Matrix calculations in R

Karline Soetaert
Centre for Estuarine and Marine Ecology
Netherlands Institute of Ecology
The Netherlands

Abstract

This document gives an overview of R functions that deal with matrices and sets of linear equations. It includes linear and quadratic programming.

Keywords: linear equations, matrices, quadratic programming, linear programming, R.

1. Introduction

Many of the linear algebra functions are part of the R core package (R Development Core Team 2008). However, some additional packages are recommended:

- **Matrix** (Bates and Maechler 2008). Matrix calculations for full and sparse matrices.
- **SparseM** (Koenker and Ng 2008). Matrix calculations for arbitrary sparse matrices.
- **MASS** (Venables and Ripley 2002). Generalised inverse, null spaces of matrices
- **limSolve** (Soetaert, Van den Meersche, and van Oevelen 2008). Least squares, quadratic and linear programming
- **quadprog** (Weingessel 2007). Quadratic programming
- **lpSolve** (Berkelaar *et al.* 2007). Linear and integer programming

2. Matrix algebra - basics

A **matrix** is a rectangular array of numbers. A matrix with m rows and n columns is said to be of order m x n; (m,n) is also referred to as the dimension of the matrix.

In expanded form, a matrix is written with coefficients aij for each entry; the first subscript refers to the rows, the second to the columns. A matrix is often symbolized with an uppercase, bold-faced letter.

$$\mathbf{A}_{2x2} = \begin{bmatrix} a11 & a12 \\ a21 & a22 \end{bmatrix}$$

A matrix with column order one is called a **column vector**. A **row vector** is a matrix with one row and several columns.

A **diagonal matrix** is one for which the values off diagonal are 0.

The **identity matrix I** is a special form of a diagonal matrix where the diagonal elements are 1.

A matrix is called **triangular** if all elements on one side of the diagonal are 0. An upper triangular matrix has non-zero elements only above the diagonal; a lower triangular matrix has non-zero elements below the diagonal:

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 5 & 6 & 7 & 0 \\ 8 & 9 & 10 & 11 \end{bmatrix}$$

In R, matrices can be created in several ways:

- By means of R-function `matrix`
- By means of R-function `diag` which constructs a diagonal matrix
- The functions `cbind` and `rbind` add columns and rows to an existing matrix, or to another vector

The following statement creates a matrix A, with two rows, and, as there are four elements, two columns.

```
> A <-matrix(nrow=2,data=c(1,2,3,4))
> A

     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Note that the data are inputted as a vector (using the c() function).

By default, R fills a matrix column-wise (see the example above). However, this can easily be overruled, using parameter `byrow`:

```
> (M <-matrix(nrow=4, ncol=3, byrow=TRUE, data=1:12))

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

Matrices can also be created by combining (binding) vectors, rowwise (`rbind`) or columnwise (`cbind`):

```
> V <- 0.5:5.5
> rbind(V,sqrt(V))
```

```
       [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
V 0.5000000 1.500000 2.500000 3.500000 4.500000 5.500000
  0.7071068 1.224745 1.581139 1.870829 2.121320 2.345208
```

A diagonal matrix is created using R-function `diag`:

```
> diag(nrow=2)
```

```
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

```
> diag(x= 1:2,nrow =2, ncol=3)
```

```
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    2    0
```

Upper and lower triangular matrices can be created with `upper.tri` and `lower.tri`

```
> m2 <- matrix(nr=4,data=1:20)
> m2[lower.tri(m2)]<-0
> m2
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    0    6   10   14   18
[3,]    0    0   11   15   19
[4,]    0    0    0   16   20
```

3. Matrix operations

The **product** between matrices A and B, written as AB, or $A \cdot B$ is only defined if the number of columns of A equals the number of rows of B. The product of a matrix with order m x n with a matrix with order n x p gives a new matrix with order m x p and whose ij-th element equals: $\sum_{k=1}^{n} a_{ik} b_{kj}$.

In R, matrix multiplication is done using `%*%`.

