Package limSolve , solving linear inverse models in R

Karline Soetaert, Karel Van den Meersche and Dick van Oevelen
Centre for Estuarine and Marine Ecology
Netherlands Institute of Ecology
The Netherlands

Abstract

R package **limSolve** (?) solves linear inverse models (LIM), consisting of linear equality and or linear inequality conditions, which may be supplemented with approximate linear equations, or a target (cost, profit) function. Depending on the determinacy of these models, they can be solved by least squares or linear programming techniques, by calculating ranges of unknowns or by randomly sampling the feasible solution space.

Amongst the possible scientific applications are: food web quantification (ecology), flux balance analysis (e.g. quantification of metabolic networks, systems biology), compositional estimation (ecology, chemistry,...), and operations research problems. Package **limSolve** contains examples of these four application domains.

In addition, **limSolve** also contains special-purpose solvers for sparse linear equations (banded, tridiagonal, block diagonal)

Keywords: Linear inverse models, food web models, flux balance analysis, linear programming, quadratic programming, R.

1. Introduction

In matrix notation, linear inverse problems are defined as: [1]

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

There are three sets of linear equations: equalities that have to be met as closely as possible (1), equalities that have to be met exactly (2) and inequalities (3).

Depending on the active set of equalities (2) and constraints (3), the system may either be underdetermined, even determined, or overdetermined. Solving these problems requires different mathematical techniques.

[1] notations: vectors and matrices are in **bold**; scalars in normal font. Vectors are indicated with a small letter; matrices with capital letter.

2. Even determined systems

An even determined problem has as many (independent and consistent) equations as unknowns. There is only one solution that satisfies the equations exactly.

Even determined systems that do not comprise inequalities, can be solved with R function `solve`, or -more generally- with **limSolve** function `Solve`. The latter is based on the Moore-Penrose generalised inverse method, and can solve any linear system of equations.

In case the model is even determined, and if $\mathbf{E}$ is square and positive definite, `Solve` returns the same solution as function `solve`. The function uses function `ginv` from package **MASS** (**?**).

Consider the following set of linear equations:

$$\begin{array}{llllll} 3 \cdot x_1 & +2 \cdot x_2 & +x_3 & = & 2 \\ x_1 & & & = & 1 \\ 2 \cdot x_1 & & +2 \cdot x_3 & = & 8 \end{array}$$

which, in matrix notation is:

$$\begin{bmatrix} 3 & 2 & 1 \\ 1 & 0 & 0 \\ 2 & 0 & 2 \end{bmatrix} \cdot \mathbf{X} = \begin{bmatrix} 2 \\ 1 \\ 8 \end{bmatrix}$$

where $\mathbf{X} = [x_1, x_2, x_3]^T$.

In R we write:

```
> E <- matrix(nrow=3,ncol=3,
+              data=c(3,1,2,2,0,0,1,0,2))
> F <- c(2,1,8)
> solve(E,F)

[1]  1 -2  3

> Solve(E,F)

[1]  1 -2  3
```

In the next example, an additional equation, which is a linear combination of the first two is added to the model (i.e. $eq_4 = eq_1 + eq_2$).

As one set of equations is redundant, this problem is equivalent to the previous one. It is even determined although it contains 4 equations and only 3 unknowns.

As the input matrix is not square, this model can only be solved with function `Solve`

$$\begin{bmatrix} 3 & 2 & 1 \\ 1 & 0 & 0 \\ 2 & 0 & 2 \\ 4 & 2 & 1 \end{bmatrix} \cdot \mathbf{X} = \begin{bmatrix} 2 \\ 1 \\ 8 \\ 3 \end{bmatrix}$$

```
> E2 <- rbind(E,E[1,]+E[2,])
> F2 <- c(F,F[1]+F[2])
> #solve(E2,F2) # error
> Solve(E2,F2)

[1]  1 -2  3
```

3. Overdetermined systems

Overdetermined linear systems contain more independent equations than unknowns. In this case, there is only one solution in the least squares sense, i.e. a solution that satisfies:

$$\min_x \|\mathbf{A} \cdot \mathbf{x} - \mathbf{b}\|^2$$

.

The least squares solution can be singled out by function `lsei` (*l*east *s*quares with *e*qualities and *i*nequalities).

If argument `fulloutput` is `TRUE`, this function also returns the parameter covariance matrix, which gives indication on the confidence interval and relations among the estimated unknowns.

