BMC Bioinformatics

BioMed Central

Software

Open Access

A two-way interface between limited Systems Biology Markup Language and R

Tomas Radivoyevitch*

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

Email: Tomas Radivoyevitch* - radivot@hal.cwru.edu

* Corresponding author

Published: 07 December 2004

BMC Bioinformatics 2004, **5**:190 doi:10.1186/1471-2105-5-190

Received: 03 June 2004

Accepted: 07 December 2004

This article is available from: http://www.biomedcentral.com/1471-2105/5/190

Abstract

Background: Systems Biology Markup Language (SBML) is gaining broad usage as a standard for representing dynamical systems as data structures. The open source statistical programming environment R is widely used by biostatisticians involved in microarray analyses. An interface between SBML and R does not exist, though one might be useful to R users interested in SBML, and SBML users interested in R.

Results: A model structure that parallels SBML to a limited degree is defined in R. An interface between this structure and SBML is provided through two function definitions: write.SBML() which maps this R model structure to SBML level 2, and read.SBML() which maps a limited range of SBML level 2 files back to R. A published model of purine metabolism is provided in this SBML-like format and used to test the interface. The model reproduces published time course responses before and after its mapping through SBML.

Conclusions: List infrastructure preexisting in R makes it well-suited for manipulating SBML models. Further developments of this SBML-R interface seem to be warranted.

Background

Systems biology markup language (SBML) is a standard for representing dynamical systems of biological interest [1,2]. Interfaces between SBML and high level computational environments are currently being developed for Mathematica [3] and Matlab [4], but to the author's knowledge, no such efforts are being carried forth for R/S-plus. This brief paper presents the author's initial developments toward a two-way SBML-R interface. The interface is currently limited in the range of SBML input files that it can handle. For example, it only handles SBML level 2 and does not handle "Events" and "FunctionDefinitions." The interface can nevertheless be used for some models, examples [5,6] of which are included under "demo" in the SBMLR package [7]. This paper provides an explicit example of one approach to an SBML-R interface. It is assumed throughout that the reader is already quite familiar with both SBML [8] and R [9].

Implementation

The software exists completely in R. It is comprised of four functions and is currently being distributed as a developmental package called "SBMLR" through Bioconductor [10]. The software was written subject to two constraints: 1) models expressed in SBML-like R must be exchangeable with a range of SBML models; and 2) models must be amenable to simulation in R. The first subsection that follows defines an SBML-like R model structure, the second illustrates how it can be used in simulations, and the third describes its conversions into and out of SBML.

```
# Curto's purine model (curto.r)
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(
PRPP    =list(id="PRPP", ic=5,        comp="cell",    bc=F),
IMP     =list(id="IMP",  ic=100,      comp="cell",    bc=F),
…
UA      =list(id="UA",   ic=100,      comp="cell",    bc=F),
R5P     =list(id="R5P",  ic=18,       comp="cell",    bc=T),
Pi      =list(id="Pi",   ic=1400,     comp="cell",    bc=T)
),

rxns=list(
list(   id="ada",              rever=F,   # v1
        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 }),
…
list(   id="PRPPS",            rever=F,   # v28
        reacts=c("R5P"),
        mods=c("ATP","GTP","Pi","PRPP"),
        prods =c("PRPP"),
        params=c(aprpps=0.9, fprpps1 =-.03, fprpps4 =-.45, fprpps8 =-.04, fprpps17=0.65,fprpps18 =0.7),
        law   = function(r,p)
        {aprpps=p["aprpps"];fprpps1=p["fprpps1"];fprpps4=p["fprpps4"];fprpps8=p["fprpps8"];
        fprpps17=p["fprpps17"];fprpps18=p["fprpps18"]
        PRPP=r["PRPP"];ATP=r["ATP"];GTP=r["GTP"];R5P=r["R5P"];Pi=r["Pi"]
        aprpps * PRPP^fprpps1 * ATP^fprpps4 * GTP^fprpps8 * R5P^fprpps17 * Pi^fprpps18}),
…
list(   id="Vxd",              rever=F,   # v37
        reacts=c("Xa"),
        prods =c("UA"),
        params=c(axd =0.949,fxd14 =0.55),
        law   = function(r,p)
        {axd=p["axd"];fxd14=p["fxd14"]
        Xa=r["Xa"]
        axd *  Xa^fxd14}) ),
units=c("micromolar","minutes")
) # end model list
```

