A Systems Biology Markup Language package for R

SBMLR Version 1.0 (10/25/2004)

By Tomas Radivoyevitch

Department of Epidemiology and Biostatistics
Case Western Reserve University
Cleveland, Ohio 44106

Email: radivot@hal.cwru.edu

Systems biology markup language (SBML) (http://sbml.org/) is a standard for representing dynamical systems of biological interest [1, 2]. This package provides an SBML-like R model structure, referred to as SBMLR, and 4 functions: `write.sbml` for exporting SBMLR models to SBML level 2 models, `read.SBML` for reading SBML level 2 models into SBMLR model files, `getIncidenceMatrix` for creating stoichiometic matrices given the reactant and product lists in a specific SBMLR model, and `fderiv` as a generic state derivative function that gets passed to `lsoda` of the ODESOLVE package, i.e. for SBMLR simulation applications. SBMLR is the name of both this package and its most central player, the SBML-like model structure; SBMLR will eventually become the name of the class of such model objects, i.e. with instances corresponding to specific dynamical systems.

Models of purine metabolism [3] and folate metabolism [4] are provided in SBMLR format. The package converts these models into valid SBML level 2 code which can be successfully reconverted back into SBMLR representations. Source code implementation details are provided at the end of this document. These details are provided in case a specific SBML model requires source code alterations. This scenario can be expected since SBMLR development has been demand-driven and as a result, some features of SBML level 2 have not been implemented, e.g. events, algebraic rules and function definitions. SBML level 1 is completely ignored in this package; level 1 models can be converted to and from level 2 models using packages such as Jdesigner [5].

SBMLR Model Structure Definition

To facilitate mappings between SBML and R we define an SBMLR structure as illustrated below for the purine metabolism model of Curto et al. [3].

```
model=list(
notes=c("This is a purine metabolism model that is geared toward studies of gout.",
"The model is fully described in Curto et al., MBSC 151 (1998) pp 1-49",
"The model uses Generalized Mass Action (power law) descriptions of reaction rate laws.",
"Such descriptions are local approximations that assume independent substrate binding."),

comps=list(list(id= "cell",vol=1)),

species=list( # IC's have already been run out to the system's steady state.
PRPP    =list(id="PRPP", ic=5,        comp="cell",    bc=F),
…
Pi      =list(id="Pi",   ic=1400,     comp="cell",    bc=T)
),

rxns=list(
list(   id="ada",             rever=FALSE,
        reacts=c("ATP"),
```

```
        prods =c("HX"),
        params=c(aada =0.001062, fada4 =0.97),
        law   = function(r,p)
        {aada=p["aada"];fada4=p["fada4"]
        ATP=r["ATP"]
        aada*ATP^fada4 }),
…
),
units=c("micromolar","minutes")
) # end model list
```

Here ellipses (…) indicate missing code not critical to current discussions; complete source codes are provided with the package. The essential components of SBML, namely, compartments, species (states and boundary conditions) and reactions, are all present in SBMLR. Each reaction is a list that includes a reaction id, the names of species that are reactants (reacts), the names of species that are reaction rate modulators (mods), the names of species that are produced by the reaction (prods), parameter values (params), and the reaction rate law (law) function definition. In this framework, only state variables need be listed as products, boundary condition reactants can equivalently be listed as modulators, and missing terms (e.g. mods in the "ada" reaction above) are equivalent to a NULL assignment. The rate law function has as its input arguments two vectors, one carrying the concentrations of reactants and modulators (r), the other carrying reaction parameter values (p). If the body of the rate law function contains *n* statements, the first *n*-1 trivially convert input vector components into variables with the same names. The *n*th statement then contains the complete reaction rate law. It can occupy multiple lines, but it must be a single statement, i.e. it cannot depend on substitution variables temporarily defined in preceding statements.

SBMLR Model Execution

Model definition codes such as those given above, when placed in a separate file (e.g. Curto.r), can be sourced into a parent script to become globally available for simulations. For example, the purine metabolism model of Curto et al. [3, 6] can be simulated using the following execution code:

```
library(odesolve)
library(SBMLR)
setwd(file.path(.path.package("SBMLR"), "demo"))
source("curto.r") # this file defines Curto et al.'s model as an SBML-R model structure

nrxns=length(model$rxns);nspcs=length(model$species);   # number of reactions and species
S0=NULL;BC=NULL;rIDs=NULL  # initialize before assignments
for (j in 1:nrxns) rIDs[j]<-model$rxns[[j]]$id
for (i in 1:nspcs)
names(S0)<-names(model$species)
y0=S0[BC==FALSE]
nStates=length(y0)
my.atol <- rep(1e-4,nStates)
finalT=70
incid=getIncidenceMatrix(model,BC,y0,nStates,nrxns,nspcs)
# NOTE: model,incid, nStates, nrxns, rIDs, S0 and BC are all passed globally to fderiv
out1=lsoda(y=y0,times=seq(-20,0,1),fderiv,  parms=c(test1=1),  rtol=1e-4, atol= my.atol)
ny0=out1[nrow(out1),2:(nStates+1)]
ny0["PRPP"]=50  # step response to PRPP change from 5 uM to 50 uM
out2=lsoda(y=ny0,times=seq(0,finalT,1),fderiv,  parms=c(test1=1),  rtol=1e-4, atol= my.atol)
outs=data.frame(rbind(out1,out2));#outs
# the next block plots the dynamic responses in fig. 2 of Curto et al (1998) Math Biosci
attach(outs)
par(mfrow=c(2,1))
plot(time,IMP,type="l")
plot(time,HX,type="l")
par(mfrow=c(1,1))
detach(outs)
```

