Practicals in the R language:
Basic matrix operations

Pierre Legendre
Département de sciences biologiques
Université de Montréal

May 2006, May 2007

1. Creating vectors and matrices

There are several ways of creating vectors in the R language.

1. Use function 'scan' (read data in the R console to form a vector)

```
vec1 = scan()
1: 1
2: 5
3: 35
4:
Read 3 items

vec1
[1]  1  5 35

vec2 = scan()
1: 1
2: 2
3: 4
4:
Read 3 items
```

2. Use function 'c' (combine)

```
vec1 = c(1, 5, 35)
vec1
[1]  1  5 35

 # 'c' also allows one to combine pre-existing vectors

vec12 = c(vec1, vec2)
vec12
[1]  1  5 35  1  2  4
```

3. Read 6 numbers into a vector, then turn the vector into a matrix

```
vec3 = c(6, 3, 0, 7, -5, 1)
mat3 = matrix(vec3, 3, 2)
# Or on a single line:   mat3 = matrix(c(6, 3, 0, 7, -5, 1), 3, 2)
mat3
     [,1] [,2]
[1,]    6    7
[2,]    3   -5
[3,]    0    1

mat1 = matrix(vec3, 3, 2, byrow=TRUE)
mat1
     [,1] [,2]
[1,]    6    3
[2,]    0    7
[3,]   -5    1

# Why are these matrices different?
# What is the default convention followed by R when it creates a matrix from a vector?

# One can also write the data to a file, save it on the disk, then ask R to read the text file using
'read.table'. How to read a data file is described in the document "Introduction_to_R".
```

4. Generate vectors containing random numbers

Use 'rnorm' to obtain numbers drawn at random from a normal distribution.

```
randNorm1 = rnorm(10)              # or   randNorm1=rnorm(10,mean=0,sd=1)
randNorm1
 [1]  0.5537578  0.3176950  1.0835924  0.1095523  0.7744556 -1.0011711  1.0175194
 [8]  0.8675578  1.5805411  0.5824715

randNorm2 = rnorm(10, mean=50, sd=10)
randNorm2
 [1] 49.51681 42.66049 54.33008 47.16372 70.80451 48.25744 55.69606 49.06176 63.45640
[10] 47.96367

# Use 'runif' to obtain numbers drawn at random from a uniform distribution.

randUnif1 = runif(5)               # or   randUnif1=runif(5,min=0,max=1)
randUnif1
[1] 0.7684457 0.8364512 0.3568679 0.9119424 0.9935300

randUnif2 = runif(5, min=10, max=20)
randUnif2
[1] 15.75800 13.37846 17.76781 13.36102 17.49161
```

Use 'rlnorm' for the lognormal distribution (the log of a lognormal distribution is a normal distribution). One can provide the mean 'meanlog' and the standard deviation 'sdlog' of the corresponding normal distribution.

```
randLNorm1 = rlnorm(5)              # or  randLNorm1 = rlnorm(5, meanlog=0, sdlog=1)
randLNorm1
[1] 0.2129886 1.6512231 3.2237191 5.8959147 1.2279456
randLNorm2 = rlnorm(5, meanlog=2, sdlog=5)
randLNorm2
[1]  0.75931953  0.02862776  1.12951044  0.51971353 48.50212899
```

5. Generate regular sequences of numbers

```
res = seq(1, 5, 0.5)
res
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

6. Repeat sequences of numbers

```
res = rep(c(1, 2, 3), 4)
res
 [1] 1 2 3 1 2 3 1 2 3 1 2 3
```

7. Functions 'fix(name)' and 'edit(name)' allow one to modify one or several values in a matrix. The object can belong to the types 'data.frame', 'matrix' or 'vector'.

'fix' modifies values of the matrix directly:

```
fix(mat1)
```

'edit' will save the changes only if a new object name has been provided. Example:

```
mat1a = edit(mat1)
```

2. Basic matrix operations

1. Add two vectors or matrices: operator '+'

