R Programming for Demographers

Matrix Algebra

Tim Riffe[*]

September 19, 2017

Contents

1 Introduction

Perhaps the most common data structure for many users in `R` is a `data.frame`. However, from a mathematical and computational point of view the `vector` and `matrix` are the most important data structures (as well as an `array` if one considers more than two dimensions). Recall the basic difference: A `matrix` is a rectangular data structure where all elements are of the same type: either numbers or logical values or character strings, etc. Of course, matrices of numeric values are the most common.

On the other hand, a `data.frame`, which is also rectangular, may have columns of different types. This allows the user to combine numeric variables along with character variables, factors, or others. Sometimes `R` seems to behave strangely, if one tries to apply methods on a data frame, which actually is supposed to be a matrix, or the other way around. So be intentional about what data structure you're using for a given job.

[*]First, a big thanks to Giancarlo Camarda, who has let me recycle his impeccable material for teaching this course. Higher order thanks are also due to the very same people that Giancarlo himself thanked, especially Sabine Zinn and Roland Rau. Sabine was my teacher in this very course in 2009 (cohort 5)! Part of these lectures has been freely inspired by courses that both Roland, Sabine, and Giancarlo taught in the past. Furthermore, in recent years, Adam Lenart, Fernando Colchero, and Silvia Rizzi have aided in teaching this course and improving handouts. On the other hand, it goes without saying that I am uniquely responsible for any deficiencies and mistakes you may find in handouts.

The most common pitfall is that objects created by the `read.table()` command are data frames even though one thinks the input as a two-dimensional array of numbers that are supposed to describe a matrix. However there is an easy way to coerce a `data.frame` into a `matrix` by applying `data.matrix()`.

Apart from reading in matrices from external files, matrices can be created by several commands such as `matrix()`. This, together with indexing and subsetting, was presented in Session 2.

2 Matrix operations

2.1 Addition and Subtraction

If you want to add or subtract matrices, both matrices have to have the same dimensions. Each element in one matrix is then added to (or subtracted from) the corresponding element in the other matrix:

```
A <- matrix(1:4, nrow = 2, ncol = 2)
B <- matrix(5:8, nrow = 2, ncol = 2)
A

##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4

B

##      [,1] [,2]
## [1,]    5    7
## [2,]    6    8

A + B

##      [,1] [,2]
## [1,]    6   10
## [2,]    8   12

A - B

##      [,1] [,2]
## [1,]   -4   -4
## [2,]   -4   -4
```

2.2 Scalar Multiplication

A matrix is multiplied with a scalar by multiplying each element with the scalar:

```
A * 5

##      [,1] [,2]
## [1,]    5   15
## [2,]   10   20

5 * A

##      [,1] [,2]
## [1,]    5   15
## [2,]   10   20
```

Scalar multiplication can be done between two vectors/matrices. In this case multiplication is done element-by-element (the hadamard product):

```
A * B

##      [,1] [,2]
## [1,]    5   21
## [2,]   12   32

B * A

##      [,1] [,2]
## [1,]    5   21
## [2,]   12   32
```

2.3 Vector Multiplication

To multiply a row-vector with a column-vector (or vice-versa) both vectors have to have the same number of elements. Then the corresponding elements are multiplied with each other and the resulting products will be summed up.

```
A.vec <- matrix(c(2, 6, 5, 8), nrow = 1)
B.vec <- matrix(c(7, 3, 9, 4), ncol = 1)
A.vec

##      [,1] [,2] [,3] [,4]
## [1,]    2    6    5    8

B.vec

##      [,1]
## [1,]    7
## [2,]    3
## [3,]    9
## [4,]    4

intermediate <- c(A.vec[1] * B.vec[1],
                  A.vec[2] * B.vec[2],
                  A.vec[3] * B.vec[3],
                  A.vec[4] * B.vec[4])
intermediate

## [1] 14 18 45 32

sum(intermediate)

## [1] 109
```

This should give the same result as (please note the different operator!):

```
A.vec %*% B.vec

##      [,1]
## [1,]  109
```

2.4 Matrix Multiplication

Two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ can be multiplied in this way $\boldsymbol{A}\% * \%\boldsymbol{B}$, only if the number of columns in matrix $\boldsymbol{A}$ is equal to the number of rows in matrix $\boldsymbol{B}$. These matrices are then called "conformable". Each row vector in matrix $\boldsymbol{A}$ is then multiplied by the corresponding column vector in matrix $\boldsymbol{B}$. An example in `R` :

```
A <- matrix(c(4, 9, 11, 7, 12, 3), nrow = 2, byrow = TRUE)
B <- matrix(c(8, 2, 6, 20, 7, 12), nrow = 3, byrow = TRUE)
A

##      [,1] [,2] [,3]
## [1,]    4    9   11
## [2,]    7   12    3

B

##      [,1] [,2]
## [1,]    8    2
## [2,]    6   20
## [3,]    7   12
```

