Package **LIM** , implementing linear inverse models in **R**

Karline Soetaert and Dick van oevelen
Royal Netherlands Institute of Sea Research
Yerseke
The Netherlands

Abstract

We present R package **LIM** (Soetaert and van Oevelen 2009) which 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 either by least squares or by linear programming techniques. By letting an algorithm formulate the mathematics, it becomes very simple to reformulate the model in case a parameter value changes, or a component is added or removed.

Three different types of problems are supported: flow networks, reaction networks and other (operations research) problems. The first two cases are based on mass balances of the components.

We give three examples, a food web example, a biogechemical reaction example and a blending example.

If you use this package, please cite as: (van Oevelen, van den Meersche, Meysman, Soetaert, Middelburg, and Vezina 2009).

Keywords: Linear inverse models, flux balance analysis, linear programming, text files, R.

1. Introduction

In many disciplines, mathematical formulation of problems lead to a combination of linear equalities that are supplemented with linear inequality constraints. Such linear equations arise for instance:

- by considerations that certain quantities have to be positive, that the summed values should not exceed a certain value (i.e. summed fractions or probabilities should remain smaller or equal to 1), etc.

- In curve fitting problems, inequality constraints may arise by requirements of monoticity, nonnegativity, convexity, while in piecewise linear fitting, equality conditions result from the need to guarantee continuity and smoothness of the curves.

- In biochemical applications, the linear equalities arise because of linear conservation relationships such as the conservation of mass, charge, etc.., while inequalities ensure that

mass remains a positive quantity.

2. Linear Inverse Models

Mathematically, linear inverse problems can be written in matrix notation as: [1]

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

These 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) andinequalities (3).

Often the problem originally only contains the latter two types of equations (2-3), and the approximate equalities are added to single out one solution.

Quadratic and linear programming methods are the main mathematical techniques to solve for the vector x in this type of models. In R, these are made available through package **limSolve** (Soetaert, Van den Meersche, and van Oevelen 2009).

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.

- If the model is even determined, there is only one solution that satisfies the equations exactly. This solution can be singled out by matrix inversion (e.g. the `solve` function, in case there are no inequalities) or using the least squares method *lsei* from package **limSolve** .

- If the model is overdetermined, there is only one solution in the least squares sense; this solution is singled out by function *lsei* (*l*east *s*quares with *e*qualities and *i*nequalities). This function also returns the parameter covariance matrix, which gives indication on the confidence interval and relationship among the estimated unknowns (elements in **x**).

- If the model is underdetermined, there exist an infinite amount of solutions. To solve such models, there are several options:

  - *ldei* - finds the "least distance" solution, i.e. the solution with minimal sum of squared unknowns.
  - *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 unknowns) is either minimized (a "cost" function) or maximized (a "profit" function)
  - *xranges* - finds the possible ranges ([min,max]) for each unknown.

---

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

– *xsample* - randomly samples the solution space using a Markov chain. This method returns the marginal probability density function for each unknown. (Van den Meersche, Soetaert, and Van Oevelen 2009)

All these functions are also available from package **LIM** .

3. Three types of LIM

One of the main remaining challenges in LIM models constitutes the setup of this type of problems. Especially when many unknowns have to be simultaneously estimated and the problem contains many equality and inequality constraints, the construction of the matrix equations may be quite complicated and error-prone. In addition to providing methods of solution, R-package **LIM** has been designed to facilitate problem implementation.

Depending on how the problem is formulated and which are the unknowns, **LIM** distinguishes three types of Linear Inverse Models (Figure 1).

- flow networks. Here the problem consists of a number of compartments, connected by flows. Solving the model then constitutes of deriving the values of the flows between the compartments.

- reaction networks. The problem consists of a number of compartments that are involved in reactions. The LIM will estimate the reaction rates.

- other. **LIM** can also solve problems often occurring in operational research, e.g. to find the optimal allocation of resources, optimal diet composition etc....

We give examples of these three types below.

3.1. Flow network problems

Flow networks are represented as a set of nodes (compartments), which are connected by arrows (flows). The arrows generally have a direction, i.e. the flows are positive. Thus

$$A \rightarrow B$$

denotes a flow directed from A to B, while

$$A \leftrightarrow B$$

denotes a flow that can proceed in both directions.

There can only be one flow from A to B (but there can also be a flow from B to A).

Solving the LIM-problem consists of finding the values of the flows.

After solution, several indices and food web properties can be estimated, using functions from package **NetIndices** (Soetaert and Kones 2008; Kones, Soetaert, van Oevelen, and Owino 2009)
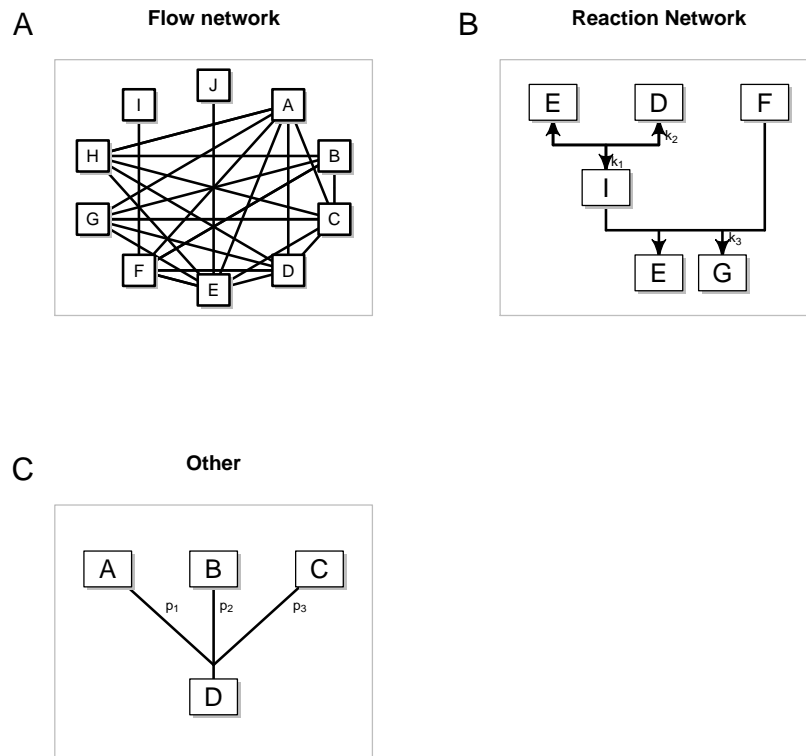
Figure 1: Three types of Linear Inverse Models that can be created and solved with R package **LIM** . A. Flow networks. B. Reaction networks, C. Other. In type (A) and (B), a mass balance of components is generated. This is not the case for type C.

Example: a simple food-web

Organisms eat and are eaten; they use part of their food for biomass production and reproduction, part is expelled as faeces or respired. Other (so-called autotrophic) organisms produce biomass from light energy and inorganic compounds, whilst dead matter (detritus) may be consumed by animals and bacteria.

When the mass balances of several groups of organisms (and dead matter) are considered together, we obtain a food web model. In this type of LIM, the unknowns are the food web *flows* that connect the components (organisms and dead matter).