```
> (A <- matrix(nrow=2,ncol=3,data=6:1))
```

```
     [,1] [,2] [,3]
[1,]    6    4    2
[2,]    5    3    1
```

```
> B <- matrix(nrow=3,ncol=4,data=1:12)
> A%*%B
```

```
     [,1] [,2] [,3] [,4]
[1,]   20   56   92  128
[2,]   14   41   68   95
```

The identity matrix plays the same role as the number 1 in scalar notation, this is: $AI = A = IA$

```
> A%*%diag(3)
```

```
     [,1] [,2] [,3]
[1,]    6    4    2
[2,]    5    3    1
```

Multiplication of a matrix **A** with a **scalar**, c, results in a new matrix with the same dimension as **A** and whose ij-th element equals $c \cdot a_{ij}$

```
> 2*A
```

```
     [,1] [,2] [,3]
[1,]   12    8    4
[2,]   10    6    2
```

Addition and subtraction can only be performed between matrices of the same order. It is done element by element: the resulting matrix has the same order and its ij-th element equals $a_{ij} + b_{ij}$.

```
> A+A
```

```
     [,1] [,2] [,3]
[1,]   12    8    4
[2,]   10    6    2
```

The sum of the elements that fall on the diagonal, $\sum_{i=1}^{n} a_{ii}$ is called the **trace** of the matrix.

```
> sum(diag(A))
```

```
[1] 9
```

The **transpose** of an mxn matrix A, is a nxm matrix denoted by A^T or A' and having rows identical to the columns of A and vice-versa:

$$A^T = \begin{bmatrix} a11 & a21 \\ a12 & a22 \end{bmatrix}$$

Note: $(A^T)^T = A$; $(A + B)^T = A^T + B^T$; $(A \cdot B)^T = B^T \cdot A^T$

R -function `t()` takes the matrix transpose.

```
> (A <- matrix(nrow=2,data=1:4))

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

> t(A)

     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

The **inverse** of a matrix A is the unique matrix A^{-1} which, when multiplied with A produces the identity matrix I: $A \cdot A^{-1} = I$. The inverse can only be taken from square matrices; not all square matrices have an inverse. Note that $(cA)^{-1} = c^{-1} \cdot A^{-1}$; $(A \cdot B)^{-1} = B^{-1} \cdot A^{-1}$

R -function `solve()` takes the matrix inverse.

```
> solve(A)

     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5

> solve(A)%*%A

     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

An **orthogonal matrix** T is a matrix whose transpose equals its inverse:

$$T^T = T^{-1}$$

Thus: $T^T \cdot T = I$, the identity matrix. Orthogonal matrices can be created by the QR transformation (see below).

```
> (A  <- matrix(nrow=2,ncol=2,data=1))

     [,1] [,2]
[1,]    1    1
[2,]    1    1

> (AO <- qr.Q(qr(A), complete = TRUE))

           [,1]       [,2]
[1,] -0.7071068 -0.7071068
[2,] -0.7071068  0.7071068
```

```
>  AO%*%t(AO)

              [,1]          [,2]
[1,]  1.000000e+00 -1.008850e-16
[2,] -1.008850e-16  1.000000e+00
```

4. Eigenvalues and eigenvectors, determinants

Given a square (m x m) real matrix A, an **eigenvalue** λ is a scalar for which: $A \cdot x = \lambda x$ for $x \neq 0$:

$$
\begin{aligned}
&a_{11} \cdot x_1 + a_{12} \cdot x_2 + \ldots a_{1n} \cdot x_n = \lambda \cdot x_1 \\
&a_{21} \cdot x_1 + a_{22} \cdot x_2 + \ldots a_{2n} \cdot x_n = \lambda \cdot x_2 \\
&\ldots \\
&a_{n1} \cdot x_1 + a_{n2} \cdot x_2 + \ldots a_{nn} \cdot x_n = \lambda \cdot x_n
\end{aligned}
$$

x is called the **(right) eigenvector** of A for the eigenvalue λ; the eigenvector is defined only up to a scaling factor.

Note: the eigenvalues of A^{-1} are the reciprocals of the eigenvalues of A.

For symmetric A the eigenvalues are real (otherwise they might be complex numbers). A matrix A is said to be **nonsingular** if it has only nonzero eigenvalues. A matrix is **positive definite** if all of its eigenvalues are positive.

The **determinant** of a matrix is the product of its eigenvalues.

In R the eigenvalue and (right) eigenvectors of a matrix are calculated by funcion `eigen`. It returns a list that contains both the eigenvalues ($values) and the eigenvectors ($vectors). The latter are represented by the columns of the matrix.