Multiplying by hand it would look like this:

```
upper.left  <- sum(c(A[1, 1] * B[1, 1], A[1, 2] * B[2, 1], A[1, 3] * B[3, 1]))
lower.left  <- sum(c(A[2, 1] * B[1, 1], A[2, 2] * B[2, 1], A[2, 3] * B[3, 1]))
upper.right <- sum(c(A[1, 1] * B[1, 2], A[1, 2] * B[2, 2], A[1, 3] * B[3, 2]))
lower.right <- sum(c(A[2, 1] * B[1, 2], A[2, 2] * B[2, 2], A[2, 3] * B[3, 2]))
matrix(c(upper.left,  upper.right,
         lower.left, lower.right),
         nrow = 2, byrow = TRUE)

##      [,1] [,2]
## [1,]  163  320
## [2,]  149  290
```

But it is much more convenient in `R` to use the built in matrix multiplication:

```
A %*% B

##      [,1] [,2]
## [1,]  163  320
## [2,]  149  290
```

Please note that the operator *matrix multiplication* is noncommutative since

$$\begin{pmatrix}0 & 2\\0 & 1\end{pmatrix} = \begin{pmatrix}1 & 1\\0 & 1\end{pmatrix} \cdot \begin{pmatrix}0 & 1\\0 & 1\end{pmatrix} \neq \begin{pmatrix}0 & 1\\0 & 1\end{pmatrix} \cdot \begin{pmatrix}1 & 1\\0 & 1\end{pmatrix} = \begin{pmatrix}0 & 1\\0 & 1\end{pmatrix}$$

We can check with our example:

```
A %*% B

##      [,1] [,2]
## [1,]  163  320
## [2,]  149  290

B %*% A

##      [,1] [,2] [,3]
## [1,]   46   96   94
## [2,]  164  294  126
## [3,]  112  207  113
```

Important Special Case

The following case should receive attention especially from demographers working with matrix population models. One can multiply a square matrix with a column vector (given the number of rows in the matrix is equal to the number of elements in the vector):

$$\boldsymbol{Ax} = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} A_{11}x_1 & + & A_{12}x_2 & + & A_{13}x_3 \\ A_{21}x_1 & + & A_{22}x_2 & + & A_{23}x_3 \\ A_{31}x_1 & + & A_{32}x_2 & + & A_{33}x_3 \end{pmatrix}$$

Let's do this now "by hand" in R :

```
A <- matrix(1:9, nrow = 3, byrow = FALSE)
x <- matrix(1:3, ncol = 1)
A

##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9

x

##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3

el1 <- A[1, 1] * x[1] + A[1, 2] * x[2] + A[1, 3] * x[3]
el2 <- A[2, 1] * x[1] + A[2, 2] * x[2] + A[2, 3] * x[3]
el3 <- A[3, 1] * x[1] + A[3, 2] * x[2] + A[3, 3] * x[3]
rbind(el1, el2, el3)

##     [,1]
## el1   30
## el2   36
## el3   42
```

And in the easy way in R :

```
A %*% x

##      [,1]
## [1,]   30
## [2,]   36
## [3,]   42
```

2.5 Transpose and Inverse

If you transpose a matrix you simply turn columns into rows and rows into columns. The turning point is the main diagonal. Usually you denote the transpose of a matrix or a vector like this: $\boldsymbol{A}^T$ or $\boldsymbol{A}'$. In R you do:

```
A <- matrix(rpois(25,10), ncol=5)
A

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    8    7   11   12    9
## [2,]   13    9    7    8   13
## [3,]    9    6    8    8    8
## [4,]   11    4   10   10   12
## [5,]   15    8    7    9    6
```

```
t(A)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    8   13    9   11   15
## [2,]    7    9    6    4    8
## [3,]   11    7    8   10    7
## [4,]   12    8    8   10    9
## [5,]    9   13    8   12    6
```

Another often-performed operation on matrices is the inverse of a matrix, typically denoted as $\boldsymbol{A}^{-1}$. For quadratic matrices (of full rank) the inverse of a matrix is such that: $\boldsymbol{A} \cdot \boldsymbol{A}^{-1} = \boldsymbol{I}$ where $\boldsymbol{I}$ denotes the Identity matrix. In `R` , this operation can be achieved by the function `solve()`, when given a single matrix argument:

```
solve(A)
```

```
##             [,1]        [,2]        [,3]        [,4]        [,5]
## [1,] -0.13853904 -0.05289673  0.06801008  0.06801008  0.09571788
## [2,]  0.02392947  0.08186398  0.25188917 -0.24811083 -0.05289673
## [3,] -0.35957179 -0.28274559  1.14924433 -0.10075567 -0.17884131
## [4,]  0.47292191  0.14420655 -1.11397985  0.13602015  0.19143577
## [5,]  0.02455919  0.13664987 -0.17569270  0.07430730 -0.08060453
```