Figure 1
The model of Curto *et al.* implemented as an SBMLR structure

An SBML-Like Model Structure in R

To facilitate mappings between SBML and R, an SBML-like list structure is defined in this subsection using the purine metabolism model of Curto et al. [6] as a specific example (Figure 1). In this figure and elsewhere, ellipses (...) indicate missing code not critical to current discussions; complete source codes are available through the SBMLR package [7]. The essential components of an SBML model, namely, its compartments, species and reactions, are all present in this R analog of an SBML model. In the model of Curto et al. [4], there is one compartment, be it the cell or the entire human body, and 18 species: 2 boundary conditions (bc = True) and 16 state variables (bc = False), each with an initial condition (ic) or value. 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

```
library(SBMLR)   # install SBMLR from Bioconductor's developmental packages repository
library(odesolve)  # this provides lsoda, an ODE solver which does NOT handle events
# The file Curto.r defines Curto et al.'s model as an SBML-R model structure
source(file.path(.path.package("SBMLR"), "demo/Curto.r"))

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){BC[i]=model$species[[i]]$bc; S0[i]=model$species[[i]]$ic}
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)
```

Figure 2
The purine metabolism model of Curto *et al.* represented in SBMLR Figure 1) simulated to respond to a 10-fold increase (5 μM to 50 μM) in phosphoribosylpyrophosphate (PRPP) at time $t = 0$

(e.g. mods in reactions 1 and 37) 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 nth 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.

SBML-like Model Execution in R

Model definition codes such as that given in Figure 1, 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. [6] can be simulated using the execution code shown in Figure 2.

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 3. Two functions called by this script are defined in the SBMLR package and shown in Figure 4. They are, getIncidenceMatrix(), which computes the incidence/stoichiometry matrix used by the second function, fderiv(), which computes state derivatives for integration by the function lsoda() of the "odesolve" 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 (SBML rules are not needed for the purine model, but are needed to implement other

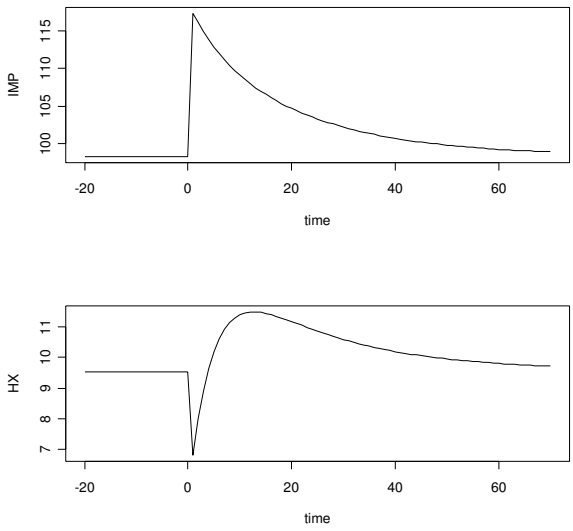

Figure 3
The purine metabolism model of Curto *et al.* responding to a 10-fold increase in phosphoribosylpyrophosphate (PRPP) at time $t = 0$ (see Figure 2). IMP is inosine monophosphate, HX is hypoxanthine, time is in minutes, and concentration is in μM.

models [5]). 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.

