Random processes STP855
University of Halmstad
Examiner: Eric Järpe

Tips for laboration about simulation

The laboration is divided into two parts: one theoretical and one practical.

In the course literature it says on pages 55–56 that the probability density function of a variable which is Rayleigh distributed with parmeter σ, is

$$f_A(a) \;=\; \frac{a}{\sigma^2}\, e^{-a^2/(2\sigma^2)} \;,\; a \geq 0.$$

Theoretical part

Using the above density function one can show that

$$P(A \leq a) \;=\; \int_0^a f_A(x)\, dx \;=\; 1 - e^{-a^2/(2\sigma^2)}$$

Exercise 1 is to perform this integration by making an apropriate substitution.

Exercise 2 is to calculate the inverse of the distribution function $F(a) = P(A \leq a)$.
Hint: let $u = F(a), a > 0$ and solve the equation with respect to a.

For exercise 3 one shall show how to use the inverse of the distribution function for simultating observations of a Rayleigh distributed random variable, i.e. show that
$\mathcal{U} \in U(0,1) \;\Rightarrow\; P(F^{-1}(\mathcal{U}) \leq a) = \ldots = F(a)$.

For exercise 4, see the course literature, page 55.

This is the theory part. I am grateful for short but clear proofs. Also notice when there are certain conditions (like $a \geq 0$, $0 \leq u \leq 1$ and so on).

Practical part

The statistics software R is available for free both for linux/UNIX and for Windows. It can be downloaded from `http://cran.r-project.org/`. Once installed, to start R, write: `R <enter>` in a terminal window (linux/UNIX) or double-click the R icon (Windows).

Below follows a small "crash course" about writing R code to manage for this laboration[1]

To terminate the session, write
`q()`
(but rather don't do it at once ;)

For on-line help, write
`help.start()`
and wait for a help page to pop up in Netscape. Once there, go to *Packages*, then further

[1]For a more thorough R manual, please talk to Ulla Johansson, room B 222. She will sell you an *The R manual* copy (105 pages) for 55:- SEK.

to *base*. There is a list of all kinds of commands and mostly solid explanations will appear if the commands are clicked.

In order to form a vector, x, consisting of the elements 2,5,3 one may write
`x <- c(2,5,3)`

For the vector consisting of $1, 2, \ldots, 100$, write
`x <- 1:100`

To pick out the fifth element in the vector, write
`x[5]`

Multiplying two vectors works element-wise:
```
x <- 2*1:5
y <- 8:12
x*y
```
gives
`[16 36 60 88 105]`
since this is $[\, 2 \cdot 8 \quad 4 \cdot 9 \quad 6 \cdot 10 \quad 8 \cdot 11 \quad 10 \cdot 12 \,]$.

To sum the elements of a vector, write
`sum(x*y)`
which renders
`320`
since this is $16 + 36 + 60 + 88 + 120$.

To form a 2×5-matrix of zeros, write
`z <- matrix(0,2,5)`

To assign the value 7 to the element in row 2, column 3, write
`z[2,3] <- 7`

To assign the values $5, 4, 3, 2, 1$ to the second row, write `z[2, ] <- 5:1`

Simulation of 3 observations of a variable distributed $R(0, \pi/2)$ variable, write
`runif(3,0,pi/2)`

Simulation of 100 observations of a $N(2, 7)$ variable, write
`rnorm(100,2,7)`

To pull the square root of 2:
`sqrt(2)`

Two examples of logical statements are
`x[3] == 4` and `(5 <= sum(x)) & (10 != sin(prod(x)))`
which mean $x_3 = 4$ (which is *false*=`F`) and $(5 \leq \sum_{i=1}^{5} x_i) \wedge (10 \neq \sin(\prod_{i=1}^{5} x_i))$ respectively.

The syntax of an *if*-statement is
`if(`< logisk utsaga >`){`
  ...
}

Example of a *for*-loop is

```
for(i in 1:3){
  z[1,i] <- 2*i+1
}
```

which assigns the values $3, 5, 7$ to the vector `z`.

To plot the vectors x and y against each other, write

```
plot(x,y,...)
```

where further optional arguments (which could be provided at `...`) are e.g.

`type=''l''`	(plots line(s) instead of points)
`ylim=c(0,5)`	(assigns the width of the y axis)
`xlab=''time''`	(assigns the text *time* next to the x-axis)

If, before the plot command, saying

```
par(mfrow=c(3,2))
```

means that the plot window is divided into 3 rows and 2 columns where each entry is a plot region.

To save a plot as a postscript-file, write

```
postscript(''npr.ps'')
```

at the line *before* `par` and `plot` and

```
dev.off()
```

at the line *after* `par` and `plot`

THAT'S ALL FOLKS!