To show that $\boldsymbol{A} \cdot \boldsymbol{A}^{-1} = \boldsymbol{I}$, we multiply the original matrix with its inverse.:

```
A %*% solve(A)
```

```
##               [,1]          [,2]          [,3]          [,4]          [,5]
## [1,]  1.000000e+00 -4.440892e-16 -1.110223e-15 -5.551115e-16 -1.110223e-16
## [2,]  9.992007e-16  1.000000e+00 -2.664535e-15 -2.220446e-16  0.000000e+00
## [3,] -1.665335e-16  0.000000e+00  1.000000e+00 -1.110223e-16 -2.220446e-16
## [4,]  0.000000e+00  0.000000e+00 -1.332268e-15  1.000000e+00  1.110223e-16
## [5,]  0.000000e+00 -2.220446e-16 -6.661338e-16  1.110223e-16  1.000000e+00
```

Despite some rounding error, we get back an identity matrix.
The same function `solve()`, when given two arguments, solves linear systems of the form:

$$\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}$$

for $\boldsymbol{x}$[1].
For example, assume that we aim to solve the following system of equations:

$$\begin{aligned} 10x_1 + 30x_2 &= 5 \\ 20x_1 + 40x_2 &= 1 \end{aligned}$$

That is

$$\boldsymbol{A} = \begin{bmatrix} 10 & 30 \\ 20 & 40 \end{bmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$$

Let's translate these objects in `R` :

```
A <- matrix(c(10, 20, 30, 40), 2, 2)
b <- matrix(c(5, 1), 2, 1)
```

[1]If the system is over-determined it gives the LS-fit, if $\boldsymbol{A}$ does not have full rank it gives an error.

In other words, what is the combination of $\boldsymbol{x} = (x_1, x_2)^T$ that satisfies both equations? In R just execute:

```
x <- solve(A, b)
x

##       [,1]
## [1,] -0.85
## [2,]  0.45
```

To test the result we reverse the operation multiplying $\boldsymbol{A}$ by $\boldsymbol{x}$:

```
A%*%x

##      [,1]
## [1,]    5
## [2,]    1
```

and obtaining $\boldsymbol{b}$ back.

3 Decomposing a matrix

Besides matrix transposition and inversion there are several matrix decompositions readily available in R . Many of them, like the Choleski decomposition (function `chol()`) for symmetric positive-definite matrices or the QR decomposition (function `qr()`) are important for numerical purposes.

3.1 Eigenvalues and eigenvectors

The calculation of eigenvalues and eigenvectors of a square matrix is a valuable tool in matrix population models, too. The function `eigen()` calculates both eigenvalues and -vectors.

```
eA <- eigen(A)
eA

## eigen() decomposition
## $values
## [1] 53.722813 -3.722813
## 
## $vectors
##            [,1]       [,2]
## [1,] -0.5657675 -0.9093767
## [2,] -0.8245648  0.4159736

A %*% eA$vec[, 1]

##           [,1]
## [1,] -30.39462
## [2,] -44.29794

eA$val[1] * eA$vec[, 1]

## [1] -30.39462 -44.29794
```

Let’s check the following two relationsships:

- the product of the eigenvalues is equal to the determinant of $\boldsymbol{A}$

```
det(A)
```

```
## [1] -200

prod(eA$val)

## [1] -200
```

- the sum of the eigenvalues is equal to the trace of $\boldsymbol{A}$, where we denote with trace the sum of the diagonal-elements

```
sum(diag(A))

## [1] 50

sum(eA$val)

## [1] 50
```

Computing the intrinsic growth rate

We saw in handout 4 how to compute the intrinsic growth rate by solving the discrete version of the Lotka equation using Newton's method.

In this section we will solve with the same problem using matrix algebra. Information about population growth is contained in the projection matrix. Specifically it turns out that the key to stability, as well as the key to understanding whether a population eventually grows or eventually decreases, is in the eigenvalues.

