Package 'bioPN'

February 19, 2015

Version 1.2.0

Date 2014-03-04

Title Simulation of deterministic and stochastic biochemical reaction networks using Petri Nets

Author Roberto Bertolusso and Marek Kimmel

Maintainer Roberto Bertolusso <rbertolusso@rice.edu>

Description bioPN is a package suited to perform simulation of deterministic and stochastic systems of biochemical reaction
networks.
Models are defined using a subset of Petri Nets, in a way that is close at how chemical reactions are defined.
For deterministic solutions, bioPN creates the associated system of differential equations ``on the fly", and
solves it with a Runge Kutta Dormand Prince 45 explicit algorithm.
For stochastic solutions, bioPN offers variants of Gillespie algorithm, or SSA.
For hybrid deterministic/stochastic,
it employs the Haseltine and Rawlings algorithm, that partitions the system in fast and slow reactions.
bioPN algorithms are developed in C to achieve adequate performance.

NeedsCompilation yes

License GPL (>= 2)

Repository CRAN

Date/Publication 2014-03-04 21:55:07

R topics documented:

a) bioPN package *Simulation of deterministic and stochastic biochemical reaction networks using Petri Nets*

Description

bioPN is a package of C functions that can be used to simulate time-dependent evolution of biochemical reaction networks. The model is defined as a place/transition Petri Net, which is close to how biochemical reactions are defined. The model can be either deterministically solved using an explicit Runge Kutta Dormand Prince 45 method, simulated using four highly optimized variants of the stochastic simulation algorithm, or as a deterministic/stochastic hybrid, according to the Haseltine and Rawlings' algorithm, or using the Partitioned Leaping Algorithm. The library has been optimized for speed and flexibility.

bioPN has been tested only on 64 bits machines, relying on integers of 64 bits. The behavior on 32 bits architectures is untested and not supported.

Details

Package:	bioPN
Type:	Package
Version:	1.2.0
Date:	2014-03-04
License:	GPL (>=2)

Author(s)

Roberto Bertolusso and Marek Kimmel

Maintainer: Roberto Bertolusso <rbertolusso@rice.edu>

References

The biological example presented in the functions is extracted from: Paszek, P. (2007) Modeling stochasticity in gene regulation: characterization in the terms of the underlying distribution function, Bull Math Biol, 69, 1567-1601.

b) Simulation Functions *Simulation of a biochemical system*

Description

These functions simulate a biochemical reacton system parameterized as a Petri Net. `GillespieOptimDirect`, `GillespieDirectGB`, `GibsonBruck`, and `GillespieDirectCR` performs pure stochastic simulations, `RungeKuttaDormandPrince45` a pure deterministic integration, `HaseltineRawlings` a hybrid of the above. `PartitionedLeaping` a dynamic-repartitioning simulation. Multiple runs can be performed at once.

See `init` for a way of defining the model that is close to the way reactions are written.

Usage

```
## Exact stochastic simulation:
GillespieOptimDirect(model, timep, delta=1, runs=1)
GillespieDirectGB(model, timep, delta=1, runs=1)
GibsonBruck(model, timep, delta=1, runs=1)
GillespieDirectCR(model, timep, delta=1, runs=1)

## Pure deterministic:
RungeKuttaDormandPrince45(model, timep, delta=1, ect = 1e-09)

## Hybrid stochastic/deterministic:
HaseltineRawlings(model, timep, delta=1, runs=1, ect = 1e-09)

## Dynamic re-partitioning:
PartitionedLeaping(model, timep, delta=1, runs=1, ect = 1e-09)
```

Arguments

`model`	list containing named elements:
`timep`	It can be either a numeric, indicating for how long (in the same time units as the propensity constants) the process will run, or a functions (R or C), in which case can be used to change the protocol at time intervals. See details.
`delta`	Interval time at which the state will be saved.
`runs`	How many runs will be performed.
`ect`	Precision for the fast reactions.

Details

model is a list containing the following elements:

- model$pre: pre matrix, with as many rows as transitions (reactions), and columns as places (reactants). It has the stoichiometrics of the left sides of the reactions.
- model$post: post matrix, with as many rows as transitions, and columns as places (products). It has the stoichiometrics of the right sides of the reactions.
- model$h: list of propensity constants or functions returning the propensity (with as many elements as transitions).
- model$slow: vector of zeros for slow transitions and ones for fast transitions. Only needed for `HaseltineRawlings`. Ignored otherwise.

- model$M: initial marking (state) of the system.
- model$place: vector with names of the places.
- model$transition: vector with names of the transitions.

Value

The functions return a list with the following elements:

`place`	vector with the names of the places if supplied. If not, the function creates names as follows: P1, P2, ...
`transition`	vector with the names of the transitions if supplied. If not, the function creates names as follows: T1, T2, ...
`dt`	vector containing the discretized times at which the state is saved (according to delta)
`run`	list with as many elements as runs. We will describe the first element, run[[1]], as the rest have exactly the same structure. It is also a list, with the following elements:
`run[[1]]$M`	list with as many elements as places, each of them containing the state of the system sampled according to delta.
`run[[1]]$transitions`	vector with as many elements as transitions, with the total of time each slow reaction fired.
`run[[1]]$tot.transitions`	numeric with the summ of run[[1]]$transitions.

