fluxweb: easily estimate energy fluxes in food webs with R

Benoit Gauzens[1,2], Andrew Barnes[1,3,4], Darren Giling[1,2,5], Jes Hines[1,5], Malte Jochum[6], Jonathan S. Lefcheck[7], Benjamin Rosenbaum[1,2], Shaopeng Wang[1,2,8], and Ulrich Brose[1,2]

[1]German Centre for Integrative Biodiversity Research (iDiv) Halle-Jena-Leipzig, Deutscher Platz 5e, 04103 Leipzig, Germany
[2]Institute of Biodiversity, University of Jena, Dornburger Str. 159, 07743 Jena, Germany
[3]School of Science, University of Waikato, Private Bag 3105, Hamilton 3204, New Zealand
[4]Institute of Landscape Ecology, University of Münster, Heisenbergstrasse 2, 48149 Münster, Germany
[5]Institute of Biology, Leipzig University, Johannisallee 21, 04103 Leipzig, Germany
[6]Institute of Plant Sciences, University of Bern, Altenbergrain 21, 3013 Bern, Switzerland
[7]Tennenbaum Marine Observatories Network, MarineGEO, Smithsonian Institution, Edgewater, Maryland
[8]Institute of Ecology, College of Urban and Environmental Science, Peking University, Beijing 100871, China

June 18, 2025

As a very simple example how to convert community data into quantitative fluxes, we propose guidelines for experimental ecologists who want to use *fluxweb* under the assumptions of the metabolic theory of ecology.

1 Preparing the data

Using *fluxweb* will require the following data:

- a matrix defining the set of **trophic interactions** between each species pair of the ecological system considered (hereafter called *food.web*).
- A vector with the average **body masses** of species (in g, hereafter called *bodymasses*).

- A vector with the total **biomass** of each population (in g, hereafter called *biomasses*).
- A vector with the **organism type** (i.e. plant, animal or detritus) of each species (hereafter called *org.types*).

Then, a vector of **metabolic types** (as defined in Table 1) of each species (thereafter called *met.types*) is not mandatory to calculate fluxes but is a set of easily accessible information that can increase the precision of metabolic rate estimations.

All of these details will allow the definition the mandatory arguments needed to calculate the fluxes: the food web (the matrix *food.web*), the physiological losses (vector *losses*), and the efficiencies (vector *efficiencies*).

food.web Information about the **food web** is the first parameter required by the fluxing function. It should be a matrix (thereafter called `mat`) of n rows and n columns, where n is the total number of species involved in the study. The order of species should be identical between rows and columns. A non-zero value at the intersection of line i and column j in the food web matrix means that predator j consumes prey i. The values used to fill this matrix can be either binary (0/1) assuming that predators' foraging preferences on their prey are unknown, or real values, defining these foraging preferences.

losses The **losses** parameter will be defined in this context as metabolic rates. They are calculated using the species body masses. This calculation can be achieved using eq. 1. It is possible to define the parameters of this equation depending on species **metabolic types** (see table 1 or [**?**]), or to use an average value. In the case of an average value, the per unit of biomass (i.e. g) metabolic rate X is:

$$X = aM^b, \tag{1}$$

where M is the body mass of the species. The average values over all species groups for parameters a and b are 0.71 and -0.25. Then, the corresponding R line of code is:

```
losses = 0.71 * bodymasses^(-0.25)
```

It is possible to obtain a more precise estimation of species metabolic rates, considering the parameters of Table 1 defined for each entry of the

vector *met.types*. Then, the definition of the vector *losses* containing species' metabolic rates can be achieved with:

```
# first, create an empty vector where length is equal to
# the number of species in the food web (nb.species)
losses = rep(NA, nb.species)
# then define values associated to the different metabolic types,
# using species' body mass (stored in the vector body.masses)
ecto.vert = met.types == 'Ectotherm vertebrates'
endo.vert = met.types == 'Endotherm vertebrates'
inv = met.types == 'Invertebrates'
losses[ecto.vert] = 18.18 * bodymasses[ecto.vert]^(-0.29)
losses[endo.vert] = 19.5 * bodymasses[endo.vert]^(-0.29)
losses[inv] = 18.18 * body.masses[inv]^(-0.29)
```

It is important to note that the calculation of metabolic rates using the equations from the metabolic theory of ecology leads to values were units are per-gram of biomass, they do not correspond to the total energetic losses of the entire populations (which can be obtained by multiplied the per-gram of biomass rates by the total biomass of the population). It is quite common in food webs to have nodes such as 'detritus' or 'dissolved organic matter'. Values for the metabolic rates of such nodes can be set to `NA` if they are basal and zero in any case.