The answer is that a projection matrix, under certain conditions, has a single dominant eigenvalue λ_1, interpretable as the net reproductive rate (NRR, a.k.a. R_0). This eigenvalue is always a positive number and essentially determines the long-term behavior of the model as follows:

1. If the dominant eigenvalue satisfies $0 < \lambda_1 < 0$, then the population eventually decreases and decays to zero
2. If the dominant eigenvalue is exactly equal to 1, the population approaches a stationary state
3. If the dominant eigenvalue is larger than 1, the population eventually grows without bound

Let's use the information about USA in 1991 for computing the intrinsic growth rate. First we need the data:

```
usaC <- read.table("FunUSA1991compl.txt", header = TRUE)
head(usaC)

##    a       L      m
## 1  0 4.95804 0.0000
## 2  5 4.95002 0.0000
## 3 10 4.94603 0.0007
## 4 15 4.93804 0.0303
## 5 20 4.92552 0.0566
## 6 25 4.91138 0.0578

L <- usaC$L
m <- usaC$m
n <- length(m)
```

Here we need the complete life table person-years and the rate of bearing female children between two ages.

Then we construct the projection matrix. Two elements are fundamental here: (1) the survivorship ratio between two age-groups which should be placed in the subdiagonal; (2) the number of female births surviving for the first row of the projection matrix. We'll give an abbreviated version here, but you can get all the juicy details in Caswell (2001). The first elements are practically the ratio between subsequent values of L

```
SurvRatio <- L[-1] / L[-n]
```

The second part is a bit more involved: We need to average the rate of bearing female births coming from a given age-group and the subsequent one devoid of the probability of surviving. Then we need to multiply these values by the first number of L which introduces the chance for a new born to survive to the next age-group. In R :

```
FB0 <- (m[1:(n - 1)] + m[2:n] * SurvRatio[1:(n - 1)]) / 2
FB  <- L[1] * FB0
```

Now the construction of the projection matrix is straightforward except for an additional step for the last open age interval:

```
A                   <- matrix(0, n, n)  # set up
A[1, 1:length(FB)] <- FB               # female births
diag(A[-1,])        <- SurvRatio        # subdiag survival probs
```

As mentioned, for the last age interval we need to intervene directly by combining the last two age-groups and project them

```
A[n, n - 1] = A[n, n] = L[n] / (L[n - 1] + L[n])
```

Let's have a final look at our projection matrix:

```
fix(A)
```

We can take any population structure and apply the matrix `A` to reach a stable population with a precise age-structure that solely depends upon the projection matrix. That's called *strong* ergodicity. Moreover we can easily compute the intrinsic growth rate (Lotka's r) by taking the logarithm of the real part of the dominant (first) eigenvalues. Of course this needs to be divided by 5 to obtain the outcome for time-unit equal to single calendar year:

```
lambda1 <- Re(eigen(A)$values[1])
r       <- log(lambda1) / 5
r

## [1] -0.0001664792
```

The result is really close to what we previously obtain and it tells us that, as n gets larger, US population will decrease about -0.0166% each year. It can also be shown that no matter what the initial population structure is, it eventually approaches a multiple of the dominant eigenvector. In R the stable equivalent age distribution can be computed by

```
C <- Re(eigen(A)$vector[, 1] / sum(eigen(A)$vector[, 1]))
```

Note that most of the theory behind this last exercise goes beyond the scope of this module and course. For a detailed description of the methodology see, among others, Caswell (2001), Keyfitz and Caswell (2005, chapter 3), and Preston et al. (2001, chapter 6-7).

3.2 Singular Value Decomposition

If $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ is a rectangular matrix, it has a singular value decomposition (SVD) of

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}',$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ have orthonormal columns: $\boldsymbol{U}^{-1} = \boldsymbol{U}'$ and $\boldsymbol{V}^{-1} = \boldsymbol{V}'$. Moreover, $\boldsymbol{D}$ is diagonal, with diagonal elements d_i, ordered in decreasing order. The number of non-zero elements gives the rank of $\boldsymbol{X}$. In R , the command for this decomposition is `svd`:

```
B

##      [,1] [,2]
## [1,]    8    2
## [2,]    6   20
## [3,]    7   12

(duv <- svd(B))

## $d
## [1] 25.406402  7.177376
##
## $u
##            [,1]       [,2]
## [1,] -0.1995367  0.9061599
## [2,] -0.8153843 -0.3646026
## [3,] -0.5434460  0.2143342
##
## $v
##            [,1]       [,2]
## [1,] -0.4051231  0.9142621
## [2,] -0.9142621 -0.4051231

duv$u %*% diag(duv$d) %*% duv$v

##           [,1]       [,2]
## [1,] -3.892449  -7.269721
## [2,] 10.785047 -17.879675
## [3,]  4.187076 -13.246448

t(duv$u) %*% duv$u

##               [,1]          [,2]
## [1,]  1.000000e+00 -1.249001e-16
## [2,] -1.249001e-16  1.000000e+00
```

The singular values d_i are the square roots of the eigenvalues $\sqrt{\lambda_i}$ of the matrix $\boldsymbol{X}^T\boldsymbol{X}$. Check this fact for our matrix `B`:

```
sqrt(eigen(t(B) %*% B)$val)

## [1] 25.406402  7.177376

duv$d

## [1] 25.406402  7.177376
```