Assume a simple food web comprising a plant, detritus and an animal that eats both the plant and detritus. For simplicity we assume that the system is in a climax situation, i.e. the masses, which are expressed in *moles C* m^{-2} are invariant in time. There are eight flows that connect the

components with each other and with the outside world. [2] The mass balance equation for the three components and with the rate of change = 0, is given by:

$$\frac{dPLANT}{dt} = 0 \;=\; \text{net primary production} - \text{grazing on plant} - \text{plant mortality}$$

$$\frac{dANIMAL}{dt} = 0 \;=\; \text{grazing on plant} + \text{grazing on detritus} - \text{animal respiration}$$
$$- \;\text{animal mortality} - \text{faeces production}$$

$$\frac{dDETRITUS}{dt} = 0 \;=\; \text{plant mortality} + \text{animal mortality} + \text{faeces production}$$
$$- \;\text{grazing on detritus} - \text{detritus mineralisation}$$

These mass balances can be written in a more general way, and using shorthand notation for the flows, as:

$$0 = 1 \cdot NPP - 1 \cdot Pgraz - 1 \cdot Pmort + 0 \cdot Dgraz + 0 \cdot Aresp + 0 \cdot Amort + 0 \cdot Faeces + 0 \cdot Detmin \quad (1)$$
$$0 = 0 \cdot NPP + 1 \cdot Pgraz + 0 \cdot Pmort + 1 \cdot Dgraz - 1 \cdot Aresp - 1 \cdot Amort - 1 \cdot Faeces + 0 \cdot Detmin \quad (2)$$
$$0 = 0 \cdot NPP + 0 \cdot Pgraz + 1 \cdot Pmort - 1 \cdot Dgraz + 0 \cdot Aresp + 1 \cdot Amort + 1 \cdot Faeces - 1 \cdot Detmin \quad (3)$$

These equations relate, on the left hand side, the zero rates of changes to a sum of products, where each product is composed of the flows and a coefficient. The coefficient indicates if and how much these flows contribute to the rate of change.

Now assume that net primary production and the total grazing rate (Grazing) of the animal has been measured (30 *mmol C m^{-2} d^{-1}* and 10 *mmol C m^{-2} d^{-1}* respectively). Thus, we can add two extra equations:

$$NPP = 30 \qquad\qquad (4)$$
$$Pgraz + Dgraz = 10 \qquad\qquad (5)$$

In matrix notation, we obtain

$$
\begin{bmatrix}
1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & -1 & -1 & -1 & 0 \\
0 & 0 & 1 & -1 & 0 & 1 & 1 & -1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0
\end{bmatrix}
\cdot
\begin{bmatrix}
NPP \\
Pgraz \\
Pmort \\
Dgraz \\
Aresp \\
Amort \\
Faeces \\
Detmin
\end{bmatrix}
=
\begin{bmatrix}
0 \\
0 \\
0 \\
30 \\
10
\end{bmatrix}
$$

---

[2]Since the foodweb is a subsystem of a larger system, we need to distinguish between model compartments, i.e. compartments whose dynamics are fully described in the model and external compartments, whose dynamics is coupled to processes occurring outside the model realm. The difference is essential: **LIM** will create mass balance equations for model compartments only. In the example, there is no balance for CO_2

The feeding, defaecation and respiration flows are not independent of one another. Firstly, organisms cannot produce more faeces than the amount of food they ingest. Thus it is customary in foodweb modelling, to assume that faeces production lies in between some range of food ingested. For our example we assume that in between 30 and 60% of total food ingested is defaecated (the food is not of high quality).

Secondly, organisms respire carbohydrates to provide the energy for growth. Thus, of the fraction of the food that is assimilated (i.e. not defaecated), part will be used to create new biomass, the other part will provide the energy to do so (this is referred to as the cost of growth). Here we assume that 30% of the assimilated food is respired. As total animal respiration also includes basal respiration (for the animal's maintenance), we impose that the animal respiration has to be larger than - or equal - to this amount:

$$0.3 \cdot Pgraz + 0.3 \cdot Dgraz <= Faeces \tag{6}$$

$$0.6 \cdot Pgraz + 0.6 \cdot Dgraz >= Faeces \tag{7}$$

$$0.3 \cdot (Pgraz + Dgraz - Faeces) <= Aresp \tag{8}$$

Adding to that the requirement that the flows have to be positive gives the following set of inequality conditions:

$$
\begin{bmatrix}
0 & -0.3 & 0 & -0.3 & 1 & 0 & 0.3 & 0 \\
0 & 0.6 & 0 & 0.6 & 0 & 0 & -1 & 0 \\
0 & -0.3 & 0 & -0.3 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\cdot
\begin{bmatrix}
NPP \\
Pgraz \\
Pmort \\
Dgraz \\
Aresp \\
Amort \\
Faeces \\
Detmin
\end{bmatrix}
>=
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}
$$

This model comprises 5 equations and 11 inequalities; there are 8 unknown flows. We will outline below how this particular problem can be implemented and solved in package **LIM** .

3.2. reaction problems

These are LIM problems which are written as a set of reactions that connect the dynamics of several constituents. For instance, in the reaction

$$A + B \rightarrow C$$
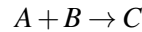
C is produced while A and B are consumed in a stoichiometric ratio of 1 to 1. Some reactions can occur in two directions, e.g.

$$A + B \leftrightarrow C$$
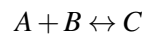
In contrast to previous ("flow") network problems, where only one link between two compartments was allowed, in reaction problems there may exist many links between the constituents. Solving the LIM amounts to finding values for the reaction rates.

The core metabolism of E.coli

The LIM software can be used for performing flux balance analysis (e.g. (Edwards, Covert, and Palsson 2002)).
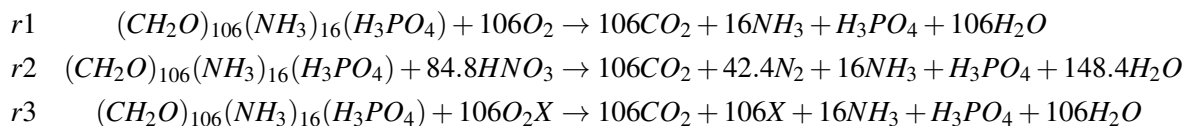
See vignette ("LIMecoli") (Soetaert 2009) for an example of how to do that.

Example: chemical reactions.

In the natural environment, the cycles of many constituents are linked via chemical reactions that produce and consume them. We take the biogeochemical cycling of carbon (C), nitrogen (N) and oxygen (O) in a marine sediment as an example.
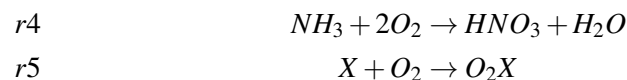
Organic matter $((CH_2O)_{106}(NH_3)_{16}(H_3PO_4))$ is mineralized (respired), using a series of oxidants: oxygen (O_2), nitrate (HNO_3) and some other, undefined oxidant (XO). The reduced byproducts of this mineralization process, ammonium (NH_3), and an undefined reduced substance (X) can be re-oxidized by a reaction with oxygen. All dissolved substances are exchanged with the water column. N_2, produced by the reaction of organic matter with nitrate, does not react in the sediment.

The mineralisation reactions can be written as:

$r1 \qquad (CH_2O)_{106}(NH_3)_{16}(H_3PO_4) + 106O_2 \rightarrow 106CO_2 + 16NH_3 + H_3PO_4 + 106H_2O$

$r2 \quad (CH_2O)_{106}(NH_3)_{16}(H_3PO_4) + 84.8HNO_3 \rightarrow 106CO_2 + 42.4N_2 + 16NH_3 + H_3PO_4 + 148.4H_2O$

$r3 \qquad (CH_2O)_{106}(NH_3)_{16}(H_3PO_4) + 106O_2X \rightarrow 106CO_2 + 106X + 16NH_3 + H_3PO_4 + 106H_2O$
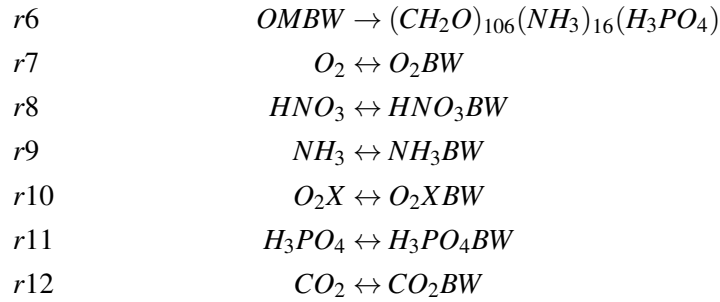
for the oxic mineralisation, denitrification and anoxic mineralisation respectively.

