RKappa ABC complex Examples

Anatoly Sorokin

Package version 1.0.140923

Contents

1 Overview

Dynamic modelling of biological processes is now established as a powerful tool for revealing the systems-level behaviour emerging from the interaction of molecular components. Modelling techniques based on a range mathematical grounds have been introduced over the past century, including kinetic modelling, deterministic and stochastic Petri nets, logical Boolean modelling, etc. The choice of the modelling approach generally depends upon system size, complexity, level of kinetic detail available and expected outcome. However, for a given model building task, there is no guarantee that a sufficient number of parameters are known well enough to approach biological plausibility or to ensure that the resulting simulation will be computationally tractable.

A relatively new modelling approach – rule-based modelling – is one of several developed to deal with combinatorial complexity emerging in multicomponent multistate systems [1]. These have been implemented using several different semantics (Kappa, BioNetGen, StochSim, etc.) and successfully applied to a number of well-described signalling pathways [2-4]. Rule-based modelling enables representation, simulation and analysis of the behaviour of large-scale systems where knowledge of exact mechanisms and parameters is limited. These features make it very appealing to a wide variety of biological modelling problems [5].

The process of building and analysing a dynamic model generally consists of the following essential steps: model assembling, model simulation, analysis of the results and model revision. As a rule, the whole process is highly iterative, therefore, an general–purpose infrastructure that supports all the steps described above would be desirable.

Indeed, for other widely used modeling techniques, such as ODE solving, a number of effective infrastructures (toolboxes) have been developed and subsequently proven their value such as COPASI, SBTOOLBOX2, SBML-SAT, SBML-PET, PottersWheel, etc [8-12]. As an example, SBTOOLBOX2 is based around the SUNDIALS simulating engine and includes a library of Matlab scripts that support model development, model simulation, fitting of models to experimental results, parameter estimation and analysis of results, including the important options for sensitivity and identifiability analysis [10]. The SBML-SAT toolbox provides the Matlab platform for local and global sensitivity analysis [11].

For the relatively new rule-based techniques, such infrastructure is sparse in its coverage. For example, a Matlab-based library is available for BioNetGen, enabling parameter scanning, visualization and analysis of simulation results [13].

What is clearly needed is a method that facilitates the development and analysis of rule-based models within a mature statistically empowered framework. Here we present the R-Kappa package that embodies this need in the widely available statistical package R and demonstrate its effectiveness through its application to Global Sensitivity Analysis (GSA).

In addition to traditional GSA that we call here "parallel" for simplicity, we have introduced a computational experimental setup based upon the distinctive compositionality feature of rule-based models, which was named "concurrent" sensitivity.

2 ABC binding toy model description

2.1 Basic model

Here we illustrate the concept with small toy model of interaction of three agents A, B and C. The kappa model is in Listing 1 and the graphical representation of interactions between agents are shown on the Fig. 1. According them B can bind either A or C, but not both (lines 19, 21 on the Listing 1). We are going to calculate sensitivity of AB fraction of B agent.

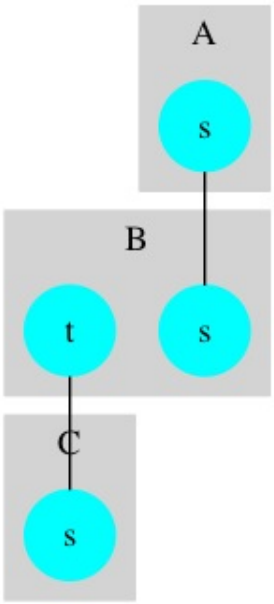

Figure 1: Contact map for the model from Listing 1

2.2 Model split

The key feature of RKappa simulation strategy is ability to run simulation of different parameter sets in parallel and combine their results after completion. It is possible because in any model we are clearly designate parts that depends upon parameters varied in the experiment either directly or by, for example, storing results of simulation that could be mixed between different parameter sets. So we separate three parts in the model:

constant part is the part of the model, which neither depends on nor modified by the application of the new