Another important aspect of the SVD is that often the first singular value(s) contain enough information about $\boldsymbol{X}$ as you can see from the differences in the values of d_i:

```
M    <- matrix(1:25,5,5)
svdM <- svd(M)
svdM$d

## [1] 7.425405e+01 3.366820e+00 7.659773e-15 1.593070e-15 4.530915e-16
```

Now we can approximate `M` using only the first singular value:

```
M

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    6   11   16   21
## [2,]    2    7   12   17   22
## [3,]    3    8   13   18   23
## [4,]    4    9   14   19   24
## [5,]    5   10   15   20   25

(svdM$u[,1] * svdM$d[1]) %*% t(svdM$v[,1])

##          [,1]     [,2]     [,3]     [,4]     [,5]
## [1,] 2.728602 7.102808 11.47701 15.85122 20.22543
## [2,] 2.912826 7.582362 12.25190 16.92143 21.59097
## [3,] 3.097051 8.061916 13.02678 17.99165 22.95651
## [4,] 3.281276 8.541470 13.80167 19.06186 24.32205
## [5,] 3.465500 9.021025 14.57655 20.13207 25.68760
```

Not too shabby!

Fitting the Lee-Carter model

This feature was used by Lee and Carter (1992) for estimating their well-known model directly on the log-death rates over age and time:

$$\ln(m_{ij}) = \alpha_i + \beta_i \cdot \kappa_j$$

where α_i, β_i and γ_j are vectors of parameters over either ages or years[2].

It can be seen that, once α_i (the average over time of the log-death rates) is subtracted from the original matrix of log-rates, we need to find the best approximation for a rectangular matrix using two vectors over the two domains: We can apply SVD. Without going into details, we need to warn that the model is under-determined, hence additional constraints are needed to make the solution unique. Commonly we take:

$$\sum_j \kappa_j = 0 \qquad\qquad \sum_i \beta_i = 1$$

In other words we need to follow this procedure:

1. Given the matrix of log-death rates, $\ln m_{ij}$
2. We average it over time, and this will be the first vector of parameters: $\alpha_i = \overline{\ln m_i}$
3. We center log-death rates: $\ln m_{ij} - \alpha_i$
4. We apply the SVD:
$$\ln m_{ij} - \alpha_i \simeq U_{,1}\, d_{1,1}\, V'_{,1}$$
5. Set and normalize vectors of parameters:
$$\begin{aligned} \beta_i &= U_{,1} \,/\, \sum_j U_{j,1} \\ \kappa_j &= s_{1,1} \cdot V_{,1} \cdot \sum_j U_{j,1} \end{aligned}$$

Let's load some data and apply SVD for fitting a Lee-Carter model. We take the log-death rates of French women from 0 to 100 in years 1930-2010.

[2]Nowadays there are more suitable way for estimating this model. Among others we would suggest to look at the Maximum Likelihood Estimation presented in Brouhns et al. (2002).

```
par(mfrow = c(1,3))
plot(ages, alpha)
       ## Error in xy.coords(x, y, xlabel, ylabel, log):  object 'alpha' not found

plot(ages, beta)
plot(years, kappa)
par(mfrow = c(1,1))
```

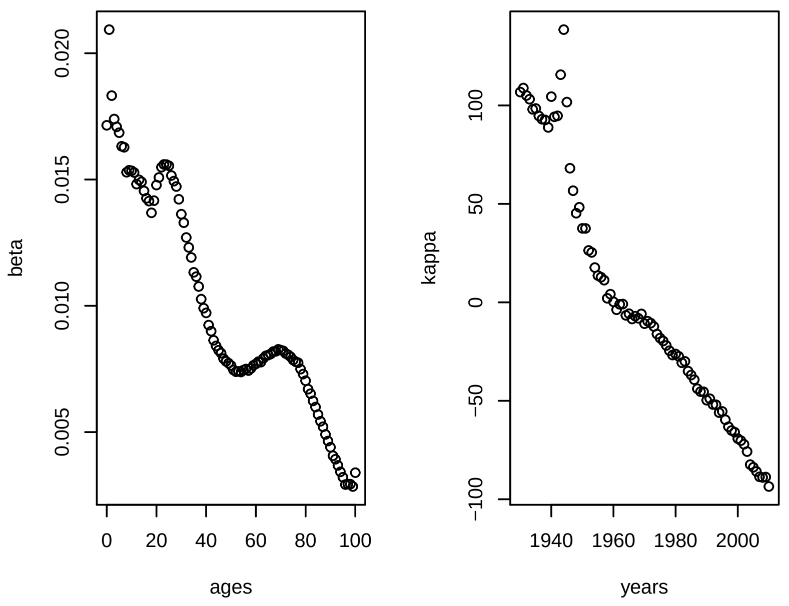

Figure 1: Parameter-vectors from the Lee-Carter model estimated with SVD. French women from 0 to 100 in years 1930-2010

```
lmx        <- read.table("lmxFRAfem.txt", header=T)
ages       <- 0:100
years      <- 1930:2010
cent.lmx   <- lmx - rowMeans(lmx)
SVD        <- svd(cent.lmx)
beta       <- SVD$u[, 1] / sum(SVD$u[, 1])
kappa      <- SVD$v[, 1] * SVD$d[1] * sum(SVD$u[, 1])
```