```
> (er <- eigen(A))

$values
[1] 2 0

$vectors
          [,1]       [,2]
[1,] 0.7071068 -0.7071068
[2,] 0.7071068  0.7071068

> A%*%er$vectors[,1]-er$values[1]*er$vectors[,1]

     [,1]
[1,]    0
[2,]    0

> A%*%er$vectors[,2]-er$values[2]*er$vectors[,2]
```

```
     [,1]
[1,]    0
[2,]    0
```

The **left eigen vectors** of A equal the right eigenvectors of its transpose.

```
> (el <- eigen(t(A)))

$values
[1] 2 0

$vectors
          [,1]       [,2]
[1,] 0.7071068 -0.7071068
[2,] 0.7071068  0.7071068

> el$vectors[,1]%*%A-el$values[1]*el$vectors[,1]

     [,1] [,2]
[1,]    0    0

> el$vectors[,2]%*%A-el$values[2]*el$vectors[,2]

     [,1] [,2]
[1,]    0    0
```

The determinant of A is calculated with R -function `det`.

```
> det(A)

[1] 0
```

5. linear equations

The n linear equations in the m unknowns x

$$\begin{aligned} &a_{11} \cdot x_1 + a_{12} \cdot x_2 + \ldots a_{1n} \cdot x_n = b_1 \\ &\ldots \\ &a_{m1} \cdot x_1 + a_{m2} \cdot x_2 + \ldots a_{mn} \cdot x_n = b_m \end{aligned}$$

can be written in matrix form:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$$

Where **A(mxn)** is called the coefficient matrix, vector **x** contains the unknowns and **b** is called the right hand side.

Vectors x_1, ... x_N are called **linearly dependent** if there exist numbers l_1,... l_N, all non-zero, such that:

$$l_1 \cdot \mathbf{x_1} + ... + l_N \cdot \mathbf{x_N} = \mathbf{0}$$

The vectors are **linearly independent** if no such numbers can be found.

The **rank** of a matrix is the minimum of (the maximum number of linearly independent rows or maximum number of linearly independent columns). It is denoted by r(A) or rank(A). Note that for a matrix A(m x n): $0 \leq r(A) \leq \min(m, n)$

A **basis of the null-space** of a matrix, M, is a matrix N such that:

$$N^t \cdot M = 0$$

and where N has the maximum number of linearly independent columns.

If A is square and positive definite, the solution of the linear system $Ax = b$ is done using R's function `solve`:

```
> A <- matrix(nrow=2,data=1:4)
> B <- c(5,6)
> X <- solve(A,B)
> A%*%X-B

     [,1]
[1,]    0
[2,]    0
```

To estimate the rank of matrix, it is easiest to use the QR decomposition. Alternatively, the singular value decomposition can be used (see below).

```
> A <- matrix(nrow=4,ncol=4,data=c(1:8,6,8,10,12,1:4))
> A    # col3=col1+col2; col4=col1

     [,1] [,2] [,3] [,4]
[1,]    1    5    6    1
[2,]    2    6    8    2
[3,]    3    7   10    3
[4,]    4    8   12    4

> qr(A)$rank

[1] 2
```

R -pakage **MASS** provides a function to estimate a basis for the null space of a matrix:

```
> Null(A)

           [,1]        [,2]
[1,] -0.4000874 -0.37407225
[2,]  0.2546329  0.79697056
[3,]  0.6909965 -0.47172438
[4,] -0.5455419  0.04882607
```