The secondary reactions (nitrification and reoxidation of other reduced substances):

$$r4 \qquad\qquad NH_3 + 2O_2 \rightarrow HNO_3 + H_2O$$
$$r5 \qquad\qquad X + O_2 \rightarrow O_2X$$

and the exchange with the bottom water:

$$
\begin{array}{ll}
r6 & OMBW \rightarrow (CH_2O)_{106}(NH_3)_{16}(H_3PO_4) \\
r7 & O_2 \leftrightarrow O_2BW \\
r8 & HNO_3 \leftrightarrow HNO_3BW \\
r9 & NH_3 \leftrightarrow NH_3BW \\
r10 & O_2X \leftrightarrow O_2XBW \\
r11 & H_3PO_4 \leftrightarrow H_3PO_4BW \\
r12 & CO_2 \leftrightarrow CO_2BW
\end{array}
$$

Note that the deposition of organic matter (r6) is directed *into* the sediment, while the direction of the other fluxes can go either into or out of the sediment.

In this LIM, the rates of the mineralisation reactions, of the secondary reactions and the exchange reactions with the bottom water are the unknowns (r1-r12). Based on these reactions, and following the law of conservation of mass, we can write a mass balance reaction for the following 7 constituents: $(CH_2O)_{106}(NH_3)_{16}(H_3PO_4)$, O_2, CO_2, NH_3, H_3PO_4, HNO_3, while for the others (e.g. O_2BW) only part of the reactions are specified. Hence these are considered to be outside the domain of the model (externals).

We give only the mass balance reactions for O_2 and HNO_3:

$$
\begin{aligned}
\frac{dO_2}{dt} &= 0 = -106 \cdot r_1 - 2 \cdot r_4 - r_5 - r_7 \\
\frac{dHNO_3}{dt} &= 0 = -84.8 \cdot r_1 + \cdot r_4 - r_8 \\
&\quad \dots
\end{aligned}
$$

As the exchange of dissolved substance across the sediment-water interface can go either way, directed into or out of the sediment, they can be either positive or negative. Only the rates of unidirectional reactions need be positive, and the following inequalities hold:

$$
\begin{aligned}
r_1 &>= 0 \\
r_2 &>= 0 \\
r_3 &>= 0 \\
r_4 &>= 0 \\
r_5 &>= 0 \\
r_6 &>= 0
\end{aligned}
$$

In this particular example, the oxygen, nitrate, and ammonium fluxes have been estimated; they are -15 (influx), 1 (efflux) and 2 *mmol* m^{-2} d^{-1} respectively. These measurements lead to the equations:

$$r7 = -15$$
$$r8 = 1$$
$$r9 = 2$$

Thus there are 10 equations (7 mass balances, 3 measurements) and 12 unknowns. In addition, there are 6 inequality conditions[3].

3.3. other problems

It is also possible to use **LIM** for specifying more general (linear) operational research problems that do not classify as network problems.

These problems often try to find the most efficient, or least costly, way of achieving something. They are often solved with linear programming techniques that optimize some function (cost or profit) given a set of linear constraints.

blending problems

This example is borrowed from **limSolve** and comes from the website of J E Beasley (find it on the web).

A manufacturer produces a feeding mix for animals. The feed mix contains two nutritive ingredients and one ingredient (filler) to provide bulk. One kg of feed mix must contain a minimum quantity of each of four nutrients as below:

Nutrient	A	B	C	D
gram	80	50	25	5

The ingredients have the following nutrient values and cost:

(gram/kg)	A	B	C	D	Cost/kg
Ingredient 1	100	50	40	10	40
Ingredient 2	200	150	10	-	60
Filler	-	-	-	-	0

The problem is to find the composition of the feeding mix that minimises the production costs subject to the constraints above. Stated otherwise: what is the optimal amount of ingredients in one kg of feeding mix?

---

[3]Note a difference with the flow networks, where the coefficients were either -1, 0, 1. Here the coefficients reflect the stoichiometry of the reaction and can differ from these numbers

Mathematically this can be estimated by solving a linear programming problem, where the equalities ensure that the sum of the three fractions equals 1, and the inequalities enforce the nutritional constraints; the quantity to be minimized is the cost function.

$$\min(x_1 \cdot 40 + x_2 \cdot 60)$$
$$x_i \geq 0$$
$$x_1 + x_2 + x_3 = 1$$

and

$$100 \cdot x_1 + 200 \cdot x_2 \geq 80$$
$$50 \cdot x_1 + 150 \cdot x_2 \geq 50$$
$$40 \cdot x_1 + 10 \cdot x_2 \geq 25$$
$$10 \cdot x_1 \geq 5$$

4. Specifying a Linear Inverse Model in R-package LIM

The previous examples were quite simple, and the resulting matrices of small or moderate size. Nevertheless, it is easy to make mistakes. Moreover, once the matrices are constructed, it may be quite a challenge to update them after adding or removing constituents. Also, based on the resulting set of linear equations it is not straightforward to infer the underlying model assumptions.

In general, a linear inverse model is first formulated verbally, after which the verbal description of the problem is translated into an equivalent mathematical description.

Typically the equations are specified on aggregated unknowns, i.e. unknowns that are themselves linear combinations of other unknowns. For instance, in the food web model example, the faeces production (the flow from the animal to detritus) is specified as a part of the amount of food ingested. Ingested food is itself the sum of the flow from the plant to the animal and from detritus to the animal.

Model input in **LIM** is close to these verbal statements. Thus to implement the food web model we first define a variable called "Ingestion" that consists of the sum of the two feeding flows and then define the defaecation constraints on this variable. When the LIM input is parsed, the constraints will be rewritten as a function of the unknowns.

Apart from this more natural input, there are many other benefits of using **LIM** . For instance, for the *flow network* and *reaction network* type of problems, **LIM** generates the mass balances for each component, based on the flows or reactions that were defined. This facilitates adding or removing flows or constituents. Finally, solving the model will also generate estimates of all defined variables.

We now document the input for each of the above introduced problems.

4.1. food web problem

```
=======================================================
Header of the file - ignored
file: foodweb.lim

Solve the model in R with:
require(LIM)
lim <- Setup("foodweb.lim")
Ldei(lim)
Xranges(lim)
=======================================================

## EXTERNAL
  CO2
  EXP   ! export
## END EXTERNAL


## COMPONENT
  Pl    ! plant
  AN    ! Animal
  Det   ! Detritus
## END COMPONENT


## Flows
  NPP   : CO2 -> Pl
  Pgraz : Pl -> An
  Pmort : Pl -> Det
  Dgraz : Det -> An
  Aresp : An ->CO2
  Amort : An ->EXP
  Faeces: An ->Det
  Detmin: Det -> CO2
## END Flows



## PARAMETERS
  minFaeces  = 0.3
  maxFaeces  = 0.6
  growthCost = 0.3
## END PARAMETERS


## VARIABLES
  Ingestion      = Pgraz + Dgraz
  Assimilation   = Ingestion - Faeces
```

```
  GrowthResp     = Assimilation*growthCost
## END VARIABLES

## Equalities
  Faeces = 30
  Det -> CO2 = 10
## End equalities

## Inequalities
  growthcost : Aresp > GrowthResp
  defaecation: Faeces = [minFaeces,maxFaeces]*Ingestion
## End inequalities
```

Note the use of sections (## SECTIONNAME ... ## END SECTIONNNAME) to declare items; the sections "COMPONENT" and "EXTERNAL" define the names; a mass balance equation is only generated for components, not for externals. A name is declared as "name: ", an exclamation mark ("!") demarcates the start of a comment.

Although more lengthy, this problem formulation is much more elegant, more flexible, less error-prone, and easier to understand than the resulting matrices themselves.