```
vec1
[1]  1  5 35
vec2
[1] 1 2 4
vec1+vec2
[1]  2  7 39

mat1
     [,1] [,2]
[1,]    6    3
[2,]    0    7
[3,]   -5    1
mat2              # Create mat2 using function matrix(), page 2
     [,1] [,2]
[1,]    3    8
[2,]    2   -1
[3,]    6   -4
mat1+mat2
     [,1] [,2]
[1,]    9   11
[2,]    2    6
[3,]    1   -3
```

Make sure that the two matrices have the same numbers of rows and columns before computing their sum.
What would be the result of the command mat1+t(mat2) ?

2. Scalar product of two vectors or matrices: operator '%*%'

```
vec1
[1]  1  5 35
vec2
[1] 1 2 4
vec1 %*% vec2
     [,1]
[1,]  151
```

When multiplying two vectors, R considers that the first one is a row matrix while the second is a column matrix.

```
# The orientation of vectors can be specified by transforming them into matrices.
Mvec1=matrix(vec1, 1, 3)
Mvec2=matrix(vec2, 3, 1)
Mvec1
     [,1] [,2] [,3]
[1,]    1    5   35
Mvec2
     [,1]
[1,]    1
[2,]    2
[3,]    4
Mvec1 %*% Mvec2
     [,1]
[1,]  151
# What would be the result of   Mvec2 %*% Mvec1 ?

# Make sure that the two matrices are conformable before computing their product, that is, that
the number of columns of the matrix on the left is the same as the number of rows of the matrix
on the right.

# Function 'dim' prints out the dimension (size, format) of a matrix:
dim(Mvec1)
[1] 1 3
dim(Mvec2)
[1] 3 1

mat1 %*% mat2
# Which message did you obtain? What can you do about this situation?

mat1 %*% t(mat2)
     [,1] [,2] [,3]
[1,]   42    9   24
[2,]   56   -7  -28
[3,]   -7  -11  -34
# Operator 't' has transposed matrix 'mat2'.
```

3. Hadamard product: element-by-element product of two vectors or matrices

```
vec1 = c(1, 5, 35)
vec2 = c(1, 2, 4)
vec1 * vec2

mat1 = matrix(c(6,0,-5,3,7,1), 3, 2)
mat2 = matrix(c(3,2,6,8,-1,-4), 3, 2)
mat1 * mat2
```

Other useful matrix commands
See the help files for details. Example: ?as.matrix

Convert a 'data.frame' or 'vector' object to the 'matrix' type: function 'as.matrix'
Centre or standardize the values in the columns of a matrix: function 'scale'
Compute a covariance matrix: function 'cov'
Compute a correlation matrix: function 'cor'
Transpose a matrix: operator 't'
Compute the determinant: function 'det'
Matrix inversion: function 'solve', or 'ginv' (*generalized inverse*, library MASS)
Compute eigenvalues and eigenvectors: function 'eigen'
Singular value decomposition: function 'svd'

4. Compute a determinant

```
mat3x3 = matrix(c(1,2,3,4,5,6,7,8,10), 3, 3, byrow=TRUE)
mat3x3
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8   10
det(mat3x3)
[1] –3
```

5. Compute eigenvalues and eigenvectors

```
mat2x2
     [,1] [,2]
[1,]    2    2
[2,]    2    5
res = eigen(mat2x2)

# Function 'summary' tells us that object 'res' contains two components: 'res$values' and
'res$vectors':
summary(res)
        Length Class  Mode
values  2      -none- numeric
vectors 4      -none- numeric

res$values
[1] 6 1

res$vectors
          [,1]       [,2]
[1,] 0.4472136  0.8944272
[2,] 0.8944272 -0.4472136
```

6. Compute power 2 of a matrix using matrix product (scalar or dot product)

```
B = matrix(c(2,3,3,5), 2, 2)
B
[,1] [,2]
[1,]    2    3
[2,]    3    5
B %*% B
     [,1] [,2]
[1,]   13   21
[2,]   21   34
```

Note: the exponent operator '^' produces a matrix in which each element is put to the stated power. **It does not power the matrix.**

```
B^2
     [,1] [,2]
[1,]    4    9
[2,]    9   25
```

7. Power a matrix by eigen-decomposition (*Numerical ecology* 1998, p. 91, equation 2.29)