This code simulates the response to a 10-fold increase in phosphoribosylpyrophosphate (PRPP) at time t=0 and plots the responses of inosine monophosphate (IMP) and hypoxanthine (HX) as shown in Figure 1. Two SBMLR defined functions called by this script are: `getIncidenceMatrix()`, which computes the incidence/stoichiometry matrix, and `fderiv()`, which computes state derivatives for integration by the function `lsoda()` of the "odesolve" package.

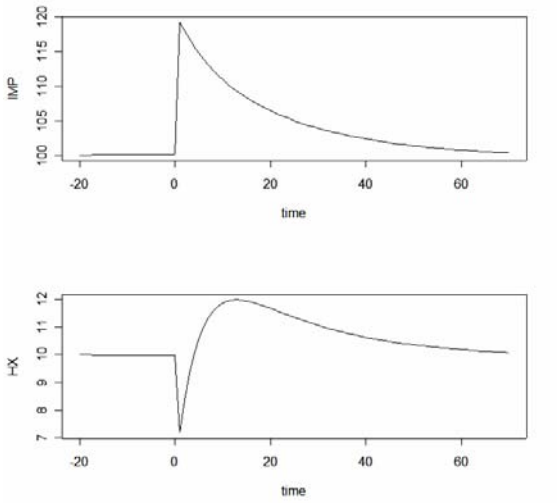

Figure 1: Purine metabolism model of Curto *et al.* responding to a 10-fold increase in phosphoribosylpyrophosphate (PRPP) at time t=0. IMP is inosine monophosphate, HX is hypoxanthine, time is in minutes, and concentration is in μM.

File Conversions

The following code converts the SBMLR model defined in Curto.r, which lives in the demo directory (e.g. C:\Program Files\R\rw2000\library\SBMLR\demo), into an SBML model Curto.xml in the same directory. It then converts this new SBML file (the reader should check to see that it is valid SBML through http://sbml.org/tools/htdocs/sbmltools.php) to recreate SBMLR code of the model in the file CurtoX.r. Note that the sourced files must be in the working directory, here set to the demo directory.

```
library(odesolve)
library(SBMLR)
setwd(file.path(.path.package("SBMLR"), "demo"))
source("curto.r") # this file defines Curto et al.'s model as an SBML-R model structure
write.SBML("Curto") # writes model to SBML (level 2) file Curto.xml
read.SBML("Curto") # converts SBML file Curto.xml into SBMLR model file CurtoX.r
```

To see that the final file conversion executed at the end of this script was successful, the simulation code given above (provided as runCurto.r in the package's demo directory) should be changed to act on the newly created CurtoX.r instead of Curto.r. The plots come out exactly as before (figure 1). The SBML file Curto.xml can now be imported into other SBML-based software (see http://sbml.org/).

SBML assignment rules are implemented by this package. They are used to implement equilibriums that are rapid relative to the time constants of the system. They can also be used to create auxiliary variables, e.g. to track a non-state variable of interest. Examples of these two cases are illustrated in the demo directory using an SBMLR representation of the folate model by Morrison and Allegra [4].

Conclusions

Compared to Matlab, which may be better equipped than R to simulate arbitrarily complex dynamical systems, R has the advantage of list handling infrastructure in `parse()` and `xmlTreeParse()`, and it also has the advantage of indexing by names instead of numbers. A further advantage, though not

exploited here, is that R is object-oriented; in future versions of this interface, a `print()` method might be defined for objects of class SBMLR (i.e. models) to generate more readable renderings of models in R. Another advantage of R over Matlab is that it provides access to a much broader collection of microarray analysis tools, e.g. see bioconductor.org. This aspect is important for those individuals who are interested in biochemical systems analyses of microarray data [7, 8]. For statisticians already familiar with R, there are also the obvious economies of maintaining system familiarity. Finally, perhaps the biggest advantage of R over Matlab is that it is free.

Implementation Details

As SBML evolves to handle a broader range of dynamical systems, it will become more and more challenging for simulation packages to handle all possible SBML models. It is envisioned here that the development of this SBML-R interface will be driven by its users, and not by the model representation capabilities of SBML, i.e. it is expected that the users of this interface will be programmers who are capable of modifying it as their needs require. The following sections are intended for R programmers interested in making code modifications to the SBMLR package.

In **`getIncidenceMatrix`**`()`, the incidence matrix is generated automatically using an *i* loop over the rows (i.e. state variables) and a *j* loop over the columns (i.e. reactions). If a state is a product of a reaction, the corresponding matrix element becomes a positive integer equal to its stoichiometry [`factor()` converts string names to factors so that `summary()` can count them], and similarly for reactants, though with negative numbers entering the matrix in this case (or possibly zero, if a reactant of a reaction happens to also be a product of the same reaction).