Table 1: Parameter values used for the calculation of species metabolic rates depending on their metabolic types. Values from [**?**]

Metabolic type	**intercept**(a)	**exponent** (b)
Ectotherm vertebrates	18.18	-0.29
Endotherm vertebrates	19.5	-0.29
Invertebrates	17.17	-0.29

efficiencies The last parameter needed to estimate fluxes is the vector of feeding **efficiencies**. Because species' physiological losses were estimated using metabolic rates, assimilation efficiencies should be used (assimilation efficiency defines the proportion of eaten biomass that can be used for biomass production plus metabolism [**?**]). These efficiencies can be defined using basic information on organism types. Indeed, the efficiency with which a predator will assimilate energy from a prey can be defined by the type of prey eaten.

Considering a vector *org.type* defining the organism types of food web nodes as 'animal', 'plant' or 'detritus', efficiency values for these three categories are respectively 0.906, 0.545 and 0.158 [**?**]. The vector of *efficiencies* can be created like:

```
# first, create an empty vector where length is equal to
# the number of species in the food web (nb.species)
efficiencies = rep(NA, nb.species)
# then define values associated to organisms types
efficiencies[org.type == 'animal'] = 0.906
efficiencies[org.type == 'plant'] = 0.545
efficiencies[ org.type == 'detritus'] = 0.158
```

2 Calculating fluxes

Once the data set is prepared as described above, the calculation of fluxes is straightforward. It is simply achieved using the fluxing function:

```
mat.fluxes = fluxing(mat, biomasses, losses, efficiencies)
```

where *mat.fluxes* is a matrix containing the fluxes between each species pair. At this point it is important to realize that we used the default behaviour of the *fluxing* function and that several options are hidden so far. Indeed, we use the default values of the optional arguments:

- *bioms.pref* = *TRUE* will scale the species diet preferences (i.e. the values from the food web matrix *mat*) to the biomasses of their prey, according to this equation:

$$W_{i,j} = \frac{mat[i,j] * biomasses[i]}{\sum_k mat[i,k] * biomasses[k]} \quad (2)$$

  where $W_{i,j}$ is the scaled preference of predator j on prey i

- *bioms.losses* = *TRUE* will calculate the total losses of species as the product of the term by term product of the vectors *losses* and *biomasses*. Thus, setting this option to TRUE corresponds to a dataset were species' metabolic losses where defined per unit of biomass. If species losses where directly measured at the population scale (using some respiration measurement for example), this parameter should be set to FALSE.

- *ef.level = "prey"* will assume that the species efficiencies are defined according to prey (i.e., for each species, it is the efficiency with which it will be assimilated once it has been preyed upon).

Using this methodology to compute fluxes with the *species.level* example (fig. 1) dataset would lead to the following lines of code:

```
# first attach the dataset
attach(species.level)
# create the vector of metabolic rates.
# In this example it is done using the general allometric equation:
met.rates = 0.71*species.level$bodymasses^-0.25
# The efficiencies are already defined in the efficiencies vector.
# Then the network of fluxes is obtained using:
mat.fluxes = fluxing(mat, biomasses, met.rates, efficiencies)
```

3 From fluxes to function

Once the matrix of fluxes is obtained, it is possible to estimate some ecosystem functions such as herbivory, detritivory or carnivory. In the following, we will define them as the sum of fluxes outgoing from plant, detritus and animal nodes, respectively. It is important to note that the fluxes estimated by the *fluxing* function correspond to energy loss from resource nodes. They differ from the energy assimilated by consumer nodes due to assimilation efficiencies. Thus, functions from the species.level example (fig. 2) can be estimated by simple sum operations on the *mat.fluxes*:

```
# basal species are species without prey
basals = colSums(mat.fluxes) == 0
names[basals]
# plants are basal species that are not  organic matter or exudates
plants = basals
plants[which(names == 'dead organic matter'
                | names == 'root exudates')] = FALSE

# Herbivory is defined as the sum of fluxes
# outgoing from plant consumers
herbivory = sum(rowSums(mat.fluxes[plants, ]))
# Carnivory is defined as the sum of
```

