smbl2dMod

Daniel Lill

23 November 2017

Load all important libraries

```
library(dMod)
# devtools::load_all("~/Promotion/Software/dMod/")
library(stringr)
# library(conveniencefunctions)
# library(magrittr)
#' importFrom magrittr "%>%"
# "%>%" <- magrittr::"%>%"
library(tidyverse)
library(magrittr)
library(SBMLR)
```

Load the example model "curto" from the SBMLR-Package and have a first look at it

```
curto_file <- system.file("models", "curto.xml", package = "SBMLR")
curto <- curto_file %>% readSBML()
summary(curto) %>% head
```

```
## $nSpecies
## [1] 18
##
## $sIDs
##  [1] "PRPP" "IMP"  "SAMP" "ATP"  "SAM"  "Ade"  "XMP"  "GTP"  "dATP" "dGTP"
## [11] "RNA"  "DNA"  "HX"   "Xa"   "Gua"  "UA"   "R5P"  "Pi"
##
## $S0
##        PRPP         IMP        SAMP         ATP         SAM         Ade
## 5.01742e+00 9.82634e+01 1.98189e-01 2.47535e+03 3.99187e+00 9.84730e-01
##         XMP         GTP        dATP        dGTP         RNA         DNA
## 2.47930e+01 4.10223e+02 6.01413e+00 3.02581e+00 2.86805e+04 5.17934e+03
##          HX          Xa         Gua          UA         R5P          Pi
## 9.51785e+00 5.05941e+00 5.50638e+00 1.00293e+02 1.80000e+01 1.40000e+03
##
## $BC
##  PRPP   IMP  SAMP   ATP   SAM   Ade   XMP   GTP  dATP  dGTP   RNA   DNA
## FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##    HX    Xa   Gua    UA   R5P    Pi
## FALSE FALSE FALSE FALSE  TRUE  TRUE
##
## $nStates
## [1] 16
##
## $y0
##        PRPP         IMP        SAMP         ATP         SAM         Ade
## 5.01742e+00 9.82634e+01 1.98189e-01 2.47535e+03 3.99187e+00 9.84730e-01
##         XMP         GTP        dATP        dGTP         RNA         DNA
## 2.47930e+01 4.10223e+02 6.01413e+00 3.02581e+00 2.86805e+04 5.17934e+03
```

```
##          HX          Xa         Gua          UA
## 9.51785e+00 5.05941e+00 5.50638e+00 1.00293e+02
```

Make dMod-Model

Extract the various things needed to make a dMod object out of it. We need:

Parameters = c(initial_values, kinetic_parameters)

```
mysummary <- curto %>% summary() # SBMLR:::summary() calculates some other values like the stoichiometr

initial_values <- mysummary$S0

kinetic_pars <- curto[["reactions"]] %>% lapply("[[", "parameters") %>% unname() %>% do.call(c,.) %>% da
duplicated_names <- names(kinetic_pars)[(names(kinetic_pars) %>% duplicated())]
if(length(duplicated_names)>0) warning(paste0("Parameters", paste(duplicated_names, collapse = ", "), "

pars <- c(initial_values, kinetic_pars)
```

ODEs

In dMod you can supply the stoichiometric matrix "S" and the rate vector "reactions". From this you can make an eqnlist-object which you can then use to make the eqnvec, a named vector specifying the right hand side of the ODE.

xdot = f(x,pars) = S %*% reactions

```
S <- mysummary[["incid"]] %>% # get stoichiometric matrix
  t %>% # transpose
  structure(.,dimnames = list(mysummary[["rIDs"]], colnames(.))) %>% # set col- and rownames
  {.[.==0] <- NA # replace zeroes by NA
  . # return the modified matrix
  }

reactions <- data.frame(Description = mysummary[["rIDs"]] , Rate = mysummary[["rLaws"]], S) # get react

f <- reactions %>%
  as.eqnlist(volumes = curto$species %>% sapply(. %>% {.[["compartment"]]})) %>% # turn into eqnlist
  as.eqnvec() # turn into eqnvec
```

Compile the model

Call the function odemodel() does two things: sensitivity equations and C-code ## Sensitivity equations dMod relies heavily on sensitivity equations for gradient computation. Unlike in finite differences, the derivatives are expressed as ODEs themselves:

d/dt(dx_i/dp_j) = df_i/dx %*% dx/dp_j + df_i/dp_j

However, this augments the ODE system and the sensitivity equations are calculated symbolically. This can take extremely long for large equation systems:

N states, M parameters f = N equations dx/dp = N*M equations dx/dx0 = N equations (dx_i/dx0_j = 0)

C-Code

The symbolic equations are then written into C-code and compiled. This can also take quite long for large systems.

