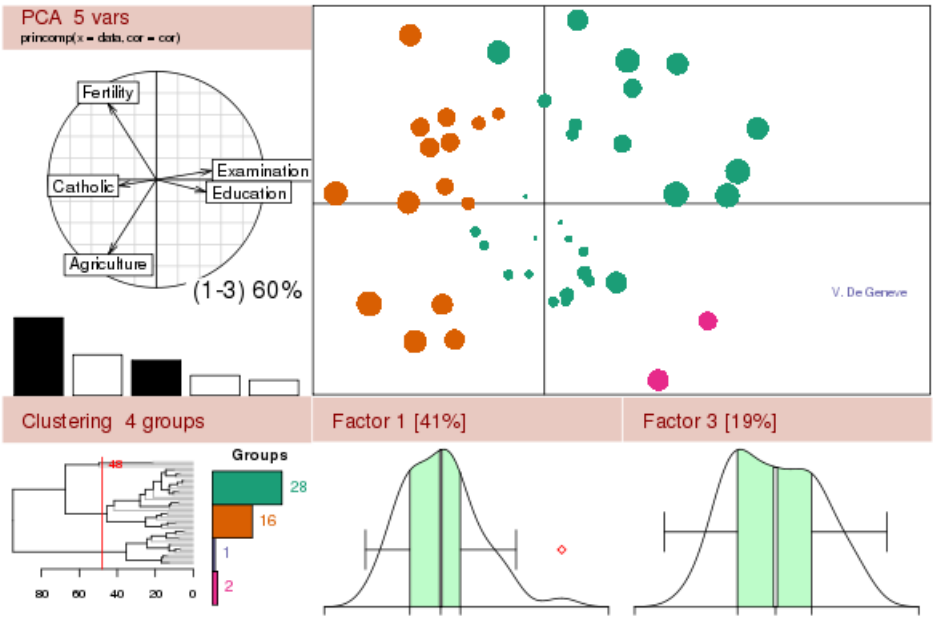
Imperial College London

diamond

A short introduction to the R statistical programming language



Diamond Light Source - April 2013 – PART 2

James Foadi

MPL, Imperial College London - Diamond Light Source Ltd, Oxfordshire

R as a numerical simulator

R is very good at dealing with arrays of numbers. The Most frequently used are "vectors" and "matrices":

```
> v <- 1:30
> v
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30
> sqrt(v)
 [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
 [9] 3.000000 3.162278 3.316625 3.464102 3.605551 3.741657 3.872983 4.000000
[17] 4.123106 4.242641 4.358899 4.472136 4.582576 4.690416 4.795832 4.898979
[25] 5.000000 5.099020 5.196152 5.291503 5.385165 5.477226
```

R as a numerical simulator

Several built-in functions are designed to deal with such arrays of numbers:

```
 > sum(v)
[1] 465
> mean(v)
[1] 15.5
> sd(v)
[1] 8.803408
> cumsum(v)
 [1]   1   3   6  10  15  21  28  36  45  55  66  78  91 105 120 136 153 171 190
[20] 210 231 253 276 300 325 351 378 406 435 465
```

R as a numerical simulator

And calculations are very fast:

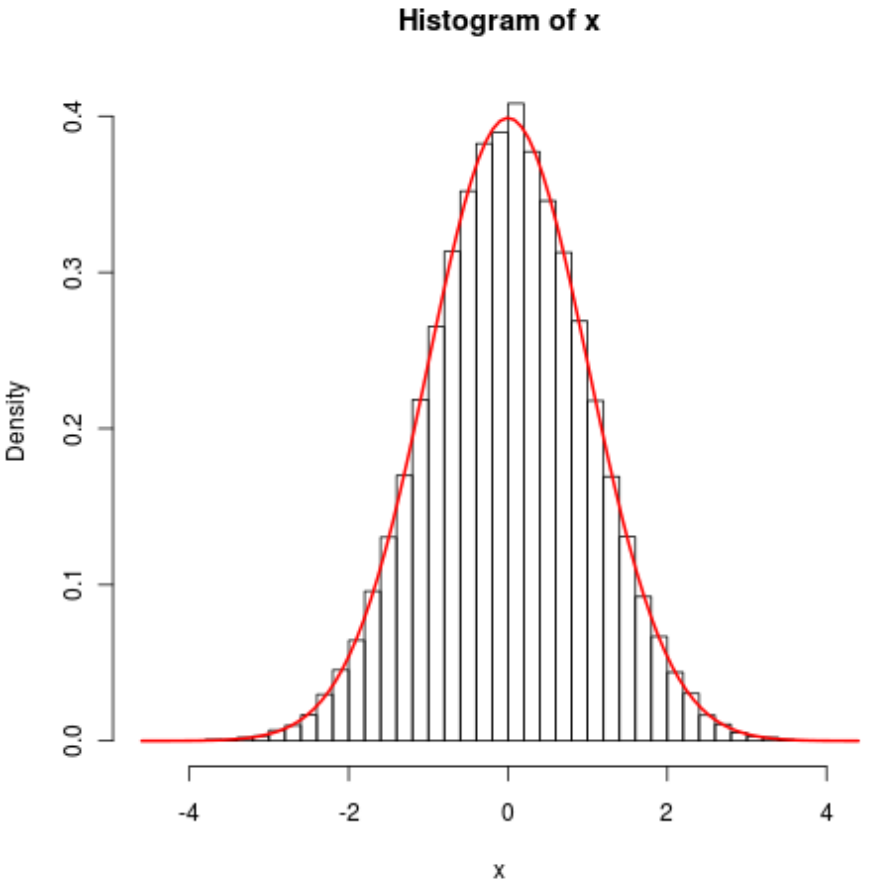
generates regular grids

```
 > x <- seq(0,100,by=0.01)
> length(x)
[1] 10001
> x[1:20]
 [1] 0.00 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10 0.11 0.12 0.13 0.14
[16] 0.15 0.16 0.17 0.18 0.19
> sum(x)
[1] 500050
> mean(x)
[1] 50
```

R as a numerical simulator

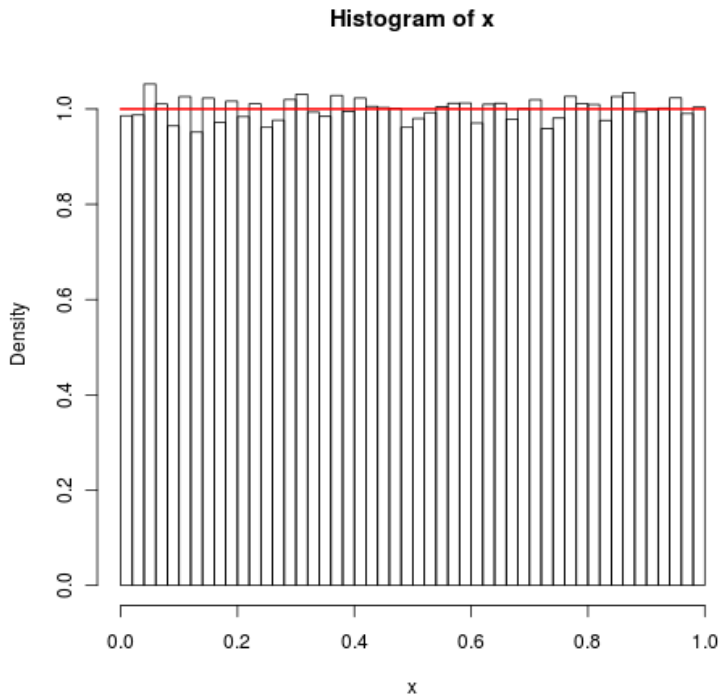
One of the frequently used features of R is the generation of random deviates:

```
> x <- rnorm(100000)
> mean(x)
[1] -0.002237135
> sd(x)
[1] 1.000233
> hist(x,breaks=50,freq=FALSE)
> curve(dnorm(x),from=-6,to=6,col=2,
lwd=2,add=TRUE)
> x <- rnorm(100000,mean=10,sd=1.5)
> mean(x)
[1] 9.996978
> sd(x)
[1] 1.498972
```



R as a numerical simulator

```
> x <- runif(100000,min=0,max=1)
> mean(x)
[1] 0.5005418
> sd(x)
[1] 0.2889073
> hist(x,breaks=50,freq=FALSE)
> curve(dunif(x,min=0,max=1),from=0,to=1,lwd=2,col=2,add=TRUE)
```



R as a numerical simulator

Exercise: simulate a game of darts with two players. Each player is arbitrarily assigned a quantitative measure of his/her ability to score in the following way; the position on the target in which a dart lands is a random vector whose x and y coordinates follow independent gaussian distributions centred somewhere close to the centre.
Design a match consisting of 3 throws and decide which player wins based on score.