See Also

`init`, `atr`

Examples

```
## bioPN has been tested only on 64 bits machines.
## It may fail in 32 bits architecture.
if (.Machine$sizeof.pointer == 8) {

####### Reaction constants
H <- 10
K <- 6
r <- 0.25
c <- 3
b <- 2
#######

Gi <- 1
Ga <- 2
mRNA <- 3
Protein <- 4

model <- list(
```

```
    pre=matrix(c(1,0,0,0, 0,1,0,0, 0,1,0,0,
        0,0,1,0, 0,0,1,0, 0,0,0,1),
        ncol=4, byrow=TRUE),
    post=matrix(c(0,1,0,0, 1,0,0,0, 0,1,1,0,
        0,0,0,0, 0,0,1,1, 0,0,0,0),
        ncol=4, byrow=TRUE),
    h=list(c, b, H, 1, K, r),
    M=c(1,0,0,0))

timep <- 200
delta <- 1

##############################
## Completely Deterministic ##
##############################
Sim <- RungeKuttaDormandPrince45(model, timep, delta)

## Note, it could also be done as follows
## slow <- rep(0, transitions)
## Sim <- HaseltineRawlings(model, timep, delta, runs = 1)

mRNA.run <- Sim$run[[1]]$M[[mRNA]]
protein.run <- Sim$run[[1]]$M[[Protein]]

## Theoretical results (red lines in following plots)
Mean.mRNA <- c/(c+b)*H
Mean.protein <- Mean.mRNA * K/r

par(mfrow=c(1,2))
par(mar=c(2, 4, 2, 1) + 0.1)
plot(Sim$dt, mRNA.run,type="l", ylab="Mean",main="mRNA")
legend(x="bottom", paste("Deterministic run"))
abline(h=Mean.mRNA,col="red", lwd=1)
plot(Sim$dt, protein.run,type="l", ylab="Mean",main="Protein")
legend(x="bottom", paste("Deterministic run"))
abline(h=Mean.protein,col="red", lwd=1)


runs <- 100 ## Increase to 10000 for better fit
############################
## Completely Stochastic ##
############################
set.seed(19761111)  ## Set a seed (for reproducible results)
Sim <- GillespieOptimDirect(model, timep, delta, runs)

## Note, it could also be done as follows
## slow <- rep(1, transitions)
## Sim <- HaseltineRawlings(model, timep, delta, runs)

mRNA.run <- sapply(Sim$run, function(run) {run$M[[mRNA]]})
protein.run <- sapply(Sim$run, function(run) {run$M[[Protein]]})

## Histograms of protein at different time points.
```

```
par(mfrow=c(2,2))
par(mar=c(2, 4, 2.5, 1) + 0.1)
hist(protein.run[Sim$dt == 1,], main="Protein Distribution at t=1sec")
hist(protein.run[Sim$dt == 2,], main="Protein Distribution at t=2sec")
hist(protein.run[Sim$dt == 10,], main="Protein Distribution at t=10sec")
hist(protein.run[Sim$dt == 200,], main="Protein Distribution at t=200sec")

## Theoretical results (red lines in following plots)
Mean.mRNA <- c/(c+b)*H
Var.mRNA <- b/(c*(1+c+b))*Mean.mRNA^2 + Mean.mRNA
Mean.protein <- Mean.mRNA * K/r
Var.protein <- r*b*(1+c+b+r)/(c*(1+r)*(1+c+b)*(r+c+b))*Mean.protein^2 +
  r/(1+r)*Mean.protein^2/Mean.mRNA + Mean.protein

if (runs > 1 ) {
  par(mfrow=c(2,2))
} else {
  par(mfrow=c(1,2))
}
par(mar=c(2, 4, 2, 1) + 0.1)
plot(Sim$dt, apply(mRNA.run,1,function(tpt) {mean(tpt)}),type="l", ylab="Mean",main="mRNA")
legend(x="bottom", paste("Gene, mRNA and Protein Stochastic\nRuns :", runs))
abline(h=Mean.mRNA,col="red", lwd=1)
plot(Sim$dt, apply(protein.run,1,function(tpt) {mean(tpt)}),type="l", ylab="Mean",main="Protein")
legend(x="bottom", paste("Gene, mRNA and Protein Stochastic\nRuns :", runs))
abline(h=Mean.protein,col="red", lwd=1)
if (runs > 1 ) {
  par(mar=c(2, 4, 0, 1) + 0.1)
  plot(Sim$dt, apply(mRNA.run,1,function(tpt) {var(tpt)}),type="l", ylab="Var")
  abline(h=Var.mRNA,col="red", lwd=1)
  plot(Sim$dt, apply(protein.run,1,function(tpt) {var(tpt)}),type="l", ylab="Var")
  abline(h=Var.protein,col="red", lwd=1)
}


#########################################################################
## Hybrid: mRNA and protein fast, gene activation/inactivation slow ##
#########################################################################
model$slow <- c(1,1,0,0,0,0)

Sim <- HaseltineRawlings(model, timep, delta, runs)

mRNA.run <- sapply(Sim$run, function(run) {run$M[[mRNA]]})
protein.run <- sapply(Sim$run, function(run) {run$M[[Protein]]})

Mean.mRNA <- c/(c+b)*H
Var.mRNA <- b/(c*(1+c+b))*Mean.mRNA^2
Mean.protein <- Mean.mRNA * K/r
Var.protein <- r*b*(1+c+b+r)/(c*(1+r)*(1+c+b)*(r+c+b))*Mean.protein^2

if (runs > 1 ) {
  par(mfrow=c(2,2))
} else {
```

```
  par(mfrow=c(1,2))
}
par(mar=c(2, 4, 2, 1) + 0.1)
plot(Sim$dt, apply(mRNA.run,1,function(tpt) {mean(tpt)}),type="l", ylab="Mean",main="mRNA")
legend(x="bottom", paste("Only Gene Stochastic\nRuns :", runs))
abline(h=Mean.mRNA,col="red", lwd=1)
plot(Sim$dt, apply(protein.run,1,function(tpt) {mean(tpt)}),type="l", ylab="Mean",main="Protein")
legend(x="bottom", paste("Only Gene Stochastic\nRuns :", runs))
abline(h=Mean.protein,col="red", lwd=1)
if (runs > 1 ) {
  par(mar=c(2, 4, 0, 1) + 0.1)
  plot(Sim$dt, apply(mRNA.run,1,function(tpt) {var(tpt)}),type="l", ylab="Var")
  abline(h=Var.mRNA,col="red", lwd=1)
  plot(Sim$dt, apply(protein.run,1,function(tpt) {var(tpt)}),type="l", ylab="Var")
  abline(h=Var.protein,col="red", lwd=1)
}
}
```