```
eig.B = eigen(B)

eig.B$values
[1] 6.8541020 0.1458980
eig.B$vectors
          [,1]       [,2]
[1,] 0.5257311  0.8506508
[2,] 0.8506508 -0.5257311
```

B^2 = U %*% (squared diagonal matrix of eigenvalues) %*% inverse of U

```
B.exp2 = eig.B$vectors %*% diag(eig.B$values^2) %*% solve(eig.B$vectors)
B.exp2
     [,1] [,2]
[1,]   13   21
[2,]   21   34
```

Put matrix B to the power 3.1416

```
BB = eig.B$vectors %*% diag(eig.B$values^3.1416) %*% solve(eig.B$vectors)
BB
         [,1]     [,2]
[1,] 116.8843 189.1189
[2,] 189.1189 306.0031
```

Note: This calculation cannot be done if there are negative eigenvalues **and** the exponent is not an integer. The reason is that a fractional exponent of a negative number is undefined.

8. Apply a function to the rows or columns of a data table: 'apply'

```
?apply

# Create a matrix of random numbers with normal distribution
mat = matrix(rnorm(15, 5, 1),5,3)
# Find the row sums of 'mat'
# Parameter "1" applies function "sum" to the rows of "mat"
row.sums = apply(mat, 1, sum)
row.sums

# Find the column sums of 'mat'
# Parameter "2" applies function "sum" to the columns of "mat"
col.sums = apply(mat, 2, sum)
col.sums
# What would the following command produce?   sum(mat)

# Find the column means of 'mat'
col.means = apply(mat, 2, mean)
col.means
# What would the following command produce?   mean(mat)

# Find the column standard deviations of 'mat'
col.sd = apply(mat, 2, sd)
col.sd
# What would the following command produce?   sd(mat)

# Find the column variances of 'mat'
col.var = apply(mat, 2, var)
col.var
# What would the following command produce?   var(mat)

# Center 'mat' by columns
mat.centred = scale(mat, center=TRUE, scale=FALSE)                    # or else
mat.centred = apply(mat, 2, scale, center=TRUE, scale=FALSE)
# Check that the column sums are now zero
apply(mat.centred, 2, sum)
# How do you read these results?

# Standardize 'mat' by columns,
# i.e., subtract the means and divide by the standard deviations
mat.stan = scale(mat, center=TRUE, scale=TRUE)                        # or else
mat.stan = apply(mat, 2, scale, center=TRUE, scale=TRUE)
# Check the column sums and variances
apply(mat.stan, 2, sum)
apply(mat.stan, 2, var)
```

9. A useful function: 'which'

```
# Find the positions of values with certain properties in a vector or matrix
?which

# What is the location of the positive values in a vector of random numbers?
vec = rnorm(10)
vec.positive = which(vec > 0)
vec
vec.positive

# What is the location of the negative values in a matrix of random numbers?
mat = matrix(rnorm(15), 5, 3)
mat. negative = which(mat < 0)
mat
mat.negative
# In what order are the data read in a matrix?
```

10. A useful function: 'sweep'

```
# Produces a matrix obtained by an operation, involving a statistic, applied to each cell of an
input matrix
?sweep

# Create a table of species-like abundances, with 10 rows and 5 columns
# The proportion of zeros in 'table' depends on the standard deviation parameter (1.5) in 'rnorm'[1]
table = matrix(round(exp(rnorm(50, 0, 1.5))), 10, 5)

# Divide each value by the row sum, creating a table of relative abundances
row.sums = apply(table, 1, sum)
table.profiles = sweep(table, 1, row.sums, "/")
table.profiles
# Check that the row sums of the new table are 1
apply(table.profiles, 1, sum)

# Divide each value by the maximum value found in the table
table.max = max(table)
table.norm = sweep(table, 2, table.max, "/")
table.norm
# Check the maximum value in the transformed table
max(table.norm)
```

[1] Formula from the program SimSSD. The formula is described on p. 443 of the following paper:

Legendre, P., D. Borcard and P. R. Peres-Neto. 2005. Analyzing beta diversity: partitioning the spatial variation of community composition data. *Ecological Monographs* 75: 435-450.