A Two-Way Interface between SBML and R

Two functions comprise the SBML-R interface: write.SBML() converts SBML-like R models (e.g. Curto.r) into SBML models (e.g. Curto.xml), and read.SBML() converts SBML models (e.g. Curto.xml) into an SBML-like R model (e.g. CurtoX.r). 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 the two functions. In write.SBML(), shown in Figure 5, recurs() 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>.

The second of the two SBML-R interface functions, read.SBML(), maps a limited range of SBML level 2 files (function definitions and events are not handled) into SBML-like R model files. Portions of read.SBML() are given in Figure 6. 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 [11]. 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.

Results

The function write.SBML() was applied to Curto.r to generate Curto.xml and the function read.SBML() was then applied to Curto.xml to generate CurtoX.r. Execution of the script given in Figure 2 with line 4 of the execution code changed to act on CurtoX.r instead of Curto.r generated the same plots as before (Figure 3). This shows that the R model was successfully converted into an SBML file that can be reconverted back into a properly functioning R model. The intermediate file Curto.xml was successfully validated as an SBML level 2 file [12]. The SBML file could thus be imported into visualization packages such as JDesigner [13].

Discussion

If the model of Curto et al. [4] were implemented in R without any knowledge of SBML, a form that it might take is that given in Appendix B (Figure 7). Compared to its SBML-like counterparts, this code is more compact and easier to understand, e.g. the system's network connectivity is clearly visible. The disadvantage of such code is that

```
getIncidenceMatrix<-function(model,BC,y0,nStates,nrxns,nspcs)
{incid=matrix(rep(0,nStates*nrxns),nrow=nStates)
indx=(1:nspcs)[BC==F]
for (i in 1:nStates)
 for (j in 1:nrxns)
  {if ( is.element(model$species[[indx[i]]]$id, model$rxns[[j]]$prods))
     incid[i,j] = summary(factor(model$rxns[[j]]$prods))[[model$species[[indx[i]]]$id]]
   if ( is.element(model$species[[indx[i]]]$id, model$rxns[[j]]$reacts))
     incid[i,j] = incid[i,j]-summary(factor(model$rxns[[j]]$reacts))[[model$species[[indx[i]]]$id]]}
rownames(incid)<-names(y0)
incid}

fderiv <- function(times, X, p)  # state derivative function sent to ODEsolve
# NOTE: much is passed to fderiv globally since p cannot be a list.
{v=rep(0,nrxns)
xp=rep(0,nStates)
St=S0
X[X<0]=0
St[BC==F]=X
nrules=length(model$rules)
if (nrules>0) for (j in 1:nrules)
       St[model$rules[[j]]$output]=model$rules[[j]]$law(St[model$rule[[j]]$inputs])
for (j in 1:nrxns)  if (model$rxns[[j]]$rever==F)
  v[j]=model$rxns[[j]]$law(St[c(model$rxns[[j]]$reacts,model$rxns[[j]]$mods)],model$rxns[[j]]$params)
xp=incid%*%v
names(xp)<-names(y0)
names(v)<-rIDs
aux=c(v,St[BC==T])
list(xp,aux)}
```

Figure 4
R codes for the functions getIncidenceMatrix() and fderiv().

it is not readily converted into SBML. Since the benefits of SBML are compelling, this disadvantage alone warrants the use of SBML-like model structures.

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.

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 [10]. This aspect is important for those individuals who are interested in biochemical systems analyses of microarray data [14,15]. 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 freely available. On balance, there seems to be ample motivation for further developments of this interface between SBML and R.