Based on this input file, the matrices are generated using **LIM** function *Setup* and put in a list (see below). The resulting LIM input can then be solved with *Lsei(lim, parsimonious=TRUE)* or with *Ldei*, which will generate the simplest -parsimonious- solution, with *Xranges* which will estimate ranges of unknowns, or with *Xsample* which will generate the conditional probability distribution of each flow.

In the table below is what we obtained from running the following R-code:

```
require(LIM)
web.lim <- Setup("foodweb.lim")
pars <- Ldei(web.lim)
webranges<- Xranges(web.lim)

data.frame(webranges,parsimonious=pars$X)
```

	min	max	parsimonious
NPP	30	80	30.000000
Pgraz	0	80	29.554950
Pmort	0	80	0.445050
Dgraz	20	100	20.445050
Aresp	6	70	9.489658
Amort	0	49	10.510342
Faeces	30	30	30.000000
Detmin	10	10	10.000000

Based on these results it is simple to create a plot which depicts the parsimonious solution and the ranges (see Figure 2):

```
xlim <- range(webranges)
dotchart(x=pars$X,labels=rownames(webranges),xlim=xlim,
         main="Food web",pch=16)
cc <- 1:nrow(webranges)
segments(x0=webranges[,1],y0=cc,x1=webranges[,2],y1=cc)
```
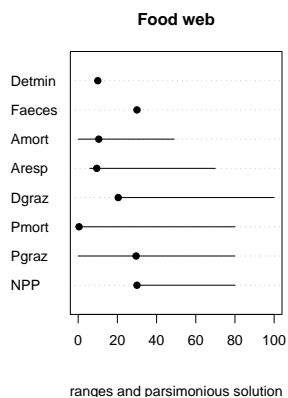

Figure 2: Ranges and parsimonious solution of foodweb example - seee text for R-code

4.2. chemical reaction problem

The input of the chemical reaction problem is:

```
=====================================================
Header of the file - ignored
file reaction.lim
0-dimensional sediment coupled C, N, O, P model
run with:
require(LIM)
reaction.lim <- Setup("reaction.lim")
X  <- Ldei(reaction.lim)
xr <- Xranges(reaction.lim, ispos = FALSE)
=====================================================
### COMPONENTS
OM
O2
CO2
NH3
```

```
H3PO4
HNO3
X
### END COMPONENTS

### EXTERNAL
H2O
N2
O2X
OMBW
O2BW
HNO3BW
NH3BW
XBW
H3PO4BW
CO2BW
### END EXTERNAL

### REACTIONS
r1: OM + 106*O2 -> 106*CO2 + 16*NH3 + H3PO4 + 106*H2O
r2: OM + 84.8*HNO3 -> 106*CO2 + 42*N2+16*NH3 + H3PO4 + 148.4*H2O
r3: OM + 106*O2X -> 106*CO2 + 106*X + 16*NH3 + H3PO4 + 106*H2O

r4: NH3+2*O2 ->HNO3 +H2O
r5: X+ O2 ->O2X

r6: OMBW -> OM
r7: O2 <-> O2BW
r8: HNO3 <-> HNO3BW
r9: NH3 <-> NH3BW
r10: X <-> XBW
r11: H3PO4 <-> H3PO4BW
r12: CO2 <-> CO2BW
### END REACTIONS

### EQUATIONS
r7 = -15
r8 = 1
r9 = 2
### END EQUATIONS

### INEQUALITY
 r1>0
```

```
 r2>0
 r3>0
 r4>0
 r5>0
 r6>0
### END INEQUALITY
```

Results are in the following table:

	min	max	parsimonious
r1	0.00000000	0.12264151	0.1226364
r2	0.00000000	0.07665094	0.00000000
r3	0.06485849	0.51709906	0.06486363
r4	1.00000000	7.50000000	1.00000000
r5	0.00000000	13.00000000	0.00005450
r6	0.18750000	0.59375000	0.18750000
r7	-15.00000000	-15.00000000	-15.00000000
r8	1.00000000	1.00000000	1.00000000
r9	2.00000000	2.00000000	2.00000000
r10	6.87500000	54.81250000	6.87500000
r11	0.18750000	0.59375000	0.18750000
r12	19.87500000	62.93750000	19.87500000

The marginal probability distribution of all reaction rates can be generated by `Xsample` and then simply plotted using R-function `pairs`. This is done in the R-script below. Before creating the pairs plot, we first remove the rates that were given a fixed value. On the diagonal of the pairs plot, we plot a histogram; we define this function first (it is copied from one of the examples in the pairs help file). We plot only the lower part of the pairs plot (i.e. set `upper.panel = NULL`).

```
xs <- Xsample(reaction.lim, jmp = 10, iter = 500)
panel.hist <- function(x, ...)  {
    usr <- par("usr")
    on.exit(par(usr))
    par(usr = c(usr[1:2], 0, 1.5) )   #redefine y-axis; x-axis stays the same
    h <- hist(x, plot = FALSE)
    breaks <- h$breaks; nB <- length(breaks)
    y <- h$counts
    y <- y/max(y)
    rect(breaks[-nB], 0, breaks[-1], y, col = "grey", ...)
}
xs <- xs[ ,-(7:9)]                       #remove constant rates
```

```
pairs(xs, upper.panel = NULL, diag.panel = panel.hist,
      pch = ".", cex = 2, main = "Reaction network")
```
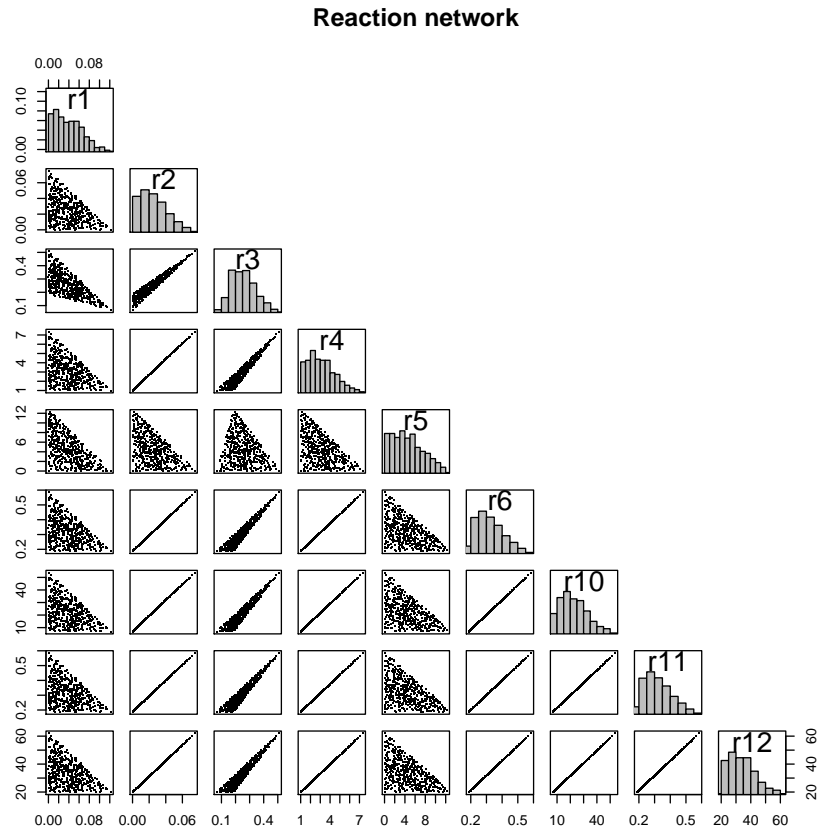
The results are in Figure 3