Listing 1 The toy kappa model of B interacting with A and C.

```
#Agent definition
%agent: A(s)
%agent: B(s,t)
%agent: C(s)

# Reaction kinetic parameters
%var: 'vol' 0.1
%var: 'BDB' 0.0005
%var: 'BRK' 0.1

# Variables definition
%var: 'nA' 1000 * 'vol'
%var: 'nB' 1000 * 'vol'
%var: 'nC' 1000 * 'vol'
%var: 'BND' 'BDB' / 'vol' # Bimolecular reaction rate depends upon volume

# Reaction rules
'assocAB' A(s), B(s,t) -> A(s!0), B(s!0,t) @ 'BND'
'dissocAB' A(s!0), B(s!0) -> A(s), B(s) @ 'BRK'
'assocBC' B(t,s), C(s) -> B(t!0,s), C(s!0) @ 'BND'
'dissocBC' B(t!0), C(s!0) -> B(t), C(s) @ 'BRK'

# Initial conditions definition
%init: 'nA' A(s)
%init: 'nB' B(s,t)
%init: 'nC' C(s)

# Observables
%obs: 'AB' B(s!_)
%obs: 'BC' B(t!_)
%obs: 'B' B(s,t)
%obs: 'AB fraction' 'AB' / 'B'

# Snapshot definition
%mod: repeat ([E+] > 1) && ([E+] [mod] 1000)=0 do $SNAPSHOT "iABCD" until [false]
```

values to the parameter set. In our toy model the only statement that belongs to that part is the snapshot definition.

parameter part is the definition of variables that is substituted by new parameter value assignment (listing 4).

variable part is the part that while is not modified by the parameter assignment, influenced by the parameter values directly, like reaction rules, or indirectly, like observables (listing 3).

In theory for parallel sensitivity we could keep most of the model in constant part, moving to the variable part only reaction rules and observables, but for concurrent sensitivity, when we are going to merge many models into one metamodel, it is important to keep constant part as small as possible.

3 Project definition. Parallel sensitivity

We will start working with library rkappa by loading it:

```
library(rkappa)


## Loading required package:  randtoolbox
## Loading required package:  rngWELL
## This is randtoolbox.  For overview, type 'help("randtoolbox")'.
## Loading required package:  gdata
## gdata:  read.xls support for 'XLS' (Excel 97-2004) files
## gdata:  ENABLED.
##
## gdata:  read.xls support for 'XLSX' (Excel 2007+) files
## gdata:  ENABLED.
##
## Attaching package:  'gdata'
##
## The following object is masked from 'package:stats':
##
##    nobs
##
## The following object is masked from 'package:utils':
##
##    object.size
##
## Loading required package:  igraph
## Loading required package:  plyr
## Loading required package:  futile.logger
```

The model depends upon three parameters: system volume *vol* (line 8); binding rate constant *BDB* (line **??**); and dissociation rate constant *BRK* (line **??**). For analysis we have to provide boundaries for those parameters. In our case we set boundaries for all parameters so that their value will be always between 10^{-9} and 1:

```
parTable<-data.frame(param=c('vol','BDB','BRK'),Min=1e-9,Max=1.0,stringsAsFactors=FALSE)
```

result is shown below:

	param	Min	Max
1	vol	0.00	1.00
2	BDB	0.00	1.00
3	BRK	0.00	1.00

The key element of the RKappa library is the kproject class. Objects of that class stores all information required for generation of the simulation task. To create new project we need to specify number of parameters:

```
abcProj<-prepareProject(project='abc',
     numSets=50,
     exec.path="~/kasim3/KaSim",
     constantfiles=c('cABC_const.ka'),
     templatefiles=c("cABC_templ.ka"),
     paramfile=c("cABC_param.ka"),
     type='parallel')
```

In this case we are going to simulate model locally so we will generate small number of parameter sets ($numSets = 50$). We already split our original model into three parts: constant part of the model (listing 2), parameter definition (listing 4) and variable part of the model (listing 3).