```
> t(Null(A))%*%A
```

```
              [,1]          [,2]          [,3]          [,4]
[1,] 4.440892e-16  0.000000e+00 4.440892e-16 4.440892e-16
[2,] 4.718448e-16 -3.885781e-16 8.326673e-17 4.718448e-16
```

6. Kronecker product

The kronecker product of two matrices ($\otimes$) is an operation on two matrices of arbitrary size resulting in a block matrix.

Example:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \otimes \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{11}b_{13} & a_{12}b_{11} & a_{12}b_{12} & a_{12}b_{13} \\ a_{11}b_{21} & a_{11}b_{22} & a_{11}b_{23} & a_{12}b_{21} & a_{12}b_{22} & a_{12}b_{23} \\ a_{21}b_{11} & a_{21}b_{12} & a_{21}b_{13} & a_{22}b_{11} & a_{22}b_{12} & a_{22}b_{13} \\ a_{21}b_{21} & a_{21}b_{22} & a_{21}b_{23} & a_{22}b_{21} & a_{22}b_{22} & a_{22}b_{23} \\ a_{31}b_{11} & a_{31}b_{12} & a_{31}b_{13} & a_{32}b_{11} & a_{32}b_{12} & a_{32}b_{13} \\ a_{31}b_{21} & a_{31}b_{22} & a_{31}b_{23} & a_{32}b_{21} & a_{32}b_{22} & a_{32}b_{23} \end{bmatrix}$$

In R , this operation can be performed using `kronecker`.

Kronecker products can be used to estimate the coefficients of a linear equation matrix. Consider the problem where the linear equality equation $Ax = b$, has to be solved for the unknown coefficients of the matrix A, i.e. x is known, A is unknown.

The set of equations with the coefficients of A as the unknowns is given by:

$$(\mathbf{x}^T \otimes \mathbf{I}) \cdot \mathbf{A} = \mathbf{b}$$

where $\mathbf{I}$ is the identity matrix of appropriate dimension, and vec($\mathbf{A}$) is a vector containing the coefficients of A, columnwise.

```
> x <- c(1,2)
> kronecker(t(x),diag(2))
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    0    2    0
[2,]    0    1    0    2
```

7. Singular value decomposition

The **singular value decomposition** (SVD), sometimes called the spectral decomposition, is a very important decomposition.

It forms the basis of the generalized inverse of a matrix (see below).

An m x n matrix A can be decomposed as:

$$A = U \cdot D \cdot V^T$$

where

$U_{(mxp)}$ and $V_{(nxp)}$ are column-orthonormal matrices, i.e. they satisfy :

$U \cdot U^T = I$ and $V \cdot V^T = I$,

I is the identity matrix of appropriate dimension.

U contains the left singular vectors, V the right singular vectors and

D(p x p) is a diagonal matrix containing the 'singular values σ_i, which are positive values, generally arranged in decreasing order, i.e. $D = diag(\sigma_1, \sigma_2, \ldots \sigma_p)$ with $\sigma_1 \geq \sigma_2 \geq, \ldots \geq \sigma_p$. Here p is the minimum of (m,n), i.e. the initial 'guess' of the rank of matrix A.

The rank r can be estimated by counting the number of non-zero singular values in D.

In order to count a singular value as zero, we have to decide on the tolerance to use, generally this is taken as a very small number (e.g. the root of machine precision in the order of 1e-8).

The approximation

$$A \simeq U_{(mxr)} D^{-1}_{(nxr)} V^T_{(nxr)}$$

is called the *low-rank or truncated singular value decomposition* of A. If singular values are truncated at sufficiently low values, the approximation will be near exact.

8. The generalized inverse

The **generalized inverse** of a matrix A, denoted by A^- is one for which

$$A \cdot A^- \cdot A = A$$

In contrast to the inverse A^{-1}, which only exists for certain square matrices, the generalized inverse always exists although it need not be unique. If the inverse of A exists, the generalized inverse is equal to the inverse.

It is constructed by singular value decomposition as follows:

If $A = U \cdot D \cdot V^T$

Then $A^- = V_{(nxr)} \cdot D^{-1}_{(nxr)} \cdot U^T_{(mxr)}$

And where $D^{-1}_{(rxr)} = diag(1/\sigma_1, 1/\sigma_2, \ldots 1/\sigma_r)$.

Here the matrices $U_{(mxr)}$, $D_{(rxr)}$ and $V_{(nxr)}$ use the actual (or better estimated) rank r of the matrix A, which can be smaller than p.

The generalized inverse forms the basis of solving any set of linear equations. A particular solution of the equation $A \cdot x = b$, where A is an m x n matrix, using the generalized inverse is written as: $x = A^- \cdot b$

We consider 3 cases.

- $m = n = rank(A)$, there are as many equations as unknowns. In this case, $A^- = A^{-1}$ and the solution is unique and given by: $x = A^{-1} \cdot b$

- $m > n \geq rank(A)$, there are more equations than unknowns. The equations are said to be *overdetermined*. The generalized inverse solution is the one that minimizes the sum of squared residuals, i.e. $min(\|Ax - b\|^2)$.

- $m < n \leq rank(A)$, there are more unknowns than equations. The system is said to be *underdetermined*, and there are an infinite number of solutions. The particular generalized inverse solution is the one that minimizes the sum of squared x's, i.e.$\min(\|x\|^2)$. This solution is often called the 'parsimonious' solution.

  A more general set of solutions is given by: $X = A^- \cdot B + (I - A^- \cdot A) \cdot z$ Where z is a vector of length n, with arbitrary numbers

The *'data' resolution matrix* $A \cdot A^-$ describes how well the equations match the right hand side. The elements on the diagonal denote the importance of the equations in determining the values of the unknowns. Values less than 1 indicate that the equations are not fully independent.

The *'model' resolution matrix* $A^- \cdot A$ characterizes whether the unknowns can be uniquely determined. A value on the diagonal less than 1 indicates that the unknowns are not fully constrained.

In the R -script below, a rectangular matrix is created (1st line), the singular value decomposition taken and shown (2nd line). The machine precision (3rd line) is then used to select the singular values that are sufficiently large (i.e. larger than the precision, 4th line). The rank of the matrix is then the number of sufficiently large singular values (5th line). (but it is simpler to use the QR decomposition for rank calculation).