3.1. Equalities only

If there are no inequalities, then the least squares solution can also be estimated with `Solve`. The following problem:

$$\begin{bmatrix} 3 & 2 & 1 \\ 1 & 0 & 0 \\ 2 & 0 & 2 \\ 0 & 1 & 0 \end{bmatrix} \cdot \mathbf{X} = \begin{bmatrix} 2 \\ 1 \\ 8 \\ 3 \end{bmatrix}$$

is solved in R as follows:

```
> A <- matrix(nrow=4,ncol=3,
+             data=c(3,1,2,0,2,0,0,1,1,0,2,0))
> B <- c(2,1,8,3)
> lsei(A=A,B=B,fulloutput=TRUE)

$X
[1] -1.1621621  0.8378378  4.8918918

$residualNorm
[1] 0

$solutionNorm
[1] 10.81081

$IsError
```

```
[1] FALSE

$type
[1] "lsei"
```

Here the *residualNorm* is the sum of absolute values of the residuals of the equalities that have to be met exactly ($\mathbf{E} \cdot \mathbf{x} = \mathbf{f}$) and of the violated inequalities ($\mathbf{G} \cdot \mathbf{x} \geq \mathbf{h}$). As in this case, there are none of those, this quantity is 0.

The *solutionNorm* is the value of the minimised quadratic function at the solution, i.e. the value of $\|\mathbf{A} \cdot \mathbf{x} - \mathbf{b}\|^2$.

The *covar* is the variance-covariance matrix of the unknowns. *RankEq* and *RankApp* give the rank of the equalities and of the approximate equalities respectively.

Alternatively, the problem can be solved by `Solve`:

```
> Solve(A,B)

[1] -1.1621622  0.8378378  4.8918919
```

3.2. Equalities and inequalities

If, in addition to the equalities, there are also inequalities, then `lsei` is the only method to find the least squares solution.

With the following inequalities:

$$\begin{aligned} x_1 - 2 \cdot x_2 &< 3 \\ x_1 - x_3 &> 2 \end{aligned}$$

the R-code becomes:

```
> G <- matrix(nrow=2, ncol=3, byrow=TRUE,
+             data=c(-1,2,0,1,0,-1))
> H <- c(-3,2)
> lsei(A=A,B=B,G=G,H=H)

$X
[1]  2.04142012 -0.47928994  0.04142012

$residualNorm
[1] 0

$solutionNorm
[1] 38.17751

$IsError
[1] FALSE

$type
[1] "lsei"
```

4. Underdetermined systems

Underdetermined linear systems contain less independent equations than unknowns. If the equations are consistent, there exist an infinite amount of solutions. To solve such models, there are several options:

- `ldei` - finds the "*l*east *d*istance" (or parsimonious) solution, i.e. the one where the sum of squared unknowns is minimal
- `lsei`- minimises some other set of linear functions ($\mathbf{A} \cdot \mathbf{x} \simeq \mathbf{b}$) in a least squares sense.
- `linp` - finds the solution where **one** linear function (i.e. the sum of flows) is either minimized (a "cost" function) or maximized (a "profit" function). Uses linear programming.
- `xranges` - finds the possible ranges ([min,max]) for each unknown.
- `xsample` - randomly samples the solution space in a Bayesian way. This method returns the conditional probability density function for each unknown.

4.1. Equalities only

We start with an example including only equalities.

$$\begin{array}{llll} 3 \cdot x_1 & +2 \cdot x_2 & +x_3 & = & 2 \\ x_1 & & & = & 1 \end{array}$$

Functions `Solve` and `ldei` retrieve the **parsimonious** solution, i.e. the solution for which $\sum x_i^2$ is minimal.

```
> E <- matrix(nrow=2, ncol=3,
+              data=c(3,1,2,0,1,0))
> F <- c(2,1)
> Solve(E,F)

[1]  1.0 -0.4 -0.2

> ldei(E=E,F=F)$X

[1]  1.0 -0.4 -0.2
```

It is slightly more complex to select the parsimonious solution using `lsei`. Here the approximate equations ($\mathbf{A}$, the identity matrix, and $\mathbf{b}$) have to be specified.

```
> lsei(E=E,F=F,A=diag(3),B=rep(0,3))$X

[1]  1.0 -0.4 -0.2
```

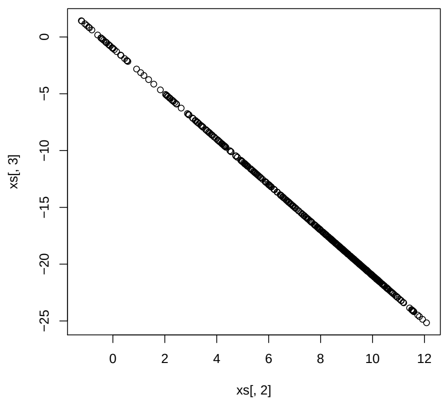

Figure 1: Random sample of the underdetermined system including only equalities. See text for explanation.