These two things greatly restric the number of states and parameters that can be used with our methods.

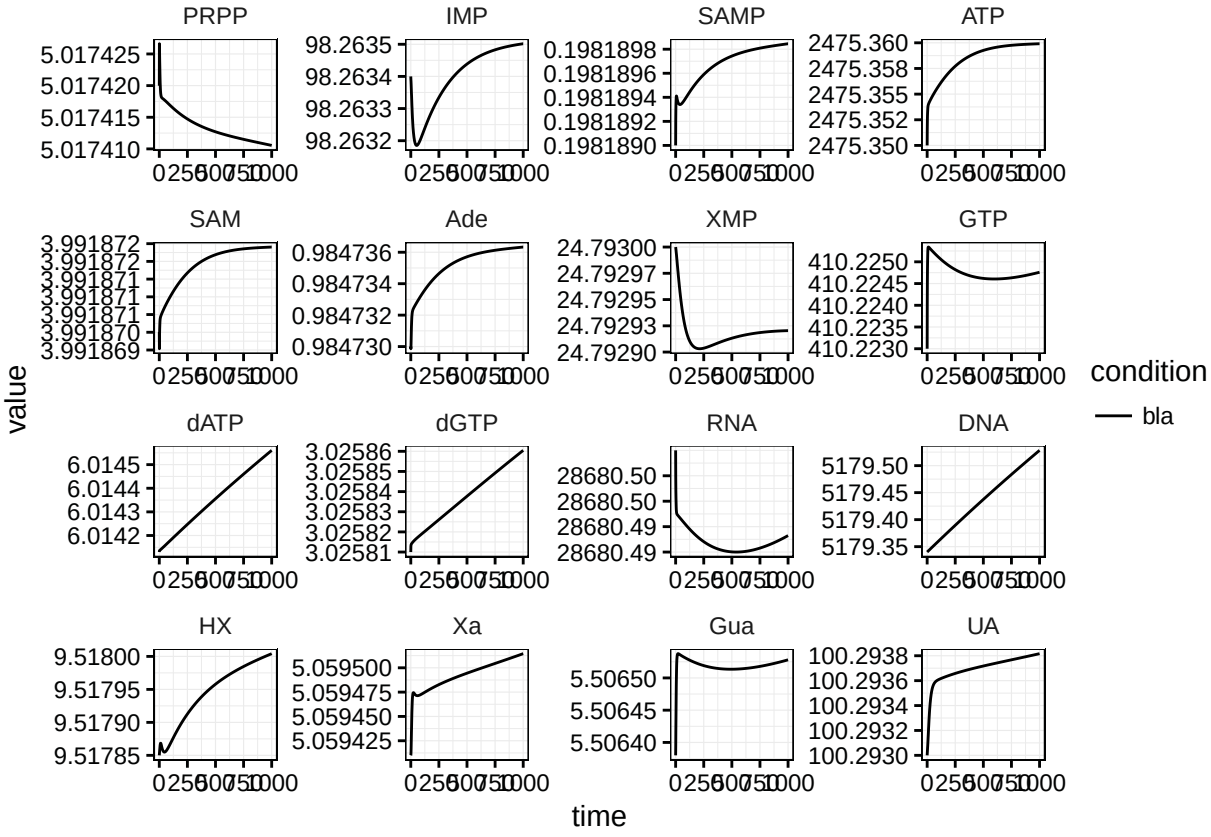
```
myodemodel <- odemodel(f, modelname = "sbml2r") # calculate sensitivity equations and compile C-code
# save(myodemodel, file = "sbml2r.rda")
# load("sbml2r.rda")
```

Compute predictions

```
x <- Xs(myodemodel, condition = "bla") # make prediction funciton

mypred <- x(seq(0,1000,1), pars, deriv = F) #comupte the prediction. If deriv == T, also the derivative

plotPrediction(mypred)
```



simulate data

I did this yesterday in the train, too lazy to delete it, but also too lazy to complete it. For all the estimation procedure it's better to read the vignette and the paper.

```
pars[names(initial_values)] <- initial_values*runif(length(initial_values), 0.5,2) # change some pars t

mydata <- x(times = c(1,5,10,50,1000), pars, deriv = F) %>%
  lapply(function(pred_cond) {
  pred_cond[,-1] <- pred_cond[,-1]*exp(rnorm(length(pred_cond[,-1]), sd = 0.05))
  return(pred_cond)
  }) %>% wide2long() %>%
  mutate(sigma = 0.05*value) %>%
  as.datalist()

plotCombined(x(times = seq(1,2000, 10), pars, deriv = F), data = mydata)
```