```
> A <- matrix(nrow=4,ncol=3,data=c(1:8,6,8,10,12))
> (s      <- svd(A))

$d
[1] 2.337183e+01 1.325693e+00 1.043941e-15

$u
           [,1]       [,2]        [,3]
[1,] -0.3340803 -0.7670661  0.53211594
[2,] -0.4359333 -0.3316054 -0.80624830
[3,] -0.5377863  0.1038552  0.01614878
[4,] -0.6396393  0.5393158  0.25798358

$v
           [,1]       [,2]       [,3]
[1,] -0.2301002  0.7834032  0.5773503
[2,] -0.5633970 -0.5909742  0.5773503
[3,] -0.7934972  0.1924290 -0.5773503

> tol     <- sqrt(.Machine$double.eps)
> nonzero <- which(s$d > tol * s$d[1])
> (rank   <- length(nonzero))
```

```
[1] 2

> qr(A)$rank   # this is easier...

[1] 2
```

The generalized inverse is taken, and right- and left multiplied with A, which returns the original matrix.

```
> gA  <- ginv(A)
> A %*% gA %*% A

     [,1] [,2] [,3]
[1,]    1    5    6
[2,]    2    6    8
[3,]    3    7   10
[4,]    4    8   12
```

After creating a vector B, which forms the right hand side of the linear set of equations $A \cdot x = B$, the parsimonious solution is calculated The third line checks if the residuals are indeed 0. Finally, the resolutions of equations and variables are estimated.

```
> B <- 0:3
> (X <- gA %*% B)

           [,1]
[1,]  0.9166667
[2,] -0.5833333
[3,]  0.3333333

> A %*% X - B                          # should be zero

              [,1]
[1,] -1.110223e-16
[2,]  0.000000e+00
[3,]  0.000000e+00
[4,]  0.000000e+00

> res.eq   <- diag(A %*%gA)
> res.var  <- diag(gA%*% A)
```

The previous calculations can also be done using functions from package **limSolve** .

```
> Solve(A,B)

[1]  0.9166667 -0.5833333  0.3333333
```

```
> resolution(A)

$row
[1] 0.7 0.3 0.3 0.7

$col
[1] 0.6666667 0.6666667 0.6666667

$nsolvable
[1] 2
```

9. Other matrix decompositions

9.1. QR

One important decomposition is the QR decomposition, which decomposes a rectangular matrix A into an upper triangular matrix R and an orthogonal matrix Q:

$$A = Q \cdot R$$

As Q is orthogonal, it follows that $Q^T \cdot Q = I$. The QR decomposition was used above to estimate the rank of a matrix.