It is also possible to **randomly sample** the solution space. This demonstrates that all valid solutions of x_2 and x_3 are located on a line (figure **??**).

```
> xs <- xsample(E=E,F=F,iter=500)$X
> plot(xs[,2],xs[,3])
```

4.2. Equalities and inequalities

Consider the following set of linear equations:

$$\begin{array}{lllllll} 3 \cdot x_1 & +2 \cdot x_2 & +x_3 & +4 \cdot x_4 & = & 2 \\ x_1 & +x_2 & +x_3 & +x_4 & = & 2 \end{array}$$

complemented with the inequalities:

$$\begin{array}{lllllll} 2 \cdot x_1 & +x_2 & +x_3 & +x_4 & \geq & -1 \\ -1 \cdot x_1 & +3 \cdot x_2 & +2 \cdot x_3 & +x_4 & \geq & 2 \\ -1 \cdot x_1 & & +x_3 & & \geq & 1 \end{array}$$

As before, the **parsimonious** solution (that minimises the sum of squared flows) can be found by functions `ldei` and `lsei`.

```
> 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)
> ldei(E,F,G=G,H=H)$X
```

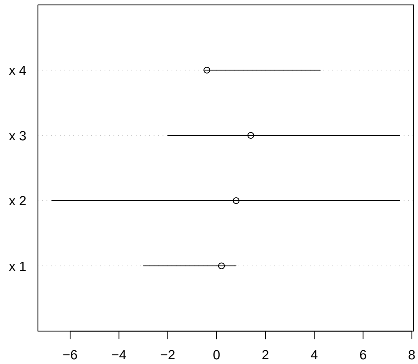

Figure 2: Parsimonious solution and ranges of the underdetermined system including equalities and inequalities. See text for explanation.

```
[1]  0.2  0.8  1.4 -0.4
```

```
> (pars<-lsei(E,F,G=G,H=H,A=diag(nrow=4),B=rep(0,4))$X)
```

```
[1]  0.2  0.8  1.4 -0.4
```

We can also estimate the **ranges** (minimal and maximal values) of all unknowns using function `xranges`.

```
> (xr<-xranges(E,F,G=G,H=H))
```

```
       min  max
[1,] -3.00 0.80
[2,] -6.75 7.50
[3,] -2.00 7.50
[4,] -0.50 4.25
```

The results are conveniently plotted using R function `dotchart` (figure **??**). We plot the parsimonious solution as a point, the range as a horizontal line.

```
> dotchart(pars,xlim=range(c(pars,xr)),label=paste("x",1:4,""))
> segments(x0=xr[,1],x1=xr[,2],y0=1:nrow(xr),y1=1:nrow(xr))
```

A **random sample** of the infinite amount of solutions is generated by function `xsample`. For small problems, the coordinates directions algorithm (”cda”) is a good choice.

```
> xs <- xsample(E=E,F=F,G=G,H=H,type="cda")$X
```

To visualise its output, we use R function `pairs`, with a density function on the diagonal, and without plotting the upper panel (figure **??**).