`c) Model Definition` *Helper functions for model definition*

Description

These functions are used to define models. They become more useful as the model has more places and transitions, as pre and post are sparse matrices so their direct manipulation may be error prone. See example of use below.

Usage

```
init(place)
atr(trans.name=NULL)
load.cfn(place, code)
unload.cfns()
```

Arguments

`place`	Places
`trans.name`	Name of the transition (reaction)
`code`	C code that returns the propensity

Details

Function `init` accesses the frame of the calling function, creating variables with the names "model", "L", "R", and "h", that are considered reserved to bioPN. It also creates a variable for each element in the place vector submitted to the function `init`. Function `atr` creates a variable for each transition name sent. `load.cfn` and `unload.cfns` are used on cases where the transitions are of a special form, and a C function wants to be used to compute it for increase performance.

Value

The functions do not return values.

See Also

GillespieOptimDirect, HaseltineRawlings

Examples

```
## bioPN has been tested only on 64 bits machines.
## It may fail in 32 bits architecture.
if (.Machine$sizeof.pointer == 8) {

####### Constants definition (convenient but not required)
H <- 10
K <- 6
r <- 0.25
c <- 3
b <- 2
#######

place <- c( "Gi", "Ga", "mRNA", "Protein")

## WARNING: function init() accesses the frame
##          of the calling function, creating variables
##          with the names "model", "L", "R", and "h",
##          that are considered reserved to bioPN.
##          It also creates a variable for each element
##          in the place vector submitted to the function
##          init(). Function atr() creates a variable
##          for each transition name sent.

####### Initialization
init(place)

####### Start of model definition

## Gi -> Ga
h <- c
L[Gi] <- 1
R[Ga] <- 1
atr("gene_activation") ## Add this reaction

## Ga -> Gi
h <- b
L[Ga] <- 1
R[Gi] <- 1
atr("gene_inactivation")

## Ga -> Ga + mRNA
h <- H
L[Ga] <- 1
```

```
R[Ga] <- 1; R[mRNA] <- 1
atr("transcription")

## mRNA -> mRNA + Protein
h <- K
L[mRNA] <- 1
R[mRNA] <- 1; R[Protein] <- 1
atr("mRNA_degradation")

## mRNA -> 0
h <- 1
L[mRNA] <- 1
atr("translation")

## Protein -> 0
h <- r
L[Protein] <- 1
atr("protein_degradation")

####### End of model definition

model$M=rep(0, model$places)
model$M[Gi] <- 1

timep <- 200
delta <- 1

## Completely Deterministic
Sim <- RungeKuttaDormandPrince45(model, timep, delta)

runs <- 100

## Completely Stochastic
set.seed(19761111)  ## Set a seed (for reproducible results)
Sim <- GillespieOptimDirect(model, timep, delta, runs)

## Hybrid run
model$slow <- rep(0, model$transitions)
model$slow[c(gene_activation, gene_inactivation)] <- 1

set.seed(19761111)  ## Set a seed (for reproducible results)
Sim <- HaseltineRawlings(model, timep, delta, runs)
}
```

Index