```
> A <- matrix(nrow=4,ncol=3,data=c(1:8,6,8,10,12))
> qrA<-qr(A)
> qr.Q(qrA)                # Q

           [,1]          [,2]       [,3]
[1,] -0.1825742 -8.164966e-01 -0.4000874
[2,] -0.3651484 -4.082483e-01  0.2546329
[3,] -0.5477226 -8.131516e-19  0.6909965
[4,] -0.7302967  4.082483e-01 -0.5455419

> qr.R(qrA)                # R

          [,1]       [,2]          [,3]
[1,] -5.477226 -12.780193 -1.825742e+01
[2,]  0.000000  -3.265986 -3.265986e+00
[3,]  0.000000   0.000000 -1.981890e-16

> qr.Q(qrA) %*% qr.R(qrA) #Q R = A

     [,1] [,2] [,3]
[1,]    1    5    6
[2,]    2    6    8
[3,]    3    7   10
[4,]    4    8   12
```

9.2. Cholesky decomposition

The cholesky decomposition takes the 'square root' of a square matrix.

$$A = R^T \cdot R$$

Similarly as a real root of a negative number does not exist, matrix A has to obey certain characteristics in order for its square root to exist: this is A has to be positive definite.

Note that sometimes, the cholesky decomposition may fail when it should not. Adding a tiny value on the diagonal helps to remedy this:

```
> (A  <- matrix(nrow=2,ncol=2,data=1))

     [,1] [,2]
[1,]    1    1
[2,]    1    1

> #chol(A)                                        # fails
> diag(A) <- diag(A) +  .Machine$double.eps
> (sqA    <- chol(A))                             # works

     [,1]         [,2]
[1,]    1 1.000000e+00
[2,]    0 1.490116e-08

> t(sqA)%*%sqA

     [,1] [,2]
[1,]    1    1
[2,]    1    1
```

9.3. lu (lower-upper) decomposition

The LU decomposition writes a matrix as the product of a lower and upper triangular matrix.

In R , LU decomposition is available by function `lu` from package **Matrix**.

Function `lu` creates the lu factorization in the form:

$$A = P \cdot L \cdot U$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n).

This package makes use of the S4 class system, and does not produce output as a list. Rather, the requested information can be retrieved by calling the appropriate "methods". In addition, it does not take as input an ordinary "matrix", but a class of type "Matrix".

Here is a function based on `lu`, but using the S3 class system [1]:

[1] which, in this case, I find easier to work with

```
> LU <- function(A)
+ {
+ ll<-expand(lu(Matrix(A)))
+ ll$L <- as.matrix(ll$L)
+ ll$U <- as.matrix(ll$U)
+ ll$P <- as.matrix(ll$P)
+ ll
+ }
> (A <- matrix(nr=3,data=runif(9)))

          [,1]      [,2]       [,3]
[1,] 0.5173217 0.6356087 0.27079626
[2,] 0.6966682 0.4762469 0.88531567
[3,] 0.1559702 0.6398133 0.02739824

> (lud <-LU(A))

$L
          [,1]      [,2] [,3]
[1,] 1.0000000 0.0000000    0
[2,] 0.2238801 1.0000000    0
[3,] 0.7425654 0.5288241    1

$U
          [,1]      [,2]       [,3]
[1,] 0.6966682 0.4762469  0.8853157
[2,] 0.0000000 0.5331911 -0.1708064
[3,] 0.0000000 0.0000000 -0.2962820