Availability and requirements

Project name: SBMLR

Project home page: http://www.bioconductor.org/repository/devel/package/html/SBMLR.html

Operating system(s): Windows XP

Programming language: R 2.0

Other requirements: R packages: XML and ODESOLVE

License: GNU GPL

Any restrictions to use by non-academics: no restrictions

```
write.SBML<-function(filename)
{  # takes R model in filename.r and maps it to an SBML model in filename.xml

r2ml<- function(type)   # maps R operator symbols into MathML
       switch(type,
             "*" = "<times/>",
             "/" = "<divide/>",
             "+" = "<plus/>",
             "-" = "<minus/>",
             "^" = "<power/>",
             "exp" = "<exp/>",
              "log" = "<ln/>",
              "not found")    # end of r2ml sub-function definition.

fid <- file(paste(filename,".xml",sep=""), "w")  # open the output file connection

recurs<-function(last)
{ if(last[[1]]=="(") {last=last[[2]]} # remove parentheses
  cat("    <apply>", file=fid, sep="\n")
  cat(sprintf("      %s",r2ml(as.character(last[[1]])) ), file=fid, sep="\n")
  for (j in 2:length(last))
   if ((class(last[[j]])=="name")|(class(last[[j]])=="numeric"))
    cat(sprintf("     <ci>%s</ci>",as.character(last[[j]]) ) , file=fid, sep="\n")  else
              Recall(last[[j]])
  cat("    </apply>", file=fid, sep="\n")
} # End of locally defined recursive function

e=parse(file=paste(filename,".r",sep=""),n=-1)  # parses the whole file into a list of lists
notes=e[[1]][[3]][["notes"]]
comps=e[[1]][[3]][["comps"]]
species=e[[1]][[3]][["species"]]
parameters=e[[1]][[3]][["parameters"]]
rules=e[[1]][[3]][["rules"]]
rxns=e[[1]][[3]][["rxns"]]
…
cat("  <kineticLaw>", file=fid, sep="\n")
fcn=rxns[[i]][["law"]][[3]]  # the third component is the body of the function
last=fcn[[length(fcn)]]
cat("    <math xmlns=\"http://www.w3.org/1998/Math/MathML\">", file=fid, sep="\n")
recurs(last)
cat("    </math>", file=fid, sep="\n")
cat("    <listOfParameters>", file=fid, sep="\n")
params=rxns[[i]][["params"]]
for (j in 2:length(params))
if (length(params[[j]])==1)
cat(sprintf(" <parameter id = \"%s\" value = \"%g\"/>",
               names(params)[j],params[[j]]), file=fid, sep="\n") else
cat(sprintf(" <parameter id = \"%s\" value = \"%g\"/>",
               names(params)[j],-params[[j]][[2]]), file=fid, sep="\n")
cat("    </listOfParameters>", file=fid, sep="\n")
cat("    </kineticLaw>", file=fid, sep="\n")
cat("  </reaction>", file=fid, sep="\n")
}
cat("</listOfReactions>", file=fid, sep="\n")
cat("</model>", file=fid, sep="\n")
cat("</sbml>", file=fid, sep="\n")
close(fid)
}  # end write.sbml function defition
```