Now we can plot the parameters vectors (Fig. 1):

4 Regression with matrices

In the following we present another instance of the usefulness of working with matrices: Estimation of regression models. As you know from your basic statistics classes, we can write a linear model in matrix notation:

$$y = X\beta + \varepsilon\,.$$

Assuming normal errors, setting the derivatives of the log-likelihood with respect to β to zero, the estimation of coefficients is given by:

$$\hat{\beta} = \left(X'X\right)^{-1} X'y$$

where X denotes the stimulus matrix (also called design matrix or model matrix) and y denotes the response. A great book on this model and its generalization is McCullagh and Nelder (1989).

Note that if we include the intercept, the first column of $\boldsymbol{X}$ is actually a column of 1s.

4.1 Simulated data

Before playing with actual data, we always suggest simulating some data yourself. This helps to check whether new approaches are working (we know the truth because we made it) and which assumptions are behind our model (because we need to impose them in the simulation setting). Here is a simple linear model:

```
n        <- 100
x1       <- sort(runif(n))
X        <- model.matrix(~x1)
beta0    <- 3
beta1    <- 4
betas    <- c(beta0, beta1)
y.true   <- X %*% betas
y        <- y.true + rnorm(n, sd = 0.2)
```

The function `model.matrix()` is used here to build up the design matrix $\boldsymbol{X}$ and it include the intercept by default.

In this simple case one could also create the model matrix by

```
X <- cbind(1, x1)
```

Let's plot the data:

```
plot(x1, y)
lines(x1, y.true, col = 2, lwd = 3)
```

We expect our estimated parameters $\hat{\beta}$ to be equal to 3 and 4 for the intercept and the slope, respectively. Using the previous formula, and separating the two parts of our system:

```
LHS <- t(X) %*% X
RHS <- t(X) %*% y
betas.hat <- solve(LHS, RHS)
betas.hat

##         [,1]
##     2.898406
## x1 4.161377
```

We can then compute the fitted values and plot along with the simulated and true ones (Fig. 2):

```
y.hat <- X %*% betas.hat
lines(x1, y.hat, col = 4, lwd = 2, lty = 2)
```

Of course, you may already know, there are much easier ways to perform this in `R` :

```
fit.lm <- lm(y ~ x1)
fit.lm$coeff # same!

## (Intercept)          x1
##    2.898406    4.161377
```

4.2 Actual application

Now we'll fit a linear model on actual data. Specifically let's see whether there is any relation between the level of GDP per capita, in the log-scale, and life expectancy (a.k.a., the Preston-curve). We use the dataset in `GDPe0.txt` for carrying out this example:

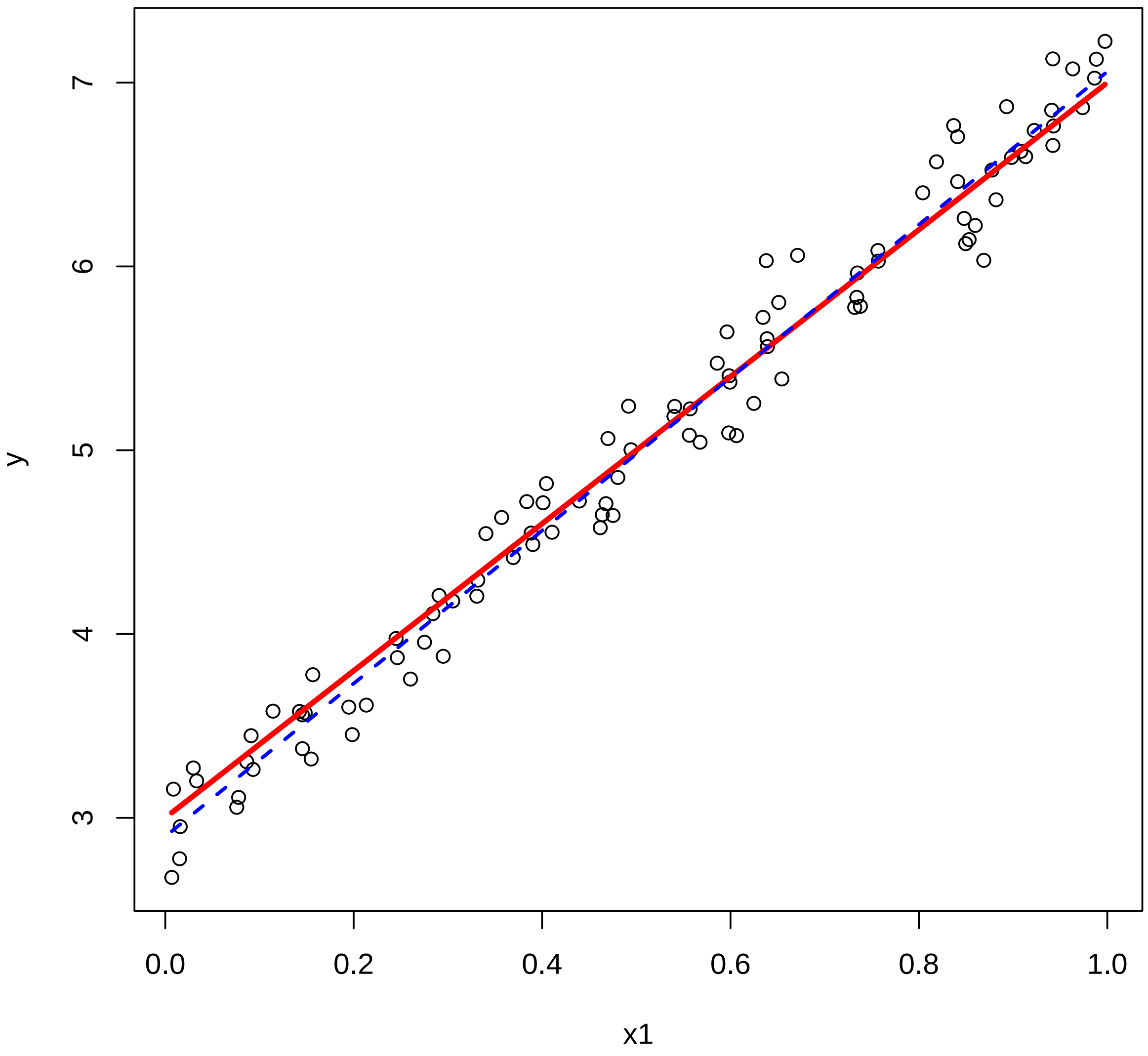

Figure 2: An example of linear model with simulated data.

```
GDPe0 <- read.table("GDPe02012.txt",
                    header = TRUE,
                    sep = ",",
                    stringsAsFactors = TRUE)
```

First we set up our response and model matrices, in which we have both log-GDP per capita and the geographical region. The idea is that log-GDP per capita acts linearly on life expectancy, and that we expect a different intercept for each region, i.e. parallel lines.

```
y  <- GDPe0$e0
x1 <- log(GDPe0$GDPcapPPP)
x2 <- GDPe0$Region
X  <- model.matrix(~x1 + x2)
```

Note that the variable `Region` is already read as a factor (categorical variable) in `R` and that, automatically, `model.matrix()` will construct dummy variables for this covariate by ordering it alphabetically and setting the first level as the reference category. In this case the reference will be the region `Africa` which actually will not appear among the column headers of `X`:

```
head(X)

##   (Intercept)       x1 x2Americas x2Eastern Mediterranean x2Europe
## 1           1 7.021976          0                       1        0
## 2           1 9.127719          0                       0        1
## 3           1 8.896314          0                       0        0
## 4           1 8.714732          0                       0        0
## 5           1 9.809012          1                       0        0
## 6           1 9.793505          1                       0        0
##   x2South-East Asia x2Western Pacific
## 1                 0                 0
## 2                 0                 0
## 3                 0                 0
## 4                 0                 0
## 5                 0                 0
## 6                 0                 0
```

We can now estimate the regression model and its coefficients:

```
LHS <- t(X) %*% X       # left-hand side
RHS <- t(X) %*% y       # right-hand side
b   <- solve(LHS, RHS)
b

##                              [,1]
## (Intercept)             28.784994
## x1                       3.981353
## x2Americas               8.601197
## x2Eastern Mediterranean  6.140195
## x2Europe                 8.923590
## x2South-East Asia        7.733042
## x2Western Pacific        7.820443
```

Of course also in this case we could have used the `R` routine `lm()`:

```
fitlm <- lm(y ~ x1 + x2)
summary(fitlm)

##
## Call:
```

```
## lm(formula = y ~ x1 + x2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.0846  -2.0501   0.3942   2.5506  11.9028
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)               28.7850     2.6664  10.796  < 2e-16 ***
## x1                         3.9814     0.3356  11.862  < 2e-16 ***
## x2Americas                 8.6012     1.0994   7.823 4.48e-13 ***
## x2Eastern Mediterranean    6.1402     1.2160   5.050 1.09e-06 ***
## x2Europe                   8.9236     1.1090   8.046 1.19e-13 ***
## x2South-East Asia          7.7330     1.4908   5.187 5.80e-07 ***
## x2Western Pacific          7.8204     1.1612   6.735 2.21e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.214 on 177 degrees of freedom
## Multiple R-squared:  0.7732,Adjusted R-squared:  0.7655
## F-statistic: 100.6 on 6 and 177 DF,  p-value: < 2.2e-16
```

obtaining the same estimated coefficients.