$P
     [,1] [,2] [,3]
[1,]    0    0    1
[2,]    1    0    0
[3,]    0    1    0

> lud$P%*%lud$L%*%lud$U   #equal to A

          [,1]      [,2]       [,3]
[1,] 0.5173217 0.6356087 0.27079626
[2,] 0.6966682 0.4762469 0.88531567
[3,] 0.1559702 0.6398133 0.02739824
```

(Note: this only works for square matrices A)

10. linear equality and inequality equations

There are many problems where linear equations are supplemented with linear inequality

constraints that have to be met. Formally, the problem can be specified as:

$$\min(f(\mathbf{x})) \qquad or \qquad \max(f(\mathbf{x}))$$

subject to:

$$\mathbf{E} \cdot \mathbf{x} = \mathbf{f}$$
$$\mathbf{G} \cdot \mathbf{x} \geq \mathbf{h}$$

where $\mathbf{x}$ is the vector with the unknowns and either $\mathbf{E}$ and/or $\mathbf{G}$ may be empty. The function $f(\mathbf{x})$ which has to be minimized or maximized need not be linear.

10.1. Quadratic programming

In case the cost function is quadratic, the problem is called a least squares problem with equality and inequality conditions (LSEI):

$$\min(\|\mathbf{A} \cdot \mathbf{x} - \mathbf{b}\|^2) \qquad (1)$$
$$\mathbf{E} \cdot \mathbf{x} = \mathbf{f} \qquad (2)$$
$$\mathbf{G} \cdot \mathbf{x} \geq \mathbf{h} \qquad (3)$$

Some of the equations must be exactly satisfied ($\mathbf{Ex} = \mathbf{f}$), whilst others do not ($\mathbf{Ax} = \mathbf{b}$), and where in addition certain inequality constraints need to be satisfied.

These problems are called "least squares" problems with equality and inequality conditions. They are solved by quadratic programming techniques.

The formalism $\|\|^2$, is also called the L2 norm. It can be rewritten as:

$$\begin{aligned} \|Ax - b\|^2 &= (Ax - b)^T (Ax - b) \\ &= b^T b + x^T A^T A x - 2 b^T A x \end{aligned}$$

As the first term does not contain the unknown $\mathbf{x}$, it can be ignored such that

$$\begin{aligned} &\min \|Ax - b\|^2 = \min(\tfrac{x^T D x}{2} - p^T x) \\ &D = A^T A \\ &p^T = b^T A \end{aligned}$$

Thus, LSEI problems can be inputted either using matrix $\mathbf{A}$ and vector $\mathbf{b}$ or matrix $\mathbf{D}$ and vector $\mathbf{p}$.

Functions `lsei` from package **limSolve** uses the first formalism, while function `solve.QP` uses the second formalism.

The following problem

$$\begin{array}{llllcl}
3 \cdot x_1 & +2 \cdot x_2 & +x_3 & +4 \cdot x_4 & = & 2 \\
x_1 & +x_2 & +x_3 & +x_4 & = & 2 \\
 & & & & & \\
2 \cdot x_1 & +x_2 & +x_3 & +x_4 & \geq & -1 \\
-1 \cdot x_1 & +3 \cdot x_2 & +2 \cdot x_3 & +x_4 & \geq & 2 \\
-1 \cdot x_1 & & +x_3 & & \geq & 1 \\
 & & & & & \\
2 \cdot x_1 & +2 \cdot x_2 & +x_3 & +6 \cdot x_4 & \simeq & 1 \\
x_1 & -x_2 & +x_3 & -x_4 & \simeq & 2
\end{array}$$

is implemented and solved in R as:

```
> E <- matrix(ncol=4, byrow=TRUE,
+             data=c(3,2,1,4,1,1,1,1))
> F <- c(2,2)
> A <- matrix(ncol=4, byrow=TRUE,
+             data=c(2,2,1,6,1,-1,1,-1))
> B <- c(1,2)
> G <-matrix(ncol=4,byrow=TRUE,
+            data=c(2,1,1,1,-1,3,2,1,-1,0,1,0))
> H <- c(-1, 2, 1)
> lsei(E=E,F=F,A=A,B=B,G=G,H=H)$X