Figure 3: MCMC sample of the reaction network

4.3. blending problems

Finally we give the input for the blending problem.

```
======================================================
Header of the file - ignored
Blending problem file blending.lim
run with:
require(LIM)
blend.lim <- Setup("blending.lim")
lp <- Linp(blend.lim)
```

```
xr <- Xranges(blend.lim, ispos = TRUE)
xs <- Xsample(blend.lim)
=======================================================

### COMPONENT
 X1   ! Part ingredient 1
 X2   ! Part ingredient 2
 X3   ! Part ingredient 3 = filler
### END COMPONENTS

## PARAMETERS
! Minimal nutrient requirements in feeding mix
 NutA = 80
 NutB = 50
 NutC = 25
 NutD = 5

! Cost of ingredients 1,2,3
 Cost1 = 40
 Cost2 = 60
 Cost3 =  0

! nutrient contents in X1 and X2
 N1_A =100
 N1_B =50
 N1_C =40
 N1_D =10

 N2_A =200
 N2_B =150
 N2_C =10
 N2_D =0

 N3_A =0
 N3_B =0
 N3_C =0
 N3_D =0
## END PARAMETERS

## COST
 Cost1*X1 + Cost2*X2 + Cost3*X3
## END COST
```

```
### EQUATIONS
 X1 + X2 + X3 = 1
### END EQUATIONS

### INEQUALITY
 X1>0
 X2>0
 X3>0

 N1_A*X1 + N2_A*X2 + N3_A*X3 >NutA
 N1_B*X1 + N2_B*X2 + N3_B*X3 >NutB
 N1_C*X1 + N2_C*X2 + N3_C*X3 >NutC
 N1_D*X1 + N2_D*X2 + N3_D*X3 >NutD

### END INEQUALITY
```

The following code generates multiple solutions (small dots) and plots these together with the minimal cost solution (large red dots) (see figure 4). Note that the MCMC-generated matrix is extended first with the parsimonious results.

```
XS <- rbind(lp$X,xs)

xsplot <- function (x,y,...) {
points(x,y,pch=".")
points(x[1],y[1],pch=16,cex=2,col="red")
}

pairs(XS,upper.panel=NULL,lower.panel=xsplot,main="blending")
```
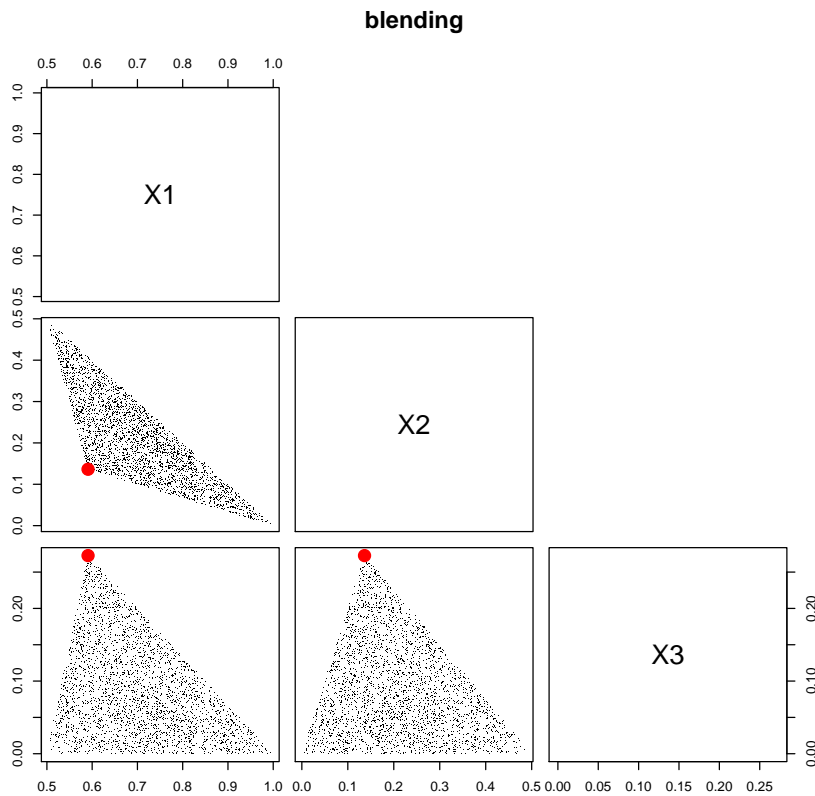
Figure 4: MCMC sample of the blending problem

4.4. A simple linear programming problem

It is also possible to write simple linear programming problems as in the following example, from Vanderbei (2006), example 2.11:

```
========================================================
File linprog.lim
Simple linear programming example
========================================================


## EQUALITIES
-x12             + x23 + x24         = 0
    - x13        - x23         + x34 = 0
## END EQUALITIES


## INEQUALITIES
```

```
 x12 + x13 + x14                        > 1
             x14        + x24 + x34 < 1
## END INEQUALITIES


## MINIMISATION
x12 + 8*x13 + 9*x14 + 2*x23 + 7*x24 + 3*x34
## END MINIMISATION
```

In this simple example, the components ("decision variables" in linear programming jargon) are not explicitly declared. Their names are inferred from the minimization function.

This model is solved as follows:

```
Linp("linprog.lim")
```

which gives:

```
$residualNorm
[1] 1.110223e-16

$solutionNorm
[1] 6

$X
      X12 X13 X14 X23 X24 X34
cost1   1   0   0   1   0   1
```

5. Structure of the LIM input file

Based on the examples above, we now discuss the general structure of the LIM input files.

The structure of the LIM declaration file has to obey the following **rules**:

- Declarations are case-INsensitive: flows, Flows, FLOWS is all the same.

- The declaration file is divided into several sections, each contained between '## section name' and '## END section name'. Only the text embraced by "##" and "## END" couples is considered by the **LIM** parser. The number of #s does not matter. Only the first four characters of the section names are considered, e.g. to designate the parameter section, we can write ## PARAM or ##PARAMETERS. The declaration sections allowed are summarised in table 1.

- text inbetween the declaration sections is ignored (and can be used to write comments). In the foodweb example for instance, all text positioned in front of "## EXTERNAL" will be ignored.

- An input file can contain declarations for externals, components, flows, parameters, variables and also defines the additional equalities (i.e. not the mass balances) and inequalities, costs, and profits (see below).

- Any line that starts with "!" or any blank line is ignored. The exclamation mark can also be used to discard part of an input line (i.e. everything past "!" is ignored).

- Simple calculations are allowed, i.e. addition and multiplication. The use of brackets for a calculation is not allowed.

- Continuation of a line is allowed via the use of the "&" sign, at the end of the line.

- flows can also be given a name using ("name :"). Although this is not mandatory, it may make the equations more readable.

- equalities and inequalities can also be given a name. This is only used for output.

A number of **shorthand** notations are available:

- If the LIM is a flow network, then *FLOWfrom(x)* is shorthand for the sum of all flows directed out of component x, while *FLOWto(x)* is shorthand for all flows directed into component x. In the foodweb model example, we wrote:

```
## VARIABLES
   Ingestion      = Pgraz + Dgraz
## VARIABLES
```

This could have been written as:

```
## VARIABLES
   Ingestion      = Flowto(An)
## VARIABLES
```

- In the inequality section, using

```
[]
```

assigns in one statement lower and upper bounds. In the foodweb example for instance, we wrote:

```
   defaecation: Faeces = [minFaeces,maxFaeces]*Ingestion
```

```
   defaecation1: Faeces > minFaeces*Ingestion
   defaecation2: Faeces < maxFaeces*Ingestion
```

Table 1: Nomenclature for LIM elements; sometimes several names are allowed for one type of element; the parser only considers the first four characters.