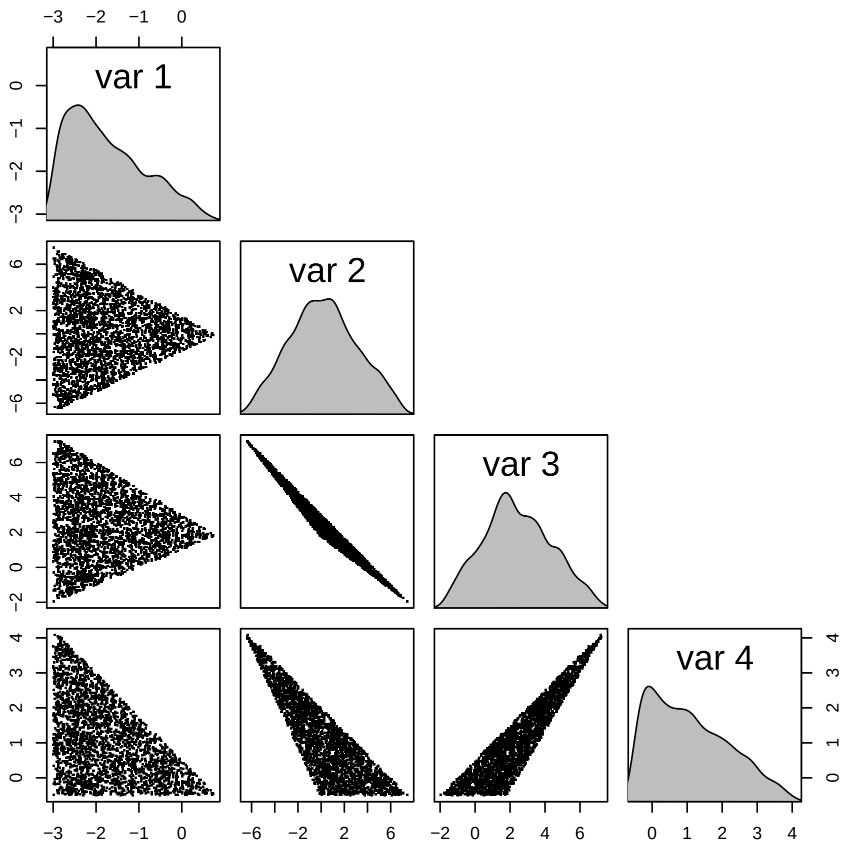

Figure 3: Random sample of the underdetermined system including equalities and inequalities. See text for explanation.

```
> panel.dens <- function(x, ...)
+ {
+     usr <- par("usr"); on.exit(par(usr))
+     par(usr = c(usr[1:2], 0, 1.5) )
+     DD <- density(x); DD$y<- DD$y/max(DD$y)
+     polygon(DD,col="grey")
+ }
> pairs(xs,pch=".",cex=2,upper.panel=NULL,diag.panel=panel.dens)
```

Assume that we define the following variable:

$$v_1 = x_1 + 2 \cdot x_2 - x_3 + x_4 + 2$$

We can use functions `varranges` and `varsample` to estimate its ranges and create a random sample respectively.

Variables are written as a matrix equation:

$$\mathbf{Va} \cdot \mathbf{x} = \mathbf{Vb}$$

```
> Va<-c(1,2,-1,1)
> Vb<- -2
> varranges(E,F,G=G,H=H,EqA=Va,EqB=Vb)

        min  max
[1,] -17.75 15.5

> summary(varsample(xs,EqA=Va,EqB=Vb))

       V1
 Min.   :-16.8138
 1st Qu.: -5.4238
 Median : -0.2374
 Mean   : -0.6488
 3rd Qu.:  4.0629
 Max.   : 15.3491
```

4.3. Equalities, inequalities and approximate equations

The following problem

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

is implemented and solved in R as:

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

[1]  0.3333333  0.3333333  1.6666667 -0.3333333
```

Function `xsample` **randomly samples** the underdetermined problem (using the metropolis algorithm), selecting likely values given the approximate equations. The probability distribution of the latter is assumed Gaussian, with given standard deviation (argument `sdB`, here assumed 1).

The jump length (argument `jmp`) is finetuned such that a sufficient number of trials (˜ 30%), but not too many, is accepted. Note how the ultimate distribution is determined both by the inequality constraints (the sharp edges) as well as by the approximate equations (figure **??**).

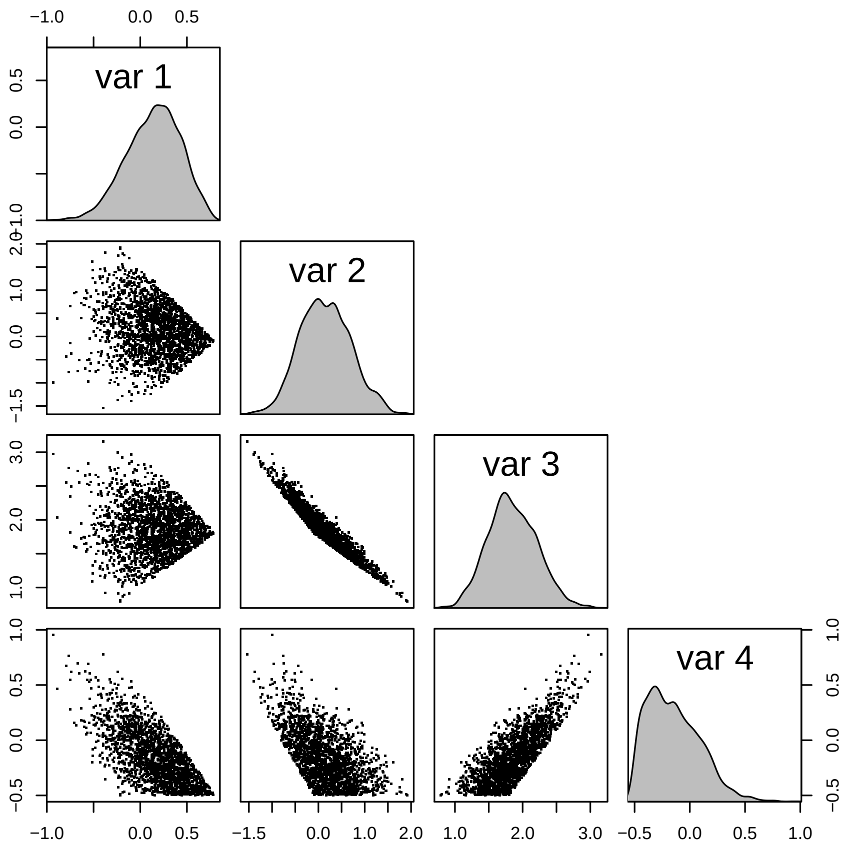

Figure 4: Random sample of the underdetermined system including equalities, inequalities, and approximate equations. See text for explanation.

```
> panel.dens <- function(x, ...)
+ {
+     usr <- par("usr"); on.exit(par(usr))
+     par(usr = c(usr[1:2], 0, 1.5) )
+     DD <- density(x); DD$y<- DD$y/max(DD$y)
+     polygon(DD,col="grey")
+ }
> xs <- xsample(E=E,F=F,G=G,H=H,A=A,B=B,jmp=0.5, sdB=1)$X
> pairs(xs,pch=".",cex=2,upper.panel=NULL,diag.panel=panel.dens)
```

4.4. Equalities and inequalities and a target function

Another way to single out one solution out of the infinite amount of valid solutions is by minimising or maximising a linear target function, using linear programming. For instance,

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

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] "linp"
```

The positivity requirement ($x_i \geq 0$) is -by default- part of the linear programming problem, unless it is toggled off by setting argument `ispos` equal to `FALSE`.

```
> LP <-linp(E=E,F=F,G=G, H= H, Cost=Cost, ispos=FALSE)
> LP$X
```

```
[1] -3.00 -6.75  7.50  4.25
```

```
> LP$solutionNorm
```

```
[1] -7
```

5. solving sets of linear equations with sparse matrices

limSolve contains special-purpose solvers to efficiently solve linear systems of equations $Ax = B$ where the nonzero elements of matrix `A` are located near the diagonal.

They include:

- `Solve.tridiag`: the A matrix is tridiagonal. i.e. the non-zero elements are on the diagonal, below and above the diagonal.
- `Solve.banded` when the A matrix has nonzero elements in bands near the diagonal.
- `Solve.block` when the A matrix is block-diagonal.

In the code below, a tridiagnoal matrix is created first, and solved with the default R method (`solve`), followed by the banded matrix solver `Solve.band`, and the tridiagonal solver `Solve.tridiag`.

To make the problem matehamtically demanding, the A matrix contains 5000 rows and 5000 columns, and the problem is solved for 10 different vectors B.