Figure 5
R code for the function write.SBML().

```
read.SBML<-function(filename)
{  # takes SBML in filename.xml and maps it to a model definition in filenameX.r
if(!require(XML)) print("Please install the XML package from http://www.omegahat.org/RSXML")
ML2R<- function(type)   # map MathML operator symbols into R symbols
       switch(type,
             "times" = "*",
             "divide" = "/",
             "plus" = "+",
             "minus" = "-",
             "power" = "^",
             "exp" = "exp",
              "ln" = "log",
              "not found") # end definition of ML2R
recurs<-function(math)
{ n=length(math)
 S=c("(")
 op=ML2R(xmlName(math[[1]]))
 for (j in 2:n )
 if (xmlName(math[[j]])=="ci") S=c(S,as.character(xmlValue(math[[j]])), ifelse(j==n,"",op)) else
 S=c(S,Recall(math[[j]]), ifelse(j==n,"",op))
S=c(S,")")
S
} # end recursive function definition

# start main part of read.SBML functino definition
fid <- file(paste(filename,"X.r",sep=""), "w")  # open the output file connection
cat("#",filename,"X.r\n", file=fid, sep="")  # X indicates R via XML
cat("model=list( ", file=fid, sep="\n")
# now read in a SBML file
doc <- xmlTreeParse(paste(filename,".xml",sep=""),getDTD=F)
cat("notes=c(", file=fid, sep="\n")
notes=doc$children$sbml[["model"]][["notes"]][["body"]]
n=length(xmlChildren(notes))
for (i in 1:n)
cat(sprintf("\"%s\"%s", xmlValue(xmlChildren(notes)[[i]]),ifelse(i==n,"\n),\n",",")),file=fid, sep="\n")
…
cat("rxns=list(", file=fid, sep="\n")
rxns=xmlChildren(doc$children$sbml[["model"]][["listOfReactions"]]); n=length(rxns)
for (i in 1:n)
{
cat(sprintf("list( id=\"%s\", rever=%s,",xmlAttrs(rxns[[i]])[["id"]],ifelse(xmlAttrs(rxns[[i]])[["reversible"]]=="true","T","F")),
                file=fid, sep="\n")
law=rxns[[i]]["kineticLaw"][[1]]
params=law[["listOfParameters"]]
math=law[["math"]][[1]]
reacts=rxns[[i]]["listOfReactants"][[1]]
…
nr=length(reacts)
if (nr>0){cat("reacts=c(", file=fid, sep="")
for (k in 1:nr) cat(sprintf("\"%s\"%s",xmlAttrs(reacts[[k]]), ifelse(k==nr,"),\n",",")  ), file=fid, sep="")}
…
npa=length(params)
if (npa>0){cat("params=c(", file=fid, sep="")
for (k in 1:npa) cat(sprintf("%s = %g%s",xmlAttrs(params[[k]])[["id"]], as.numeric(xmlAttrs(params[[k]])[["value"]]), ifelse(k==npa,"),\n",",")),
                file=fid, sep="")}
# always have a law
cat("law=function(r,p){", file=fid, sep="\n")
if (npa>0) for (k in 1:npa) cat(sprintf("%s = p[\"%s\"];",xmlAttrs(params[[k]])[["id"]],xmlAttrs(params[[k]])[["id"]]  ), file=fid, sep="")
cat(" ", file=fid, sep="\n")
if (nr>0) for (k in 1:nr) cat(sprintf("%s = r[\"%s\"];",xmlAttrs(reacts[[k]]),xmlAttrs(reacts[[k]])), file=fid, sep="")
if (nm>0) for (k in 1:nm) cat(sprintf("%s = r[\"%s\"];",xmlAttrs(mods[[k]]),xmlAttrs(mods[[k]])), file=fid, sep="")
cat(" ", file=fid, sep="\n")
cat(paste(recurs(math),collapse=""), file=fid, sep="")
cat(sprintf("}  \n %s",ifelse(i==n,"))) \n",",")  ), file=fid, sep="\n")
}  # end for loop on reactions
close(fid)
} # end read.SBML function definition
```

Figure 6
R code for the function read.SBML().

List of abbreviations

SBML = Systems Biology Markup Language; XML = extensible markup language; MathML = Mathematical Markup Language; ODE = ordinary differential equation.

Authors' contributions

TR is the sole contributor.

