MAS115 R programming 2016-17
Lab Class 3
Homework Solutions

Introduction

There are two ways to solve these problems. The first relies on loops while the second uses the fact that **R** is a vectorised language. As a consequence of **R**'s ability to operate on vectors elementwise (and the use of `apply` on matrices) many loops can be done away with and written as a single line of code.

Picky programmers would suggest that you should write the code using as few loops as possible and take full advantage of **R**'s vectorisation. This is because looping in **R** can be quite slow whereas using vectorisation is very fast. However, at this stage, it is much more important that you get the correct answer than write optimal code which runs faster.

For this homework whichever approach (looping or not) you have taken doesn't matter. You should however have a look at the vectorised code to understand how it works and appreciate the benefits that **R** provides.

You should also try running both versions of the solution. Can you observe a difference in the run time for each? The vectorised version should run much quicker than the `for` loop version. This is why generally we try to get rid of all unnecessary loops in **R**.

1 Estimating π (Task 4.1.1)

Using loops

Using a `for` loop we can simply create 10,000 points and see how many lie within the unit square:

```
## for loop version ###
N <- 10000 # Choose how many points to create
inside <- rep(NA, N) # Vector to store whether each point is inside circle
for(i in 1:N) {
  x <- runif(1) # Create random points
  y <- runif(1)
  inside[i] <- ((x^2 + y^2) < 1) # Check if inside circle by testing d^2 < 1^2
                                 # where d is Euclidean distance from the centre
}
# Find estimate for pi/4
est1 <- mean(inside) # mean of a vector of logicals is mean of 0s and 1s
                     # same as counting points inside and dividing by N
# Find estimate for pi
estpi <- 4*mean(inside)
```

With vectorisation

We don't however need the `for` loop at all as we could create all the points together. Make sure you understand what this code is doing:

```
## Vectorised version
N <- 10000 # Choose how many points to create
# Create vectors of N x coord & y coords
x <- runif(N)
y <- runif(N)
inside <- ( x^2 + y^2) < 1 )

# Find estimate for pi/4
est1 <- mean(inside)

# Find estimate for pi
estpi <- 4*mean(inside)
```

2 Finding prime numbers (Task 4.2.2)

Testing an individual number

To test whether an individual number is prime I wrote the following code

```
##############################
# Prime Numbers
##############################

n <- 5
# Initialise by setting Prime as TRUE
prime <- TRUE
# Now loop through and see if any number divides
for(i in 2:floor(sqrt(n))) {
# Deal with the special case when n = 2,3
  if(n == 2 || n == 3) break
# Otherwise test if i divides n
  if((n %% i) == 0) {
    prime <- FALSE
    break
  }
}
```

Note that to deal with the numbers 2 and 3 I have added a separate check just testing for those two numbers `if((n == 2 || n == 3)) break` guaranteeing that they return with with `prime` being `TRUE`.

All primes up to 10000

Now to create a list of all the prime numbers between 2 and 10000, I will create a vector called `pvec` to store them in. I will initialise this vector to store the first primes (2 and 3) before checking each number from 4 to 10000 for "primeness". If I find any number `i` to be prime then I will append it to my vector of primes `pvec` using the command `c(pvec, i)`. This is shown below:

```
# All the prime numbers from 2 to 10000

# Create pvec (which will store prime numbers)
# and initialise with known values 2 and 3
pvec <- c(2,3)

# Now try each number n in turn from 4 to 10000
# running code to test each as written above
for(n in 4:10000) {
# Initialise by setting Prime as TRUE
  prime <- TRUE
# Now loop through and see if any number divides
# As our loop starts as n = 4 we don't need to worry
# about the special case when n = 2,3
  for(i in 2:floor(sqrt(n))) {
    if((n %% i) == 0) {
      prime <- FALSE
      break
    }
  }
  if(prime) pvec <- c(pvec, n)
}

length(pvec) # There should be 1229 primes less than 10000
pvec[1000]   # The 1000th prime should be 7919
```