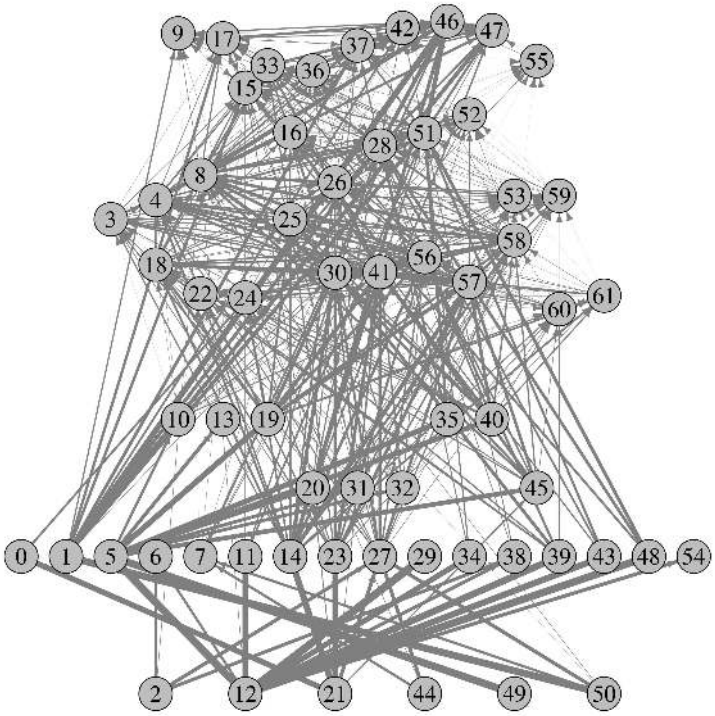

Figure 1: Representation of the *species.level* food web. Width of links scales with the log of fluxes. Nodes' labels correpond to the species ordering in the *species.level* dataset

```
# fluxes outgoing from animals
carnivory = sum(rowSums(mat.fluxes[!basals, ]))
# detritivory is defined as the sum of fluxes
# outgoing from detritus consumers
detritivory = sum(mat.fluxes[names == 'dead organic matter',])
# total fluxes
total = sum(mat.fluxes)
```

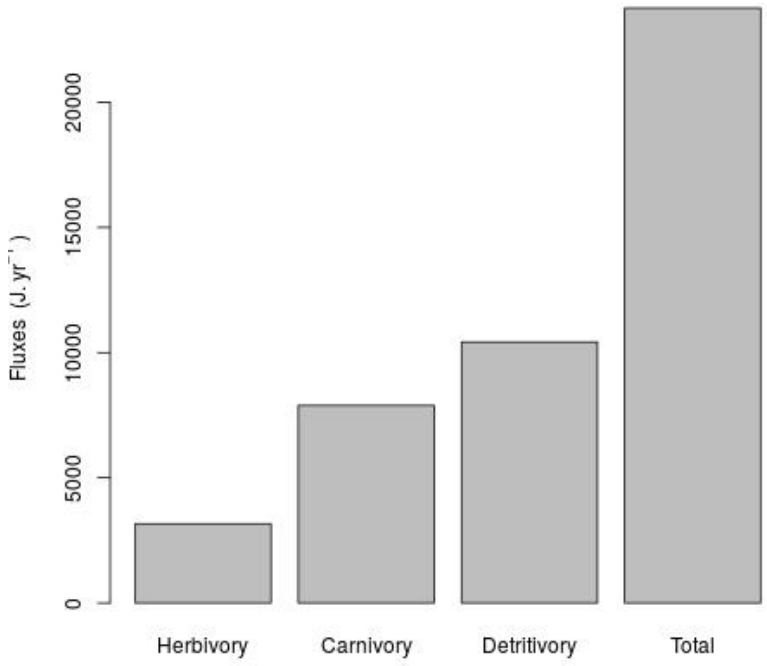

Figure 2: Estimation of the herbivory, carnivory and detritivory functions for the *species.level* food web, as well as the total amount of energy transiting in the food web over one year.

4 Sensitivity to input parameters

We estimated here if the uncertainty or the lack of precision of the estimation of parameters tended to lead to large errors in the estimation of fluxes. To do so, we used the *sensitivity* function to estimate the sensitivity of the *fluxing* function to input parameters. The *sensitivity* function applies a random variation to a selected input parameter of the *fluxing* function. As a result, it returns a matrix containing, for each for each flux, its average coefficient of variation, estimated as:

$$cv = \frac{F'[i,j] - F[i,j]}{F[i,j]}$$

were $F[i,j]$ is the flux from species i to species j when no variation is applied to parameters and $F'[i,j]$ is its equivalent when a random variation

is applied.