Name	Description
COMP, STATE, STOCKS, DECI, UNKN	compartments (or components) between which flows are defined. If neither *FLOWS* nor *REACTIONS* are defined, then the compartments constitute the unknowns to be estimated (so-called "decision variables", see the blending example). If *FLOWS* or *REACTIONS* are present, then there will be one mass balance generated for each compartment; this distinguishes them from *EXTERNALS*. If the components are not explicitly specified, they will be generated from the *FLOWS*, *REACTIONS* or *COST* or *PROFIT* section. To avoid errors due to typing mistakes, it is recommended to explicitly define components; in this case the parser can check if all items used in the flow or reaction section actually exist. Compartments may be given a value.
EXTERNALS	compartments that represent the external world. There is no mass balance generated for *EXTERNALS*
PARAMETERS	Parameters (with their values) that have constant values during one model application but whose value can be altered for other applications. They can be changed in monte carlo runs (see below). Parameters may be calculated based on other parameters (that have been declared in front of the derived parameter).
FLOWS	Flows between two components, written either as Flow(Source,Sink) or Source $\rightarrow$ Sink, and where source and sink are components. If this section is defined, then the *FLOWS* will be considered the unknowns that have to be estimated. This declaration section forms the basis of a set of mass balance equations, one for each component. Cannot co-occur together with *REACTIONS* declarations
REACTION	A reaction occurring between components, e.g. A $\rightarrow$ B + 2*C. Reactions that can occur in two directions are denoted as in A $\leftrightarrow$ B+C. For each unidirectional reaction, the rates are positive; thus an inequality condition (rate>0) is imposed. Reactions occurring in two directions need not be positive. If this section is defined, then the *reaction rates* will be considered the unknowns that have to be estimated. The reactions form the basis of a set of mass balance equations, one for each component. A *REACTION* section can not co-occur with a *FLOWS* declaration section.
VARIABLES	A linear expression involving the unknowns, parameters or other variables. Variables are derived quantities, useful to make the declaration of e.g. inequality constraints easier and more readable. Their values are estimated by function `Variables`; the ranges of variables can be solved with function `Varranges`
COST, MINIMUM	One or more linear expression(s) that should be minimized
PROFIT, MAXIMUM,	One or more linear expression(s) that should be maximized
RATES	Only present when the problem is a flow or reaction network and the net rate of change is not 0. It should give the net rate of change for a compartment. If not specified, steady-state is assumed and *RATES* gets the value 0.
EQUALITIES	Relationships between unknowns, or measured values that are assumed to be exactly known
INEQUALITIES	Relationships between unknowns, or measured values that are assumed to be known only with certain bounds.

6. setting up the linear inverse model

The LIM input is used to create the matrices and vectors that constitute the lsei problem (least squares with equalities and inequalities).

This is done in two steps.

- Function *Read* performs the first step, which creates the *liminput*, a list that defines all elements of the LIM as a function of the other elements.

- Function *Setup.lim* or *Setup* performs the second step. Based on the *liminput*, all terms are written as a function of the unknowns only and the matrices and vectors **A**, **b**, **G** and **h** are created.

It is also possible to create the LIM matrices and vectors directly from an input file. This is done by calling function *Setup* which takes as input a file name.

Thus:

```
lim <- Setup("linprog.lim")
```

does the same as:

```
liminput <- Read("linprog.lim")
lim <- Setup(liminput)
```

Splitting problem generation in two steps is convenient when several runs need to be performed with different parameter values, e.g. for performing a monte carlo analysis. Thus, the values of the parameters can be directly altered in the *liminput* list, after which *Setup* will recreate the corresponding matrices and vectors.

In the next sections we take a closer look at how setting up the LIM is achieved. This rather technical information can be skipped.

6.1. creating a liminput list

Here the names of all section elements, and their calculations are saved as a list of type "liminput". All elements are considered to result from linear calculations, which are saved as a data frame. In this data frame, one line denotes a product, while subsequent lines belonging to the same calculation are sums. A product can be composed of the following items (columns): constant values (column "val"), parameters (up to 4, columns "par1",..."par4"), variables ("var"), flows ("flow"), components ("comp"), externals and reactions.

Except for the constants, all items are denoted with their number.

Consider the following part of the liminput generated by reading the simple linear programming example.

The data frame captures the calculation of the inequalities. They were defined as:

```
## INEQUALITIES
 x12 + x13 + x14                      > 1
              x14       + x24 + x34 < 1
## END INEQUALITIES
```

and are parsed into the following data.frame:

```
$constraints
    name nr val par1 par2 par3 par4 var flow  comp external reaction
1 ineq1  1   1  NA   NA   NA   NA  NA   NA    1    NA       NA
2 ineq1  1   1  NA   NA   NA   NA  NA   NA    2    NA       NA
3 ineq1  1   1  NA   NA   NA   NA  NA   NA    3    NA       NA
4 ineq1  1  -1  NA   NA   NA   NA  NA   NA   NA    NA       NA
5 ineq2  2  -1  NA   NA   NA   NA  NA   NA    3    NA       NA
6 ineq2  2  -1  NA   NA   NA   NA  NA   NA    5    NA       NA
7 ineq2  2  -1  NA   NA   NA   NA  NA   NA    6    NA       NA
8 ineq2  2   1  NA   NA   NA   NA  NA   NA   NA    NA       NA
```

which should be understood as follows:

First of all, there are two inequalities, numbered 1 and 2 (column "nr"); both are the sum of 4 terms (there are 4 lines in the data frame for each ineqauality). On line 2, only column "val" and "comp" are not a NA. This term should be read as 1*comp[2].

Inequality 1 can thus be reconstructed as: $1 * comp[1] + 1 * comp[2] + 1 * comp[3] - 1 > 0$

The liminput data.frame contains the following elements:

- "file" - the name of the input file

- "pars" - a data.frame with the parameters

- "comp" - a data.frame with the components (state variables)

- "rate" - a data.frame with rates of change

- "extern" - a data.frame with externals

- "flows" - a data.frame with flows (these have a different -simplified structure)

- "vars" - a data.frame with variables

- "cost" - a data.frame with cost function

- "profit" - a data.frame with profit function

- "equations" - a data.frame with equality conditions

- "constraints" - a data.frame with inequality conditions

- "reactions" - a data.frame with reaction

- "posreac" - a vector of logical elements: TRUE if corresponding reaction is positive (i.e. unidirectional reaction, →) , FALSE otherwise (i.e. two-ways reaction, ↔)

- "marker" - a data.frame with markers (see below)

- "parnames" - a vector with parameter names

- "varnames" - a vector with variable names

- "compnames" - a vector with component (state variable) names

- "externnames" - a vector with names of externals

- "Type" - one of "flow", "reaction",or "simple"

6.2. creating LIM matrices and vectors

Based on the *liminput* list, function *Setup* rewrites all terms as a function of the unknowns only. It creates an instance of class *lim*, a list that contains, amongst other things the matrices and vectors **A**, **b**, **G** and **h**.

The following elements are in type *lim*

- "file" - The name of the input file

- "NUnknowns" - the number of unknowns

- "NEquations" - the number of equations inputted (the "true" equality conditions, i.e. excluding the mass balances for flow and reaction networks)

- "NConstraints" - the number of constraints inputted (the "true" inequality conditions, i.e. excluding the positivity constraints which are assumed for flow networks)

- "NComponents" - the number of components or state variables

- "NExternal" - the number of externals

- "NVariables" - the number of variables

- "A" - the matrix **A**, containing the coefficients of the equalities. If the problem is a flow or reaction network, then the first *NComponent* equations are the mass balances, the last *NEquations* rows correspond to the inputted equalities.

- "B" - the vector **b**, containing the right hand side of the equalities

- "G" - the matrix **G**, containing the coefficients of the inequalities. If the problem is a flow or reaction network, then the first *NConstraints* rows correspond to the inputted inequalities, while the last rows correspond to the imposed positivity constraints. For flow networks, there are *NComponents* positivity constraints ; for reaction networks the number of positivity constraints are less than or equal to this amount.