The function **`fderiv`**`()` creates the current species vector by overriding initial states with current states clipped to positive values, and by overriding any time varying boundary conditions defined by rules. The function `fderiv()` then computes the reaction rate flux vector (v) based on the current species vector (St) and multiplies it by the incidence matrix to produce the current state derivative vector (xp). The names of xp and v are reset at the end of each function call to override the problem of variables gaining new composite names from the names of their expression arguments.

`write.SBML``()` converts SBMLR models (e.g. Curto.r) into SBML models (e.g. Curto.xml), and `read.SBML()` converts SBML models (e.g. Curto.xml) into an SBMLR models (e.g. CurtoX.r) with an added X (to denote that it came from XML) in the name to avoid overwriting the original which may contain comments not carried over. A key component of these two interface functions is a locally defined recursive function named `recurs()`. This function converts arbitrary R expressions into arbitrary MathML expressions, and vice-versa; it is defined differently, locally, in each of `write.SBML()` and `read.SBML()`. In `write.SBML` the function `recurs()` includes `R2ML()`, which maps R operator symbols into MathML operator symbols. This function initially takes as its input argument the last component of the body of the kinetic rate law function definition, which is the entire rate law expression (as mentioned above, rate laws involving multiple R statements are not supported). In R, expressions are LISP like in that they contain a first element, the operator, and the remaining elements, the arguments, any of which can be an expression. If the operator is the parentheses operator, the action taken is that of a unary identity operator, and we simply skip it and move on to its argument since parentheses are not needed in MathML. Each nested call to the function `recurs()` sends "<apply>" and the converted operator to the output file on its way in, and a matching "</apply>" on its way out. Nested calling continues until all nodes of the expression tree are of class

"name" or "numeric," i.e. when all found objects are leaves of the tree rather than "expressions" that require further parsing. Leaves are then sent to the output file bracketed by <ci> and </ci>.

`read.SBML()` maps SBML level 2 files into SBMLR model files. The main difference between this function, `read.SBML()`, and the previous function, `write.SBML()`, is that here, rather than using `parse()` to decompose the list-of-lists structure of the model defined in R, the SBML model is instead decomposed as an XML object using `xmlTreeParse()` of the XML package available to R. In `read.SBML()`, the locally defined recursive function `recurs()` uses an overkill of parentheses to avoid operator precedence issues. This recursive function is passed a MathML reaction rate law which it parses recursively until the leaves of the tree (the "ci") are all found. During the recursion a corresponding R expression is built as a vector of character strings which, upon exit from the last of the recursive calls, is collapsed into a single string and sent to the output file as the last line of the current rate law function definition.

Unlike the package odesolve, the XML package must be installed from a "local zip" after downloading from http://www.omegahat.org/RSXML. With R 2.0 (and lower), an error message from library(XML) can be resolved by moving copies of the *.dll files of the XML package (e.g. in C:\Program Files\R\rw2000\library\XML\libs) into the "C:\windows" directory (where Windows can find them).

Acknowledgements This work was supported by the Biostatistics Core Facility of the Comprehensive Cancer Center of Case Western Reserve University and University Hospitals of Cleveland (P30 CA43703), the Integrative Cancer Biology Program (1 P20 CA112963-01) and the American Cancer Society (IRG-91-022-09).

References

1. Hucka M, Finney A, Sauro HM, Bolouri H, Doyle JC, Kitano H, Arkin AP, Bornstein BJ, Bray D, Cornish-Bowden A *et al*: **The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models**. *Bioinformatics* 2003, **19**(4):524-531.
2. Finney A, Hucka M: **Systems biology markup language: Level 2 and beyond**. *Biochem Soc Trans* 2003, **31**(Pt 6):1472-1473.
3. Curto R, Voit EO, Sorribas A, Cascante M: **Validation and steady-state analysis of a power-law model of purine metabolism in man**. *BiochemJ* 1997, **324 (Pt 3)**:761-775.
4. Morrison PF, Allegra CJ: **Folate cycle kinetics in human breast cancer cells**. *JBiolChem* 1989, **264**(18):10552-10566.
5. Sauro HM, Hucka M, Finney A, Wellock C, Bolouri H, Doyle J, Kitano H: **Next generation simulation tools: the Systems Biology Workbench and BioSPICE integration**. *Omics* 2003, **7**(4):355-372.
6. Curto R, Voit EO, Sorribas A, Cascante M: **Mathematical models of purine metabolism in man**. *MathBiosci* 1998, **151**(1):1-49.
7. Radivoyevitch T: **Sphingoid base metabolism in yeast: Mapping gene expression patterns into qualitative metabolite time course predictions**. *Comparative & Functional Genomics* 2001, **2**:289-294.
8. Voit EO, Radivoyevitch T: **Biochemical systems analysis of genome-wide expression data**. *Bioinformatics* 2000, **16**:1023-1037.