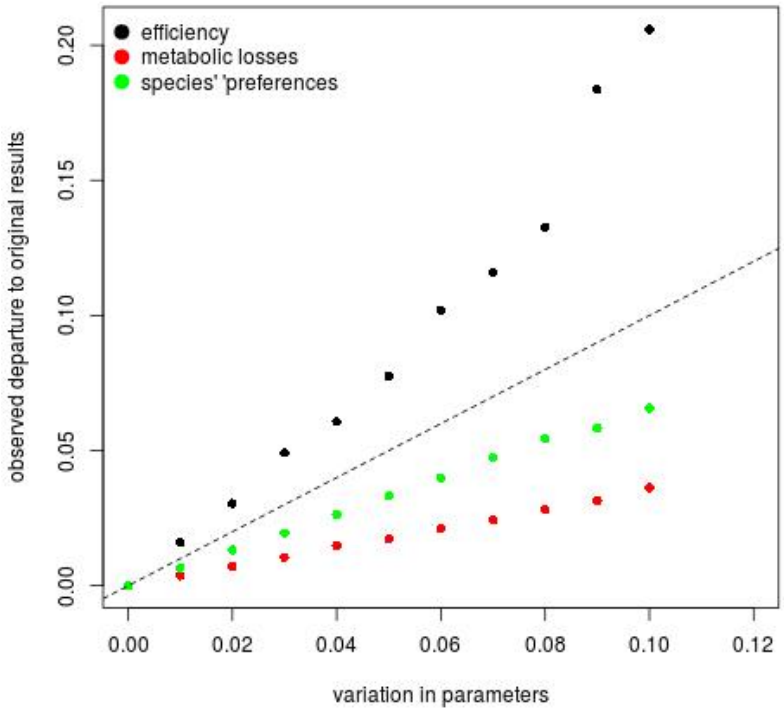

Figure 3: Representation of output uncertainties of the *fluxing* function (y axis) when a random variation is applied to an input parameter. The x axis represent the uncertainty applied to parameters (*var* argument of the *sensitivity* function). The dashed line represents the identity.

Here, we considered the sensitivity of the *fluxing* function to the *losses*, *efficiencies* and *preferences* parameters using the *species.level* example.
For each of these parameters, we simulated an uncertainty on the precision of the estimation method by increasing the value of the *var* parameter of *sensitivity* from 0 to 0.12 (by steps of 0.01) using 50 replicates each. Thus, for each flux and each parameter variation, we obtained the standard deviation of its departure (*cv*) to the original value. To summarise these results, we calculated the mean of these standard deviations over all fluxes for each parameter variation. We then obtained a scalar value representative of the uncertainty of the result of the *fluxing* function depending on the lack of precision of parameter estimation. The fig. 3 was generated using the following

code:

```
attach(species.level)
set.seed(12)
losses = 0.71*bodymasses^-0.25

# creation of vectors to store the standard deviation of c.v.
# for each uncertainty level
sd.cvs.eff = c()
sd.cvs.los = c()
sd.cvs.mat = c()


for (var in seq(0, 0.12, 0.01)){
  cat('var: ', var, '\n')
  # for efficiencies
  res = sensitivity(fluxing, "efficiencies", var, 50,
                    mat = mat,
                    biomasses = biomasses,
                    losses = losses,
                    efficiencies = efficiencies)
  sd.cvs.eff = c(sd.cvs.eff, mean(res[[2]], na.rm = T))

  # for losses
  res = sensitivity(fluxing, "losses", var, 50,
                    mat = mat,
                    biomasses = biomasses,
                    losses = losses,
                    efficiencies = efficiencies)
  sd.cvs.los = c(sd.cvs.los, mean(res[[2]], na.rm = T))

  # for preferences
  res = sensitivity(fluxing, "mat", var, 50,
                    mat = mat,
                    biomasses = biomasses,
                    losses = losses,
                    efficiencies = efficiencies)
  sd.cvs.mat = c(sd.cvs.mat, mean(res[[2]], na.rm = T))
}
```

```
plot(sd.cvs.eff ~ seq, xlim = c(0,0.12),
     xlab = 'variation in parameters',
     ylab = 'observed departure to original results',
     pch = 16)
points(sd.cvs.los ~ seq, col = 'red', pch = 16)
points(sd.cvs.mat ~ seq, col = 'green', pch = 16)
abline(a = 0, b= 1, lty = 2)
legend('topleft',
       legend = c('efficiency', 'metabolic losses',
                          "species' 'preferences"),
       col = c('black', 'red', 'green'),
       pt.cex=1.5, bty='n',
       pt.bg = c('black', 'red', 'green'), pch = 21)
```