- "H" - the vector **h**, containing the right hand side of the inequalities

- "Cost" - the cost vector, contains the coefficients of the cost function

- "Profit" - the profit vector, contains the coefficients of the profit function

- "Flowmatrix" - if a flow network: a matrix whose i,jth value denotes the flow number from i to j

- "VarA" - variable matrix; contains the coefficients of the variables. [4]

- "VarB" - variable vector; contains the right hand side of the variable declarations

- "Parameters" - names and values of all parameters

- "Components" - names and values of all components

- "Externals" - names and values of all externals

- "rates" - names and values of all rates

- "markers" - names and values of all markers (see below)

- "Variables" - names of the variables

- "eqnames" - names of the equalities

- "ineqnames" - names of the inequalities

- "costnames" - names of the cost functions

- "profitnames" - names of the profit functions

- "Unknowns" - names of the unknowns

- "ispos" - true if the unknowns have to be positive, i.e. for certain flows in a flow network

6.3. Solving LIM problems

During an inverse solution, two norms are calculated:

- the residual to the equations E * X = F. This is called the residual norm. A model that can be solved has a residual norm ~ 0.

---

[4]Variables are linear functions of the unknowns defined as: VarX*x-VarB

- the value of the function that has been minimised or maximised: MIN(f(Flows)) or MAX (f(Flows). This is the solution norm.

7. Examples

Package **LIM** contains many example input files. They are present in three different subdirectories of the packages `examples` subdirectory.

- directory `Foodweb` contains the following food web examples:

    - `BrouageMudflat.input` (Leguerrier, Niquil, Boileau, Rzeznik, Sauriau, Le Moine, and Bacher 2003)
    - `CaliforniaSediment.input` (Eldridge and Jackson 1993)
    - `Everglades.input` (Diffendorfer, Richards, Dalrymple, and DeAngelis 2001)
    - `foodweb.lim`, the simple food web in this vignette
    - `RigaAutumn.input,RigaSummer.input`, `RigaSpring.input` (Donali, Olli, Heiskanen, and Andersen 1999)
    - `ScheldtIntertidal.input` (Van Oevelen, Soetaert, Middelburg, Herman, Moodley, Hamels, Moens, and Heip 2006)
    - `Takapoto.input` (Niquil, Jackson, Legendre, and Delesalle 1998)

- directory `Reactions` contains

    - `E_coli.lim` (Edwards *et al.* 2002)
    - `fba_simple.lim`
    - `reaction.lim` the example from this vignette

- directory `LinearProg` contains

    - `alloymixture.input`
    - `blending.input`
    - `diet.input`
    - `FoodManufacture.001`
    - `Greenberg1.input`
    - `linprog.lim` the example from this vignette
    - `Machines.input`
    - `manpower.001`
    - `refinery.001`
    - `transportation.input`

– `Vanderbei1.input`, `Vanderbei2.input`, `Vanderbei3.input`, `Vanderbei4.input`, `Vanderbei5.input` 5 examples from the book of (Vanderbei 2000)

Additional foodweb examples can be found in package **ToxLim** (de Laender, van Oevelen, Middelburg, and Soetaert 2009)

8. Extensions

8.1. Markers

For some flow network applications it may be useful to solve not only the default mass balance that is setup based on the section "## COMP", but also an additional mass balances based e.g. on stable isotope data. Stable isotope data are collected in many food web studies to decipher food sources or trophic position of a consumer. Mathematically, isotope data are evaluated with so-called linear mixing models. Verbally, a linear mixing model assumes that the isotope value of a consumer is a flow-weighted average of its resources. Mathematically this is represented by:

$$X_j = \frac{\sum_i F_{i \to j} (X_i + \Delta_j)}{\sum_i F_{i \to j}} \tag{9}$$

in which X_j is the isotope composition of the consumer, $F_{i \to j}$ is the flow from resource i to consumer j and Δ_j is trophic fractionation (e.g. ^{15}N fractionates with $\sim$3 %■per trophic level).

Stable isotope data can be inputted as data equations in a **LIM** . Suppose that in the food web example we have δ^{15}N isotope data for plants (Pl_iso), detritus (Det_iso) and the consumer AN (AN_iso). Further, we assume a common fractionation factor of 3.4%■for δ^{15} (Frac_iso). The resulting isotope equation can be included in the ## EQUALITIES section as:

```
## EQUALITIES
 ...
 AN_iso*Ingestion = Pl_iso*Pl->AN + Frac_iso*Pl->AN +
Det_iso*Det->AN + Frac_iso*Det->AN
 ...
## END EQUALITIES
```

This notation works fine and is easily manageable for this simple example. When however, food webs interactions are more complex, the **LIM** package offers a short-hand notation for the above. We can rewrite equation 9 as

$$\sum_i F_{i \to j} (X_i - X_j) = -\Delta_j \sum_i F_{i \to j} \tag{10}$$

in which the left-hand side of the equation is the mass balance of the isotope and the right-hand side accounts for trophic fractionation processes.

In the file we first include the parameter value for Frac_iso in the ## PARAMETER section and define an additional section "## MARKER" in which the marker values (in this case δ^{15}N data) are assigned to the components of the food web.

```
## PARAMETERS
 ...
Frac_iso = 3.4
 ...
## END PARAMETERS


## MARKER
 ...
 Pl  =   10
 Det =   5
 AN  =   11
 ...
## END MARKER
```

Subsequently, we define the additional mass balance for the isotope in the equalities.

```
## EQUALITIES
 ...
 Massbalance(AN) = -flowto(AN)*Frac_iso
 ...
## END EQUALITIES
```

In the above formulation, the left-hand side of the equality gives the mass balance notation as derived above and the right-hand side takes care of the trophic fractionation. This short-hand notation has several advantages. First of all, the notation in the equalities becomes must shorter (especially in large flow networks). Secondly, when one decides to remove a particular food source from a consumer in the ## FLOWS section, this change in the food web structure is also propagated to the isotope balance, reducing the change of inconsistencies between different mass balances. Currently, it is only possible to define one additional marker in the **LIM** package using the short-hand notation.

8.2. Monte Carlo runs

Sometimes, a certain model needs to be solved with different parameter values.

Consider the following flow network:

```
==========================================================================
file simple.input
```

```
=============================================================================
##Parameter
 f1 = 1
## end parameter

## FLOWS
A                -> B
A                -> C
B                -> C
C                -> A
B                -> D
D                -> A

## END FLOWS

## EQUATIONS
0.5* A->B = A->C

C->A = D->A
C->A = f1
## END EQUATIONS
```

where there are four compartments, and 6 flows. The value of the flow from C to A is specified with a parameter (f1).

To solve this model, we write:

```
Ldei("simple.input")
```

which outputs:

```
$X
      A->B       A->C       B->C       C->A       B->D       D->A
1.3333333 0.6666667 0.3333333 1.0000000 1.0000000 1.0000000

$unconstrained.solution
[1] 1.3333333 0.6666667 0.3333333 1.0000000 1.0000000 1.0000000

$residualNorm
[1] 1.887379e-15

$solutionNorm
[1] 5.333333
```

```
$IsError
[1] FALSE

$type
[1] "ldei"
```

Now we want solve the model successively for increasing values of the flow f1.

Here is how to do this: we first create the `liminput` structure by just reading the input file (`Read("simple.input")`.

This structure has a list item called "pars" which looks like:

```
$pars
     name nr val par1 par2 par3 par4 var flow comp external reaction
left   f1  1   1   NA   NA   NA   NA  NA   NA   NA       NA       NA
```

(it specifies the name `f1` and the value (`val`) = 1.