Appendix A

The SBMLR package is available through Bioconductor as a developmental package [7]. It has been developed and tested only under Windows XP. To install, do NOT unzip the file SBMLR.zip after downloading to a local directory, rather, within the R GUI, click *packages* and *install from local zip*. The XML package installs similarly [11]. Note that an error message from library(XML) can be resolved by copying the *.dll files of the XML package libs directory into the "C:\windows" directory. The ODESOLVE package must be installed before running simulations. This package is installed from the R GUI by clicking *packages* and *install from CRAN*.

```
library(odesolve)
fada4 =0.97; fade6 =0.55; fadrnr4 =0.1; fadrnr9 =-0.3;fadrnr10=0.87; fampd4 =0.8; fampd8 =-0.03; fampd18 =-0.1;
faprt1 =0.5; faprt4 =-0.8; faprt6 =0.75; fasuc2 =0.4;fasuc4 =-.24; fasuc8 =0.2; fasuc18 =-.05; fasli3 =0.99;
fasli4 =-.95; fdada9 =1; fden1 =2; fden2 =-.06;fden4 =-.25; fden8 =-.2; fden18 =-.08; fdgnuc10=1;
fdnan12=1; fdnap9 =0.42; fdnap10 =0.33; fgdrnr8 =0.4;fgdrnr9 =-1.2; fgdrnr10=-.39; fgmpr2 =-.15; fgmpr4 =-.07;
fgmpr7 =-.76; fgmpr8 =0.7; fgmps4 =0.12; fgmps7 =0.16;fgnuc8 =0.9; fgnuc18 =-.34; fgprt1 =1.2; fgprt8 =-1.2;
fgprt15 =0.42; fgua15 =0.5; fhprt1 =1.1; fhprt2 =-.89;fhprt13 =0.48; fhx13 =1.12; fhxd13 =0.65; fimpd2 =0.15;
fimpd7 =-.09; fimpd8 =-.03; finuc2 =0.8; finuc18 =-.36;fmat4=0.2; fmat5=-.6; fpolyam5=0.9; fprpps1 =-.03;
fprpps4 =-.45; fprpps8 =-.04; fprpps17=0.65; fprpps18 =0.7;fpyr1 =1.27; frnan11 =1; frnap4 =0.05; frnap8 =0.13;
ftrans5 =0.33; fua16 =2.21; fx14 =2.0; fxd14 =0.55;
aada =0.001062; aade =0.01; aadna=3.2789; aadrnr =0.0602;aampd =0.02688; aaprt =233.8; aarna =614.5; aasuc =3.5932;
aasli =66544; adada =0.03333; aden =5.2728; adgnuc=0.03333;adnaa =0.001938; adnag =0.001318; agdna=2.2296; agdrnr =0.1199;
agmpr =0.3005; agmps=0.3738; agnuc=0.2511; agprt =361.69;agrna =409.6; agua =0.4919; ahprt =12.569; ahx =0.003793;
ahxd =0.2754; aimpd =1.2823; ainuc =0.9135; amat =7.2067;apolyam=0.29; aprpps=0.9; apyr =1.2951; arnaa =0.06923;
arnag =0.04615; atrans =8.8539; aua =0.00008744; ax =0.0012;axd =0.949;
R5P=18; Pi=1400;

fpur <- function(t, X, p)
{vada   =aada   * X[4]^fada4;
vade   =aade   * X[6]^fade6;
vadna  =aadna  * X[9]^fdnap9 * X[10]^fdnap10;
vadrnr =aadrnr * X[4]^fadrnr4 * X[9]^fadrnr9 * X[10]^fadrnr10;
vampd  =aampd  * X[4]^fampd4 * X[8]^fampd8 * Pi^fampd18;
vaprt  =aaprt  * X[1]^faprt1 * X[4]^faprt4 * X[6]^faprt6;
varna  =aarna  * X[4]^frnap4* X[8]^frnap8;
vasuc  =aasuc  * X[2]^fasuc2 * X[4]^fasuc4 * X[8]^fasuc8 * Pi^fasuc18;
vasli  =aasli  * X[3]^fasli3 * X[4]^fasli4;
vdada  =adada  * X[9]^fdada9;
vden   =aden   * X[1]^fden1 *  X[2]^fden2 * X[4]^fden4 * X[8]^fden8 *  Pi^fden18;
vdgnuc =adgnuc * X[10]^fdgnuc10;
