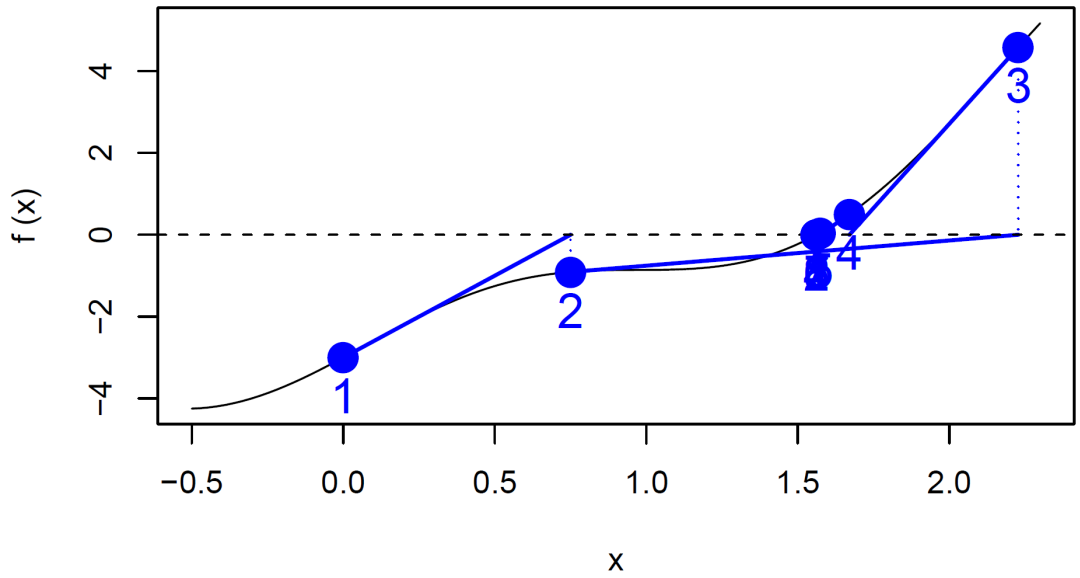
```
make.power <- function(n){
   pow <- function(x) {x^n}
   pow
   }

cube <- make.power(3)
square <- make.power(2)

> cube(3)
[1] 27
> square(3)
[1] 9
```

Example: functions as arguments

Functions can take other functions as arguments. This is helpful with finding *roots* of a function; values of x such that $f(x) = 0$.

With the Newton-Raphson method, a root can be found by the following iteration procedure until convergence:

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



Example: functions as arguments

This function implements the Newton-Raphson method, given input of arguments, a place to start, and convergence tolerance:

```
newton.raphson <- function(f,fprime,x0,thresh){
   myabsdiff <- Inf
   xold      <- x0
   while(myabsdiff>thresh){ # have we converged yet? If no, move;
      xnew      <- xold-f(xold)/(fprime(xold))
      myabsdiff <- abs(xnew-xold)
      xold      <- xnew
   }
return(xnew)
}
```

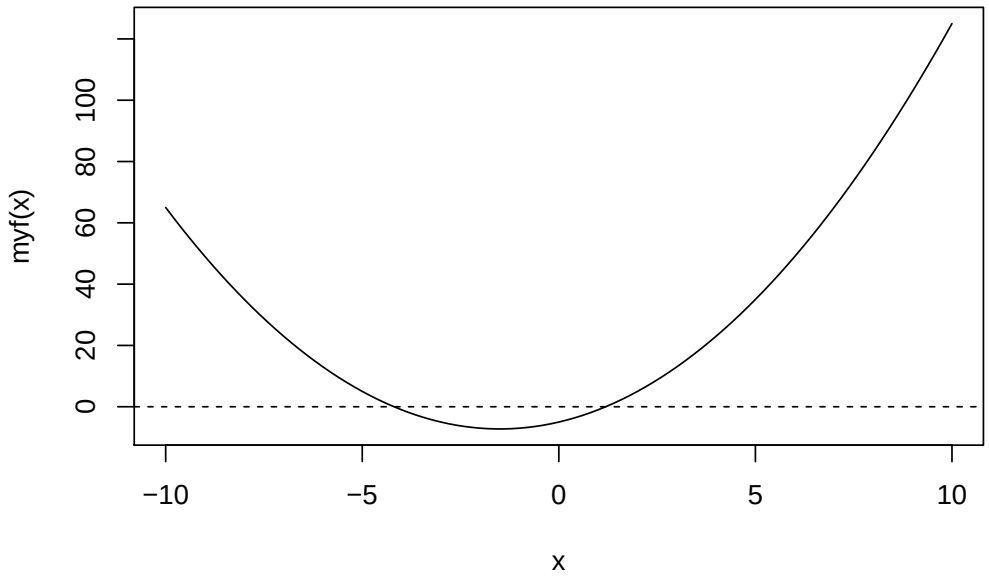
Example: functions as arguments

We'll find the roots of $f(x) = x^2+3x-5$, using Newton-Raphson. We need the derivative of $f(x)$: $f'(x) = 2x + 3$

```
myf <- function(x){ x^2+3*x-5 }
myfprime<-function(x){ 2*x+3 }
```

We use the `newton.raphson()` function with initial value of 10 and a convergence threshold of 0.0001 to obtain a root:

```
> newton.raphson(f=myf,fprime=myfprime,x0=10,thresh=0.0001)
[1] 1.192582
```



Tips for writing functions

- Avoid rewriting the same code...use functions!
- Modularize as much as possible: functions calling other functions
- Provide documentation, including detailed comments describing the procedures being conducted by the functions, especially for large, complex programs
- Test your functions: use data/arguments for which you know the results to verify that your functions are working properly
- Use meaningful variable and function names

Summary

- User-defined functions can easily be created in `R` with function(argument list)

- Arguments of a function are allowed to be practically any `R` object including lists, numeric vectors, data frames, and functions

- In functions calls, arguments are matched by name or by position

- An object can be returned by a function with `return()`. If `return()` is not invoked, the last evaluated expression in the body of a function will be returned.

- `list()` can be used for returning multiple values