Note that p-values for the covariate `Region` are computed to test the difference of each intecept with the reference one (e.g. Africa) and the significance can not be interpreted as a real presence of independent intercepts for each geographical region. To check for this possible misunderstanding, we will `relevel` the covariate `x2` with Europe as reference and run the same model:

```
x2new <- relevel(x2, "Europe")
fitlm <- lm(y ~ x1 + x2new)
summary(fitlm)

##
## Call:
## lm(formula = y ~ x1 + x2new)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.0846  -2.0501   0.3942   2.5506  11.9028
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  37.7086     3.3456  11.271  < 2e-16 ***
## x1                            3.9814     0.3356  11.862  < 2e-16 ***
## x2newAfrica                  -8.9236     1.1090  -8.046 1.19e-13 ***
## x2newAmericas                -0.3224     0.9488  -0.340   0.7344
## x2newEastern Mediterranean   -2.7834     1.1419  -2.438   0.0158 *
## x2newSouth-East Asia         -1.1905     1.5283  -0.779   0.4370
## x2newWestern Pacific         -1.1031     1.0709  -1.030   0.3044
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.214 on 177 degrees of freedom
## Multiple R-squared:  0.7732,Adjusted R-squared:  0.7655
## F-statistic: 100.6 on 6 and 177 DF,  p-value: < 2.2e-16
```

A simple plot for looking at the outcome is given here and shown in Figure 3:

```
plot(x1, y, col = as.numeric(x2), pch = 16)
abline(a = b[1], b = b[2], col = 1)
abline(a = b[1] + b[3], b = b[2], col = 2)
abline(a = b[1] + b[4], b = b[2], col = 3)
abline(a = b[1] + b[5], b = b[2], col = 4)
abline(a = b[1] + b[6], b = b[2], col = 5)
abline(a = b[1] + b[7], b = b[2], col = 6)
legend("bottomright", levels(GDPe0$Region), col = 1:7, pch = 16)
```

Figure 3: Actual and fitted values from a linear model for log-GDP per capita vs. life expectancy for 184 countries in 2012.

5 Exercise

5.1 Exercise 1

Using matrix algebra, solve the following system of equations:

$$\begin{cases} 8x + 10y + 11z &= 13 \\ 9x + 10y + 5z &= 11 \\ 14x + 35y + 4z &= 11 \end{cases}$$

finding the vector $[x, y, z]$.

5.2 Exercise 2

For this exercise you need to load some data called `Jims.txt`. As already mentioned they contain three variables:

`year` the observation year

`rle` the record life expectancy

`country` the name of the country that had the `rle`

For this exercise you need to use just the first two variables.

As you may already know, the main insight of Oeppen and Vaupel (2002) is that record life expectancy increased linearly with time during the last century and half. In particular, our interest is to partly reproduce their result using our knowledge in R and linear algebra.

Firstly, just to get an idea, you need to simply plot years vs. record life expectancy.

Then, since they used a linear normal model, you have to find the two coefficients (for the intercept and for the years) using matrix algebra.

Test your outcomes using the R -function `lm`, and plot the fitted lines over the previous plot using `abline()`.

References

Brouhns, N., M. Denuit, and J. K. Vermunt (2002). A poisson log-bilinear regression approach to the construction of projected lifetables. *Insurance: Mathematics and Economics 31*(3), 373–393.

Caswell, H. (2001). *Matrix population models: construction, analysis, and interpretation*. Sinaur Associates, Inc. Publishers.

Keyfitz, N. and H. Caswell (2005). *Applied mathematical demography*, Volume 47. Springer.

Lee, R. D. and L. R. Carter (1992). Modeling and forecasting us mortality. *Journal of the American statistical association 87*(419), 659–671.

McCullagh, P. and J. A. Nelder (1989). *Generalized linear models*, Volume 37. CRC press.

Oeppen, J. and J. W. Vaupel (2002). Broken limits to life expectancy. *Science 296*(5570), 1029–1031.

Preston, S. H., P. Heuveline, and M. Guillot (2001). *Demography: Measuring and modeling population processes*. Oxford: Blackwell.