vdnaa  =adnaa  * X[12]^fdnan12;
vdnag  =adnag  * X[12]^fdnan12;
vgdna  =agdna  * X[9]^fdnap9 * X[10]^fdnap10;
vgdrnr =agdrnr * X[8]^fgdrnr8 * X[9]^fgdrnr9 * X[10]^fgdrnr10;
vgmpr  =agmpr  * X[2]^fgmpr2 * X[4]^fgmpr4 * X[7]^fgmpr7 * X[8]^fgmpr8;
vgmps  =agmps  * X[4]^fgmps4 * X[7]^fgmps7;
vgnuc  =agnuc  * X[8]^fgnuc8 * Pi^fgnuc18;
vgprt  =agprt  * X[1]^fgprt1 * X[8]^fgprt8* X[15]^fgprt15;
vgrna  =agrna  * X[8]^frnap8* X[4]^frnap4;
vgua   =agua   * X[15]^fgua15;
vhprt  =ahprt  * X[1]^fhprt1 * X[2]^fhprt2 * X[13]^fhprt13;
vhx    =ahx    * X[13]^fhx13;
vhxd   =ahxd   * X[13]^fhxd13;
vimpd  =aimpd  * X[2]^fimpd2 * X[7]^fimpd7 * X[8]^fimpd8;
vinuc  =ainuc  * X[2]^finuc2 * Pi^finuc18;
vmat   =amat   * X[4]^fmat4 * X[5]^fmat5;
vpolyam=apolyam* X[5]^fpolyam5;
vprpps =aprpps * X[1]^fprpps1 * X[4]^fprpps4 * X[8]^fprpps8 * R5P^fprpps17 * Pi^fprpps18;
vpyr   =apyr   * X[1]^fpyr1;
vrnaa  =arnaa  * X[11]^frnan11;
vrnag  =arnag  * X[11]^frnan11;
vtrans =atrans * X[5]^ftrans5;
vua    =aua    * X[16]^fua16;
vx     =ax     * X[14]^fx14;
vxd    =axd    * X[14]^fxd14;
X1p = vprpps-vgprt-vhprt-vaprt-vden-vpyr;
X2p = vden+vgmpr+vhprt+vampd-vimpd-vasuc-vinuc;
X3p = vasuc-vasli;
X4p = vaprt+vasli+vtrans+vrnaa-vmat-vampd-varna-vadrnr-vada;
X5p = vmat-vtrans-vpolyam;
X6p = vpolyam-vaprt-vade;
X7p = vimpd-vgmps;
X8p = vgmps+vrnag+vgprt-vgmpr-vgrna-vgdrnr-vgnuc;
X9p = vadrnr+vdnaa-vadna-vdada;
X10p= vgdrnr+vdnag-vgdna-vdgnuc;
X11p= varna+vgrna-vrnaa-vrnag;
X12p= vgdna+vadna-vdnaa-vdnag;
X13p= vdada+vada+vinuc-vhprt-vhxd-vhx;
X14p= vhxd+vgua-vxd-vx;
X15p= vgnuc+vdgnuc-vgua-vgprt;
X16p= vxd-vua;
XP = c(X1p, X2p, X3p, X4p, X5p, X6p, X7p, X8p, X9p, X10p, X11p, X12p, X13p, X14p, X15p, X16p);
V=c(vada,vade,vadna,vadrnr,vampd,vaprt,varna,vasuc,vasli,vdada,vden,vdgnuc,vdnaa,vdnag,vgdna,vgdrnr,vgmpr,vgmps,vgnuc,
vgprt,vgrna,vgua,vhprt,vhx,vhxd,vimpd,vinuc,vmat,vpolyam,vprpps,vpyr,vrnaa,vrnag,vtrans,vua,vx,vxd)
names(V)<-c("vada","vade","vadna","vadrnr","vampd","vaprt","varna","vasuc","vasli","vdada","vden","vdgnuc",
"vdnaa","vdnag","vgdna","vgdrnr","vgmpr","vgmps","vgnuc","vgprt","vgrna","vgua","vhprt","vhx","vhxd","vimpd",
"vinuc","vmat","vpolyam","vprpps","vpyr","vrnaa","vrnag","vtrans","vua","vx","vxd")
list(XP,V)}

y0=c(PRPP=5,IMP=100,SAMP=.2,Ado=2500,SAM=4,Ade=1,XMP=25,GMP=400,dAdo=6,dGMP=3,RNA=28600,DNA=5160,HX=10,Xa=5,Gua=5,UA=100);
out1=lsoda(y=y0,times=seq(-20,0,1),fpur, parms=c(test1=1), rtol=1e-4, atol= rep(1e-4,16)); ny0=out1[nrow(out1),2:17]
ny0[1]=50  # jump response to a change in PRPP from 5 uM to 50 uM as in Curto's Math Biosci paper
out2=lsoda(y=ny0,times=seq(0,70,1),fpur, parms=c(test1=1), rtol=1e-4, atol= rep(1e-4,16))
outs=data.frame(rbind(out1,out2))
attach(outs);par(mfrow=c(2,1))
plot(time,IMP,type="l");plot(time,HX,type="l")
par(mfrow=c(1,1)); detach(outs)
```