We start by defining the non-zero bands on (`bb`), below (`aa`) and above (`cc`) the diagonal, required for input to the tridiagonal solver; we prepare the full matrix necessary for the default solver and the banded matrix `abd`, required for the banded solver.

```
> nn   <- 5000
```

Input for the tridiagonal system:

```
> aa <- runif(nn-1)
> bb <- runif(nn)
> cc <- runif(nn-1)
```

The full matrix has the nonzero elements on, above and below the diagonal

```
> A <-matrix(nrow=nn,ncol=nn,data=0)
> A [cbind((1:(nn-1)),(2:nn))]<-cc
> A [cbind((2:nn),(1:(nn-1)))]<-aa
> diag(A) <- bb
```

The banded matrix is more compact:

```
> abd <- rbind(c(0,cc),bb,c(aa,0))
```

Parts of the input are:

```
> A[1:5,1:5]

           [,1]       [,2]      [,3]       [,4]      [,5]
[1,] 0.08822781 0.64818526 0.0000000 0.00000000 0.0000000
[2,] 0.81288350 0.29341319 0.4808695 0.00000000 0.0000000
[3,] 0.00000000 0.06342866 0.4141302 0.32888011 0.0000000
[4,] 0.00000000 0.00000000 0.8400167 0.08246744 0.0875743
[5,] 0.00000000 0.00000000 0.0000000 0.87697160 0.6974295

> aa[1:5]

[1] 0.81288350 0.06342866 0.84001674 0.87697160 0.61965424

> bb[1:5]

[1] 0.08822781 0.29341319 0.41413018 0.08246744 0.69742952

> cc[1:5]

[1] 0.6481853 0.4808695 0.3288801 0.0875743 0.7060897

> abd[,5]

                 bb
0.0875743 0.6974295 0.6196542
```

The right hand side consists of 5 vectors:

```
> B <- runif(nn)
> B <- cbind(B,2*B,3*B,4*B,5*B,6*B,7*B,8*B,9*B,10*B)
```

The problem is then solved using the three different solvers. The duration of the computation is estimated and printed in *milliseconds*. `print(system.time()*1000)` does this. [2]

```
> print(system.time(Full <- solve(A,B))*1000)

   user  system elapsed
   1420     180    1610

> print(system.time(Band <- Solve.banded(abd,nup,ndwn,B))*1000)

   user  system elapsed
     20       0      20
```

[2] Note that the banded and tridiagonal solver are so efficient (on my computer) that these systems are solved quasi-instantaneously and the time returned = 0.

```
> print(system.time(tri  <- Solve.tridiag(aa,bb,cc,B))*1000)

   user  system elapsed 
      0       0       0 
```

The solvers return 5 solution vectors X, one for each right-hand side:

```
> Full[1:10,]

                B                                                                                 
 [1,]   0.1873297   0.3746593   0.561989   0.7493186   0.9366483   1.123978   1.311308
 [2,]   0.8154989   1.6309979   2.446497   3.2619958   4.0774947   4.892994   5.708493
 [3,]  -0.3755720  -0.7511441  -1.126716  -1.5022882  -1.8778602  -2.253432  -2.629004
 [4,]   3.0503702   6.1007403   9.151110  12.2014806  15.2518508  18.302221  21.352591
 [5,]  10.3273755  20.6547510  30.982126  41.3095019  51.6368774  61.964253  72.291628
 [6,] -13.8470430 -27.6940860 -41.541129 -55.3881720 -69.2352150 -83.082258 -96.929301
 [7,]   7.2271934  14.4543867  21.681580  28.9087735  36.1359668  43.363160  50.590354
 [8,]   9.8339509  19.6679019  29.501853  39.3358037  49.1697547  59.003706  68.837657
 [9,] -12.0266419 -24.0532839 -36.079926 -48.1065677 -60.1332097 -72.159852 -84.186494
[10,]  -4.9331373  -9.8662746 -14.799412 -19.7325492 -24.6656865 -29.598824 -34.531961

 [1,]    1.498637    1.685967    1.873297
 [2,]    6.523992    7.339490    8.154989
 [3,]   -3.004576   -3.380148   -3.755720
 [4,]   24.402961   27.453331   30.503702
 [5,]   82.619004   92.946379  103.273755
 [6,] -110.776344 -124.623387 -138.470430
 [7,]   57.817547   65.044740   72.271934
 [8,]   78.671607   88.505558   98.339509
 [9,]  -96.213135 -108.239777 -120.266419
[10,]  -39.465098  -44.398236  -49.331373
```

Comparison of the different solutions (here only for the second vector) show that they yield the same result.

```
> head(cbind(Full=Full[,2],Band=Band[,2], Tri=tri[,2]))

            Full        Band         Tri
[1,]   0.3746593   0.3746593   0.3746593
[2,]   1.6309979   1.6309979   1.6309979
[3,]  -0.7511441  -0.7511441  -0.7511441
[4,]   6.1007403   6.1007403   6.1007403
[5,]  20.6547510  20.6547510  20.6547510
[6,] -27.6940860 -27.6940860 -27.6940860
```