[1]  0.3333333  0.3333333  1.6666667 -0.3333333
```

10.2. Linear programming

In case the function to be minimised or maximise is linear, the problem can be solved with linear programming algorithms. They also assume that all x>0.

$$\min(\sum_i a_i x_i) \qquad or \qquad \max(\sum_i a_i x_i)$$

subject to:

$$\begin{array}{c}
\mathbf{E} \cdot \mathbf{x} = \mathbf{f} \\
\mathbf{G} \cdot \mathbf{x} \geq \mathbf{h} \\
\mathbf{x_i} > 0
\end{array}$$

A robust linear programming function, `lp` can be found in package **lpSolve**.

Package **limSolve** provides a wrapper around `lp` such that the input is compatable with the input of function `lsei`. Example:

$$\min(x_1 + 2 \cdot x_2 - 1 \cdot x_3 + 4 \cdot x_4)$$

subject to :

$$\begin{array}{llllll}
3 \cdot x_1 & +2 \cdot x_2 & +x_3 & +4 \cdot x_4 & = & 2 \\
x_1 & +x_2 & +x_3 & +x_4 & = & 2 \\
 & & & & & \\
2 \cdot x_1 & +x_2 & +x_3 & +x_4 & \geq & -1 \\
-1 \cdot x_1 & +3 \cdot x_2 & +2 \cdot x_3 & +x_4 & \geq & 2 \\
-1 \cdot x_1 & & +x_3 & & \geq & 1 \\
 & & & & & \\
x_i & \geq & 0 & \forall i & &
\end{array}$$

is implemented in R as:

```
> E <- matrix(ncol=4, byrow=TRUE,
+             data=c(3,2,1,4,1,1,1,1))
> F <- c(2,2)
> G <-matrix(ncol=4,byrow=TRUE,
+            data=c(2,1,1,1,-1,3,2,1,-1,0,1,0))
> H <- c(-1, 2, 1)
> Cost <- c(1,2,-1,4)
> linp(E,F,G,H,Cost)

$X
[1] 0 0 2 0

$residualNorm
[1] 0

$solutionNorm
[1] -2

$IsError
[1] FALSE

$type
[1] "simplex"
```

Table 1: Summary of matrix functions in R

	R -Function	Package	Description
A^T	t(A)	base	
$A^T \cdot B$	crossprod(A,B)	base	
$A^T \cdot A$	crossprod(A,A)	base	
$A \cdot B^T$	tcrossprod(A,B)	base	
$A \cdot A^T$	tcrossprod(A,A)	base	
$A \otimes B$	kronecker(A,B)	base	kronecker product
solve $Ax = b$ for x	solve(A,B)	base	A=square, positive definite
solve $Ax = b$ for x	Solve(A,B)	limSolve	any A
A^{-1}	solve(A)	base	inverse, A=square, positive definite
A^-	ginv(A)	MASS	Moore-Penrose generalised inverse
A^-	Solve(A)	limSolve	Moore-Penrose generalised inverse
	eigen(A)	base	eigen values and eigen vectors
	svd(A)	base	singular value decomposition of A
	det(A)	base	determinant
	chol(A)	base	Choleski factorization of A
	lu(A)	Matrix	LU decomposition of A
	qr(A)$qr	base	QR decomposition of A
	qr(A)$rank	base	rank of A

References

Bates D, Maechler M (2008). *Matrix: A Matrix package for R*. R package version 0.999375-9.

Berkelaar M, *et al.* (2007). *lpSolve: Interface to Lp_solve v. 5.5 to solve linear or integer programs*. R package version 5.5.8.

Koenker R, Ng P (2008). *SparseM: Sparse Linear Algebra*. R package version 0.75, URL http://www.econ.uiuc.edu/~roger/research/sparse/sparse.html.

R Development Core Team (2008). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Soetaert K, Van den Meersche K, van Oevelen D (2008). *limSolve: Solving linear inverse models*. R package version 1.2.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. Springer, New York, fourth edition. ISBN 0-387-95457-0, URL http://www.stats.ox.ac.uk/pub/MASS4.

Weingessel A (2007). *quadprog: Functions to solve Quadratic Programming Problems.S original by Berwin A. Turlach R port by Andreas Weingessel*. R package version 1.4-11.

Affiliation:

Karline Soetaert
Centre for Estuarine and Marine Ecology (CEME)
Netherlands Institute of Ecology (NIOO)
4401 NT Yerseke, Netherlands E-mail: k.soetaert@nioo.knaw.nl
URL: http://www.nioo.knaw.nl/ppages/ksoetaert