Figure 7
The purine metabolism model of Curto *et al.* implemented in "natural R".

Appendix B

The implementation of Curto et al.'s model shown in Figure 7 is independent of any knowledge of SBML. It is included here to illustrate what comes "naturally" when implementing a model in R, see Discussion.

Acknowledgements

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

References

1. Hucka M, Finney A, Sauro HM, Bolouri H, Doyle JC, Kitano H, Arkin AP, Bornstein BJ, Bray D, Cornish-Bowden A, Cuellar AA, Dronov S, Gilles ED, Ginkel M, Gor V, Goryanin II, Hedley WJ, Hodgman TC, Hofmeyr JH, Hunter PJ, Juty NS, Kasberger JL, Kremling A, Kummer U, Le Novere N, Loew LM, Lucio D, Mendes P, Minch E, Mjolsness ED, Nakayama Y, Nelson MR, Nielsen PF, Sakurada T, Schaff JC, Shapiro BE, Shimizu TS, Spence HD, Stelling J, Takahashi K, Tomita M, Wagner J, Wang J: **The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models.** *Bioinformatics* 2003, **19:**524-531.
2. Finney A, Hucka M: **Systems biology markup language: Level 2 and beyond.** *Biochem Soc Trans* 2003, **31:**1472-1473.
3. Shapiro BE, Hucka M, Finney A, Doyle J: **MathSBML: a package for manipulating SBML-based biological models.** *Bioinformatics* 2004, **20:**2829-2831.
4. Keating SM: **SBMLToolbox.** [http://sbml.org/software/sbmltoolbox/].
5. Morrison PF, Allegra CJ: **Folate cycle kinetics in human breast cancer cells.** *J Biol Chem* 1989, **264:**10552-10566.
6. Curto R, Voit EO, Sorribas A, Cascante M: **Mathematical models of purine metabolism in man.** *Math Biosci* 1998, **151:**1-49.
7. **SBMLR** [http://www.bioconductor.org/repository/devel/package/html/SBMLR.html]
8. **Systems Biology Markup Language** [http://sbml.org]
9. **The R Project for Statistical Computing** [http://www.r-project.org/]
10. **Bioconductor** [http://www.bioconductor.org/]
11. **XML** [http://www.omegahat.org/RSXML/]
12. **SBML Online Tools** [http://sbml.org/tools/htdocs/sbmltools.php]
13. 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:**355-372.
14. Radivoyevitch T: **Sphingoid base metabolism in yeast: Mapping gene expression patterns into qualitative metabolite time course predictions.** *Comparative & Functional Genomics* 2001, **2:**289-294.
15. Voit EO, Radivoyevitch T: **Biochemical systems analysis of genome-wide expression data.** *Bioinformatics* 2000, **16:**1023-1037.