6. datasets

There are five example applications in package **limSolve** :

- `Blending`. In this underdetermined problem, an optimal composition of a feeding mix is sought such that production costs are minimised subject to minimal nutrient constraints. The problem consists of one equality and 4 inequality conditions, and a cost function. It is solved by linear programming (`linp`). Feasible ranges are estimated (`xranges`) and feasible solutions generated (`xsample`)

- `Chemtax`. This is an overdetermined linear inverse problem, where the algal composition of a (field) sample is estimated based on (experimentally-determined) pigment biomarkers (?). See also R -package **BCE** (?), (?). The problem contains 8 unknowns; it consists of 1 equality, 12 approximate equations, and 8 inequalities. It is solved using `lsei` and `xsample`.

- `Minkdiet`. This is another -underdetermined- compositional estimation problem, where the diet composition of Southeast Alaskan Mink is estimated, based on the C and N isotopic ratios of Mink and of its prey items (?). The problem consists of 7 unknowns, 3 equations, and 7 inequalities

- `RigaWeb`. This is a food web problem, where food web flows of the Gulf of Riga planktonic food web in Spring are quantified (?). This underdetermined model comprises 26 unknowns, 14 equalities, and 45 inequalities. It is solved by `lsei`, `xranges`, and `xsample`.

- `E_coli`. This is a flux balance problem, estimating the core metabolic fluxes of Escherichia coli (?). It is the largest example included in **limSolve** . There are 70 unknowns, 54 equalities and 62 inequalities, and one function to maximise. This model is solved using `lsei`, `linp`, `xranges`, and `xsample`.

7. Notes

Package **limSolve** provides FORTRAN implementations of:

- the least distance algorithms from (?) (`ldp`, `ldei`, `nnls`).
- the least squares with equality and inequality algorithms from (?) (`lsei`).
- a solver for banded linear systems from LAPACK (?).
- a solver for block diagonal linear systems from LAPACK (?).
- a solver for tridiagonal linear systems from LAPACK (?).

In addition, the package provides a wrapper around functions:

- function `lp` from package **lpSolve** (?)
- function `solve.QP` from package **quadprog** (?)

Table 1: Summary of the functions in package **limSolve**

Function	Description
Solve	Finds the generalised inverse solution of $A \cdot x = b$
Solve.banded	Solves a banded system of linear equations
Solve.tridiag	Solves a tridiagonal system of linear equations
Solve.block	Solves a block diagonal system of linear equations
ldei	Least distance programming with equality and inequality conditions
ldp	Least distance programming (only inequality conditions)
linp	Linear programming
lsei	Least squares with equality and inequality conditions
nnls	Nonnegative least squares
resolution	Row and column resolution of a matrix
xranges	Calculates ranges of unknowns
varranges	Calculates ranges of variables (linear combinations of unknonws)
xsample	Randomly samples a linear inverse problem for the unknowns
varsample	Randomly samples a linear inverse problem for the inverse variables

This way, similar input can be used to solve least distance, least squares and linear programming problems. Note that the packages **lpSolve** and **quadprog** provide more options than used here.

For solving linear programming problems, `lp` from **lpSolve** is the only algorithm included. It is also the most robust linear programming algorithm we are familiar with.

For quadratic programming, we added the code `lsei`, which in our experience often gives a valid solution whereas other functions (including `solve.QP`) may fail.

Finalisation of this package was done using R-Forge (`http://r-forge.r-project.org/`), the framework for R project developers, based on GForge and tortoiseSVN (`http://tortoisesvn.net/`) for (sub) version control.

This vignette was created using Sweave (**?**).

The package is on CRAN, the R-archive website ((**?**))

More examples can be found in the demo of package **limSolve** ("demo(limSolve)")

Another R -package, **LIM** (**?**) is designed for reading and solving linear inverse models (LIM). The model problem is formulated in text files in a way that is natural and comprehensible. **LIM** then converts this input into the required linear equality and inequality conditions, which can be solved by the functions in package **limSolve** .

A list of all functions in **limSolve** is in table (1).

Affiliation:

Karline Soetaert, Karel Van den Meersche, Dick van Oevelen
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