Hints:
a) Load function "throwDart" from file "course_functions.R";
b) Load function "drawTarget" from the same file;
c) Load function "drawHit" from the same file

R as a numerical simulator

In order to use the function "draw.circle" included in our "drawTarget" function, we will need to install a package not included with the default R distribution. This is fairly straightforward using the function "install.packages". The simplest way to use it is:

```
> install.packages("plotrix")
```

This function prompts a window in which it is asked to enter the preferred repository (choose one close to you!). Once done it will install package "plotrix".

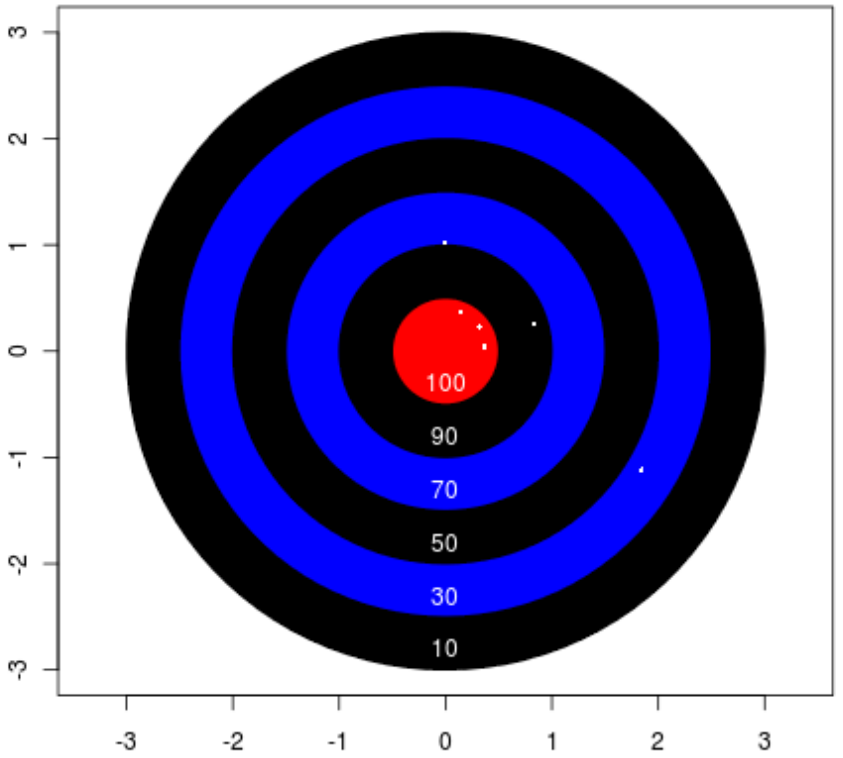
R as a numerical simulator

This is what I would do:

```
> source("course_functions.R")
> drawTarget()
> p <- throwDart(biasx=0.5,sdx=0.2,biasy=0.2,sdy=0.2)
> drawHit(p)
> p <- throwDart(biasx=0.5,sdx=0.2,biasy=0.2,sdy=0.2)
> drawHit(p)
> p <- throwDart(biasx=0.5,sdx=0.2,biasy=0.2,sdy=0.2)
> drawHit(p)
> p <- throwDart(biasx=0.5,sdx=0.5,biasy=0.2,sdy=0.5)
> drawHit(p)
> p <- throwDart(biasx=0.5,sdx=0.5,biasy=0.2,sdy=0.5)
> drawHit(p)
> p <- throwDart(biasx=0.5,sdx=0.5,biasy=0.2,sdy=0.5)
> drawHit(p)
```

R as a numerical simulator

And this is what is the resulting drawing:



Player A totals 300; player B totals 210. Player A wins!

