MAS115 R programming, Homework Solutions 1

Prof. P. G. Blackwell

2017-18

Q1 - Gamma random variables

First, we ensure our results are reproducible.

```
# We make sure we are going to be using the same values by calling
# the set.seed() command. This initialises our random number generator
# at the same value so our experiment is repeatable and we get the
# same random numbers
set.seed(1)
```

(a) The command

```
rgamma(10, 1 , 5)
```

```
 [1] 0.03102827 0.37648032 0.36090250 0.16723553 0.24450873 0.23167105
 [7] 0.19800399 0.06147466 0.01892382 0.03144031
```

will produce 10 random variables from the gamma distribution with shape $a = 1$ and **rate** $s = 5$. If you looked at the help file the order that the arguments to `rgamma` are passed is

```
rgamma(n, shape, rate = 1, scale = 1/rate)
```

If you do not name the arguments to the function then R will fill them in using the order given in the help file. **It is generally therefore best to use the argument names given when specifying variables to pass to a function.**

Having looked at the help file you should have seen that to specify the scale parameter, there is an argument called `scale` that you can give, instead of the rate. To generate 10000 numbers from the gamma distribution with shape parameter 2 and scale 4 the easiest way is to write

```
# Produce 10000 numbers from a gamma(2,4)
x <- rgamma(10000, shape = 2, scale = 4)
```

(b) In order to find the mean and the standard deviation you use the commands

```
mean(x)
```

```
[1] 8.076132
```

```
sd(x)
```

```
[1] 5.693396
```

(c) To find the mean of all the entries in `x` which are strictly greater than 2.5 we use the command

```
mean(x[x > 2.5])
```

```
[1] 9.026721
```

This can look intimidating so to explain we'll split it into its several steps. This is always worth doing with a complicated expression. We can try it out with a smaller number (maybe 10 or so) of gamma variables and observe what each step gives.

```
(y <- rgamma(10, shape = 2, scale = 4))
```

```
 [1]  4.4391389  5.4294473  7.2992195  1.1632820  0.9595559 14.1425164
 [7]  2.0948830  9.8873455  9.6607838  8.8449832
```

```
# Can create a logical condition to check if each value of y is > 2.5
#(vector of TRUE and FALSE values)
y > 2.5
```

```
 [1]  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE
```

```
# Use this logical condition to select the values
# i.e just select the x's which have a TRUE in the logical above
y[y > 0.5]
```

```
 [1]  4.4391389  5.4294473  7.2992195  1.1632820  0.9595559 14.1425164
 [7]  2.0948830  9.8873455  9.6607838  8.8449832
```

```
# Find out the mean of these values
mean(y[y > 2.5])
```

```
[1] 8.529062
```

(d) The command `sum(x > 2.5)` counts the number of random variables which are greater than 2.5. Again, the inner part just creates a boolean or logical vector (i.e. a vector with values `TRUE` or `FALSE`). If you apply a 'numerical' operator, e.g. `sum`, to boolean variables, then R will automatically (and silently!) convert `TRUE` to 1 and `FALSE` to 0 before carrying out the calculation. Try it with some other operators!

```
# Count the number of gamma random variables (out of the 10000)
# which are larger than 2.5
sum(x > 2.5)
```

```
[1] 8725
```

Q2 - Estimating π

(a) The distance from the centre of the toothpick and the nearest notepad line is $U[0, 2]$ since the toothpick has equal likelihood of landing at any one point. Hence the vertical distance up the page of its centre is also uniformly distributed. Its centre can't be more than 2cm away from the nearest line and so `CentDist` $\sim U[0, 2]$.

To create 10000 such distances use the following commands.

```
set.seed(13) # set.seed() takes as an argument any integer
nsims <- 10000
CentDist <- runif(nsims, 0, 2)
```

(b) The distribution of the angle from the vertical is $U[0, \frac{\pi}{2}]$. We can sample some values using

```
# Angle from the vertical is Uniform[0, pi/2]
Angles <- runif(nsims, 0, pi/2)
```

(c) Consider a toothpick of length 2 at an angle θ. If we draw a vertical line going through its centre then trigonometry (see the figure in the question) tells us that the ends of the toothpick will be $\cos\theta$ vertically above or below its centre. If this is greater than the distance to the nearest line then the toothpick will cross it. Hence it will cross if and only if

$$x < \cos(\theta)$$

We can count the number of times our simulated throws cross the nearest line (and hence the proportion) by using a logical filtering. It was given in the question that this will tend to $\frac{1}{\pi}$ so we can see how close we get too. We will first create a vector `d` and then compare it to the sampled distances to the nearest line

```
# For toothpick to touch we need CentDist < cos(Angles)
d <- cos(Angles)
ntouch <- sum(CentDist < d)
ptouch <- ntouch/nsims
ptouch
```

```
[1] 0.32
```

```
# We can then check to see if this is 1/pi
1/ptouch
```

```
[1] 3.125
```

(d) Remember that $X \sim U[0,2]$ and $\Theta \sim U[0, \frac{\pi}{2}]$. We will work out the answer by conditioning on the value of Θ observed:

$$\begin{aligned} P\{X < \cos(\Theta)\} &= \int_0^{\pi/2} P\{\mathbf{X} < \cos(\Theta) | \Theta = \theta\} \, p_\Theta(\theta) d\theta \\ &= \int_0^{\pi/2} P\{\mathbf{X} < \cos(\theta)\} \, p_\Theta(\theta) d\theta \\ &= \int_0^{\pi/2} \left\{ \frac{\cos(\theta)}{2} \right\} \frac{2}{\pi} d\theta \\ &= \frac{1}{\pi} \int_0^{\pi/2} \cos(\theta) d\theta \\ &= \frac{1}{\pi}. \end{aligned}$$