MAS115 R programming 2016-17

Lab Class 5
Homework Solutions

1 Calculation of population variance

(a) Use the function as given, with argument `1:6` or `c(1,2,3,4,5,6)` or similar.

```
# Find the 'empirical' (or 'population' or 'uncorrected') variance of a vector
EmpVar <- function(x)
{
  n <- length(x) # Find the length of x
  return (sum(x^2) - n * mean(x)^2) / n) # Calculate and return empirical variance
}
> EmpVar(1:6)
[1] 2.916667
```

(b) There are many ways to add this functionality. With only two options, a logical or boolean argument is probably the easiest. A character-valued argument is possible; R has some built-in tools to help with more complicated cases—see e.g. the help page for `charmatch`.

```
# Find either the empirical OR the usual sample variance of a vector
EitherVar <- function(x, SampleVar=FALSE)
{
  if(SampleVar)
  {
    v <- var(x)
  } else
  {
    n <- length(x) # Find the length of x
    v <- (sum(x^2) - n * mean(x)^2) / n
  }
  return(v)
}
> EitherVar(1:6)
[1] 2.916667
> EitherVar(1:6,SampleVar=TRUE)
[1] 3.5
> EitherVar(1:6,TRUE)
[1] 3.5
```

(c) In theory, the value should be unaffected by adding a constant to all elements of the vector. See the following example (note that the inner parentheses are not necessary, but help clarify what is being passed to the function).

```
> EmpVar((1:6)+10^6)
[1] 2.916667
```

When adding larger constants i.e. larger values of m, the two terms involved in the subtraction can become so large compared with the difference that numerical accuracy is completely lost. For example:

```
> EmpVar((1:6)+10^7)
[1] 2.916667
> EmpVar((1:6)+1*10^8)
[1] 2.666667
> EmpVar((1:6)+2*10^8)
[1] -5.333333
```

(d) To fix this, we could calculate the empirical variance without forming such large quantities along the way.

```
EmpVar2 <- function(x)
{
  n <- length(x) # Find the length of x
  mu <- mean(x) # find the mean of x
  return (sum((x-mu)^2) / n) # Calculate and return empirical variance
}
> EmpVar2(1:6+10^9)
[1] 2.916667
```

Alternatives would be so pre-process `x`, by shifting to more 'reasonable' values without affecting the correct answer, or to use the built-in `var` to do the main work and then adjust the answer afterwards.

2 Newton-Raphson Algorithm

My Newton-Raphson code is as follows - note again that I have added useful comments with the # key so the reader know what the functions does. I have also added some lines which write to the screen while it is running just to make it a little easier to understand.

```
####### Newton-Raphson #######
# This code will implement Newton-Raphson method
```

```
# Define function to solve = 0
f1.func <- function(x) {
  x^2 - 2
}

# Define derivative of function
f1deriv.func <- function(x) {
  2*x
}

# Performs Newton-Raphson algorithm to solve equation funct = 0
# Input: x = starting point, funct = function to find zeros of,
#        deriv = function to calculate derivative,
#        maxit = maximum iteration number, tol = when to stop
# Output: x = estimate, iter = number iterations, converged = whether converged

myNR.func <- function(x, funct, deriv, maxit = 100, tol = 0.000001) {
  eps <- Inf
  iteration <- 0
  while((iteration < maxit) && (abs(eps) > tol)) {
    iteration <- iteration + 1
    f <- funct(x)
    df <- deriv(x)
    eps <- f/df
    x <- x - eps
    cat("Iteration = ", iteration, "Root estimate = ", x, "\n")
  }
  return(list(x = x, iter = iteration, converged = (iteration < maxit)))
}
```

When I run this code I find that it converges extremely quickly to $\sqrt{2}$ as can be seen with

```
> myest <- myNR.func(2,f1.func,f1deriv.func)
Iteration =  1 Root estimate =  1.5
Iteration =  2 Root estimate =  1.416667
Iteration =  3 Root estimate =  1.414216
Iteration =  4 Root estimate =  1.414214
Iteration =  5 Root estimate =  1.414214
> myest
$x
[1] 1.414214
```

```
$iter
[1] 5
$converged
[1] TRUE
```

Solving $\cos(x) = x$

To find the solution to $\cos(x) = x$ we just need to change the function arguments passed to the main function. We want to find the root of

$$f(x) = \cos(x) - x$$
$$\Rightarrow f'(x) = -\sin(x) - 1$$

so we define functions

```
cosminusx <- function(x) {
  cos(x) - x
}

cosderiv <- function(x) {
  -sin(x) - 1
}
```

Running this code we find the answer to be $x \approx 0.739$:

```
> myest <- myNR.func(1,cosminusx,cosderiv)
Iteration =  1 Root estimate =  0.7503639
Iteration =  2 Root estimate =  0.7391129
Iteration =  3 Root estimate =  0.7390851
Iteration =  4 Root estimate =  0.7390851
> myest
$x
[1] 0.7390851

$iter
[1] 4

$converged
[1] TRUE
```