R as a numerical simulator

Matrices. These are essentially long vectors with rows and columns specified:

```
> M <- matrix(c(1,2,3,4,5,6),nrow=2)
> M
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> M <- matrix(c(1,2,3,4,5,6),nrow=2,byrow=TRUE)
> M
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> U3 <- diag(3)
> U3
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

R as a numerical simulator

```
> dim(M)
[1] 2 3
> dim(U3)
[1] 3 3
> C <- M %*% U3
> dim(C)
[1] 2 3
> C
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> Mt <- t(M)
> dim(Mt)
[1] 3 2
> Mt
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> Mt %*% M
     [,1] [,2] [,3]
[1,]   17   22   27
[2,]   22   29   36
[3,]   27   36   45
```

```
> A <- matrix(rnorm(9),nrow=3)
> A
           [,1]       [,2]       [,3]
[1,] -0.9946193 -1.0367969  0.2113490
[2,]  0.8977808  2.3350694 -0.1445486
[3,]  0.6788108  0.2688433 -0.1681882
> A_1 <- solve(A)
> A_1
           [,1]       [,2]       [,3]
[1,]  -26.901705  -8.936866  -26.124524
[2,]    4.019629   1.810621    3.495024
[3,] -102.150566 -33.175149 -105.798131
> A %*% A_1
             [,1]         [,2]         [,3]
[1,] 1.000000e+00 -8.881784e-16 0.000000e+00
[2,] 1.776357e-15  1.000000e+00 1.776357e-15
[3,] 0.000000e+00  2.664535e-15 1.000000e+00
> A_1 %*% A
             [,1]         [,2]         [,3]
[1,]  1.000000e+00  7.993606e-15 -1.776357e-15
[2,] -1.332268e-15  1.000000e+00  2.220446e-16
[3,]  1.421085e-14 -3.552714e-15  1.000000e+00
```

R as a numerical simulator

On matrices you can use this very effective and useful command:

```
> A <- matrix(rnorm(200),ncol=10)
> dim(A)
[1] 20 10
> A[1,]
 [1]  0.56435304  0.77057778 -1.20553399 -2.04911264  1.32794769 -2.03359538
 [7] -1.27702584 -1.06364178  0.05903153  0.95063687
> A[,1]
 [1]  0.56435304 -0.34025342  0.09324284 -0.21328636 -0.45238874 -1.50287452
 [7]  1.61660385  0.73380978  0.36888828 -0.87451911  0.35519375 -0.40495468
[13] -0.19933757  0.92296672  0.19263452 -1.03261206 -0.86420501 -0.09639143
[19]  0.27197970 -1.23108061
> A[3:6,8:10]
           [,1]       [,2]       [,3]
[1,]  0.6964565 -0.2263148  0.91498137
[2,]  1.1983945 -0.9041424 -0.03920181
[3,] -0.5718469 -1.1094583  2.19262563
[4,] -0.5241946 -0.4708177 -0.21935156
```

R as a numerical simulator

```
> mrow <- apply(A,1,mean)
> mrow
 [1] -0.39563627  0.31207537  0.13502556  0.53812817  0.20464784 -0.14046040
 [7]  0.30502986 -0.10216710 -0.12538671  0.25302871 -0.23206275  0.08461194
[13] -0.04175629  0.26930286 -0.34589690 -0.24401831  0.11890782  0.42650411
[19]  0.16047605 -0.17447522
> mcol <- apply(A,2,mean)
> mcol
 [1] -0.10461155 -0.05536781 -0.24355399 -0.08737137  0.01333444  0.31463715
 [7]  0.16423313  0.17924824  0.30761315  0.01477779
```

"apply" can be used with any function, even with a user-defined one.

Try it yourself!

Exercise 1

Write a function “throwDart2” similar to “throwDart”, whose underlying distributions are uniform. Add to function “drawHit” the possibility to colour the dot with a colour different from “white”. This is useful to discriminate between hits for multiple players.

Exercise 2

Generates a matrix of gaussian random deviates of mean 0 and standard deviation 1, with 20 rows and 25 columns. Determine the column whose elements add up to the lowest number.