We then loop over all required values of f1 (`for ( pars in seq(0,1,by=0.2))`), each time setting the value of the parameter in the liminput structure (`ls$pars$val <- pars`) and then solving the model with this input.

```
ls <- Read("simple.input")

Res <- NULL
for ( pars in seq(0,1,by=0.2))
{
 ls$pars$val <- pars
 Res<- rbind(Res,c(pars,Ldei(Setup(ls))$X))
}
Res
```

The results look like:

```
               A->B      A->C        B->C C->A B->D D->A
[1,] 0.0 0.0000000 0.0000000 0.00000000  0.0  0.0  0.0
[2,] 0.2 0.2666667 0.1333333 0.06666667  0.2  0.2  0.2
[3,] 0.4 0.5333333 0.2666667 0.13333333  0.4  0.4  0.4
[4,] 0.6 0.8000000 0.4000000 0.20000000  0.6  0.6  0.6
[5,] 0.8 1.0666667 0.5333333 0.26666667  0.8  0.8  0.8
[6,] 1.0 1.3333333 0.6666667 0.33333333  1.0  1.0  1.0
```

5

---

[5]For the die-hard who has actually reached this part of the vignette. R makes a vignette only from files that have extension "rnw" and that are then processed by R-function Sweave (Leisch 2002). Sweave interprets the R-code and makes a tex file. When I started writing, I did not yet know how to Sweave. By renaming the "tex" file as "rnw", R is tricked to believing it is a true Sweave file (which it is not) and thus makes a vignette. This means that the "R-code" that you can read is not interpreted

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

A list of useful functions in other packages is table (2)

Table 2: Table 1. Summary of the functions in package **limSolve**

Function	Description
Flowmatrix	Generates a flow matrix from a LIM problem
Plotranges	Plots minimum and maximum (ranges) and a central value of a LIM problem
PrintMat	Print the matrices of a LIM problem
Read	Reads a LIM input file and creates a liminput list
Setup	Composes a LIM problem from either a liminput list or from a file
Ldei	Solves a LIM problem using ldei (Least distance programming with equalities and inequalities)
Linp	Solves a LIM problem using Linear programming
Lsei	Solves a LIM problem using lsei (Least squares with equality and inequality conditions)
Xranges	Calculates ranges of unknowns
Varranges	Calculates ranges of variables (linear combinations of unknonws)
Xsample	Randomly samples a LIM problem for the unknowns
Varsample	Randomly samples a LIM problem for the inverse variables

Table 3: Table 2. Useful functions from other packages: **diagram** (Soetaert 2008), **ToxLim** (de Laender *et al.* 2009) and **NetIndices** (Soetaert and Kones 2008; Kones *et al.* 2009)

Function	Package	Description
plotweb	diagram	Plots a web, based on a flowmatrix [6], thickness of arrow =value of flow
plotmat	diagram	Visualises the transpose of a flowmatrix [6] as labeled boxes connected by arrows (created using **LIM** function Flowmatrix
LimOmega	ToxLim	Predicts internal concentrations of hydrophobic chemicals in aquatic organisms, based on a LIM
AscInd	NetIndices	Based on a flowmatrix [6], estimates the ascendency network indices
Dependency	NetIndices	Based on a flowmatrix [6], estimates the dependency network indices
EffInd	NetIndices	Based on a flowmatrix [6], estimates the effective connectivity, flows, nodes, roles network indices
EnvInd	NetIndices	Based on a flowmatrix [6], estimates the indices of homogenization, synergism,...
GenInd	NetIndices	Based on a flowmatrix [6], estimates the general network indices
PathInd	NetIndices	Based on a flowmatrix [6], estimates the direct and indirecgt pathways in a network
UncInd	NetIndices	Based on a flowmatrix [6], estimates the statistical, realised and conditional uncertainty,...
TrophInd	NetIndices	Based on a flowmatrix [6], estimates the trophic level and omnivory indices

References

de Laender F, van Oevelen D, Middelburg JJ, Soetaert K (2009). "Incorporating Ecological Data and Associated Uncertainty in Bioaccumulation Modeling: Methodology Development and Case Study." *Environmental Science and Technology*, **43 No. 7**, 2620–2626.

Diffendorfer J, Richards P, Dalrymple G, DeAngelis D (2001). "Applying Linear Programming to estimate fluxes in ecosystems or food webs: an example from the herpetological assemblage of the freshwater Everglades." *Ecological Modelling*, **144**, 99–120.

Donali E, Olli K, Heiskanen AS, Andersen T (1999). "Carbon flow patterns in the planktonic food web of the Gulf of Riga, the Baltic Sea: a reconstruction by the inverse method." *Journal of Marine Systems*, **23**, 251–268.

Edwards J, Covert M, Palsson B (2002). "Metabolic Modeling of Microbes: the Flux Balance Approach." *Environmental Microbiology*, **4(3)**, 133–140.

Eldridge P, Jackson G (1993). "Benthic trophic dynamics in California coastal basin and continental slope communities inferred using inverse analysis." *Marine Ecology Progress Series*, **99**, 115–135.

Kones JK, Soetaert K, van Oevelen D, Owino J (2009). "Are network indices robust indicators of food web functioning? a Monte Carlo approach." *Ecological Modelling*, **220**, 370–382. doi:http://dx.doi.org/10.1016/j.ecolmodel.2008.10.012.

Leguerrier D, Niquil N, Boileau N, Rzeznik J, Sauriau P, Le Moine O, Bacher C (2003). "Numerical analysis of the food web of an intertidal mudflat ecosystem on the Atlantic coast of France." *Marine Ecology Progress Series*, **246**, 17–37.

Leisch F (2002). "Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis." In W Härdle, B Rönz (eds.), *Compstat 2002 - Proceedings in Computational Statistics*, pp. 575–580. Physica Verlag, Heidelberg. ISBN 3-7908-1517-9, URL http://www.stat.uni-muenchen.de/~leisch/Sweave.

Niquil N, Jackson G, Legendre L, Delesalle B (1998). "Inverse model analysis of the planktonic food web of Takapoto Atoll (French Polynesia)." *Marine Ecology Progress Series*, **165**, 17–29.

Soetaert K (2008). *diagram: Functions for visualising simple graphs (networks), plotting flow diagrams*. R package version 1.4.

Soetaert K (2009). *Escherichia coli Core Metabolism Model in LIM*. LIM package vignette.

Soetaert K, Kones JK (2008). *NetIndices: Estimating network indices, including trophic structure of foodwebs in R*. R package version 1.2.

Soetaert K, Van den Meersche K, van Oevelen D (2009). *limSolve: Solving linear inverse models*. R package version 1.5.

Soetaert K, van Oevelen D (2009). *LIM: Linear Inverse Model examples and solution methods.* R package version 1.4.

Van den Meersche K, Soetaert K, Van Oevelen D (2009). "`xsample()`: An R Function for Sampling Linear Inverse Problems." *Journal of Statistical Software, Code Snippets*, **30**(1), 1–15. URL http://www.jstatsoft.org/v30/c01/.

Van Oevelen D, Soetaert K, Middelburg J, Herman P, Moodley L, Hamels I, Moens T, Heip C (2006). "Carbon flows through a benthic food web: Integrating biomass, isotope and tracer data." *Journal of Marine Research*, **64**, 1–30.

van Oevelen D, van den Meersche K, Meysman F, Soetaert K, Middelburg JJ, Vezina AF (2009). "Quantifying Food Web Flows Using Linear Inverse Models." *Ecosystems*. doi:10.1007/s10021-009-9297-6. URL http://www.springerlink.com/content/4q6h4011511731m5/fulltext.pdf.

Vanderbei R (2000). *Linear programming, Foundations and extensions.* Kluwer Academic Publishers.

Affiliation:

Karline Soetaert
Royal Netherlands Institute of Sea Research (NIOZ)
4401 NT Yerseke, Netherlands E-mail: k.soetaert@nioz.nl
URL: http://www.nioz.nl

Dick van Oevelen
Royal Netherlands Institute of Sea Research (NIOZ)
4401 NT Yerseke, Netherlands E-mail: d.vanoevelen@nioz.nl
URL: http://www.nioz.nl