Listing 2 The constant part of the toy model.

```
# Snapshot definition
%mod: repeat ([E+] > 1) && ([E+] [mod] 1000)=0 do $SNAPSHOT "cABC" until [false]
```

Listing 3 The variable part of the toy model.

```
#Agent definition
%agent: A_-(s)
%agent: B_-(s,t)
%agent: C_-(s)

# Variables definition
%var: 'nA_-' 1000 * 'vol_-'
%var: 'nB_-' 1000 * 'vol_-'
%var: 'nC_-' 1000 * 'vol_-'
%var: 'BND_-' 'MOD_-' / 'vol_-' # Bimolecular reaction rate depends upon volume


# Reaction rules
'assocAB_-' A_-(s), B_-(s,t) -> A_-(s!0), B_-(s!0,t) @ 'BND_-'
'dissocAB_-' A_-(s!0), B_-(s!0) -> A_-(s), B_-(s) @ 'BRK_-'
'assocBC' B_-(t,s), C_-(s) -> B_-(t!0,s), C_-(s!0) @ 'BND_-'
'dissocBC' B_-(t!0), C_-(s!0) -> B_-(t), C_-(s) @ 'BRK_-'

# Initial conditions definition
%init: 'nA_-' A_-(s)
%init: 'nB_-' B_-(s,t)
%init: 'nC_-' C_-(s)

# Observables
%obs: 'AB_-' B_-(s!_)
%obs: 'BC_-' B_-(t!_)
%obs: 'B_-' B_-(s,t)
%obs: 'AB_- fraction' 'AB_-' / 'B_-'
```

Original project template contains scripts for execution on SGE MPI job management system:

```
print(abcProj$shLines)


## $run.sh.templ
##  [1] "#!/bin/bash"
##  [2] "numEv=10"
##  [3] "time=1000"
##  [4] "if [ \"$1\" != \"\" ]; then"
##  [5] "numEv= $1"
##  [6] "echo \"number of events to simulate=$numEv\" "
```

```
##  [7] "fi"
##  [8] "if [ \"$2\" != \"\" ]; then"
##  [9] "time= $2"
## [10] "echo \"number of seconsd to simulate=$time\" "
## [11] "fi"
## [12] ""
## [13] "i=1"
## [14] "echo $i"
## [15] "mkdir -p \"./pset***/try$i\""
## [16] "$KASIM_EXE _- -t $time -p $time -d \"./pset***/try$i\" -make-sim prom.kasim"
## [17] "while [ $i -lt $numEv ]"
## [18] "do"
## [19] "i=$[$i+1]"
## [20] "mkdir -p \"./pset***/try$i\""
## [21] "$KASIM_EXE -t $time -p $time -d \"./pset***/try$i\" -load-sim  ./pset***/try1/prom.kasim"
## [22] "done"
##
## $jobConc.sh
##  [1] "#!/bin/bash"
##  [2] "########################################"
##  [3] "#                                      #"
##  [4] "# SGE MPI job script for ECDF Cluster  #"
##  [5] "#                                      #"
##  [6] "# by ECDF System Team                  #"
##  [7] "# ecdf-systems-team@lists.ed.ac.uk     #"
##  [8] "#                                      #"
##  [9] "########################################"
## [10] ""
## [11] "# Grid Engine options"
## [12] ""
## [13] "#$ -N jsimConc "
## [14] "#$ -cwd"
## [15] "#$ -l h_rt=24:00:00"
## [16] ""
## [17] "# Initialise environment module"
## [18] ""
## [19] "export KASIM_EXE=KKKKKK"
## [20] "N=$SGE_TASK_ID"
## [21] "./runConc.sh "
##
## $job.sh.templ
##  [1] "#!/bin/bash"
##  [2] "########################################"
##  [3] "#                                      #"
##  [4] "# SGE MPI job script for ECDF Cluster  #"
##  [5] "#                                      #"
##  [6] "# by ECDF System Team                  #"
##  [7] "# ecdf-systems-team@lists.ed.ac.uk     #"
##  [8] "#                                      #"
##  [9] "########################################"
## [10] ""
## [11] "# Grid Engine options"
## [12] ""
## [13] "#$ -N jsim "
## [14] "#$ -cwd"
## [15] "#$ -l h_rt=24:00:00"
## [16] ""
## [17] "export KASIM_EXE=KKKKKK"
```

```
## [18] "# Initialise environment module"
## [19] ""
## [20] "N=$SGE_TASK_ID"
## [21] "./run$N.sh "
```

Listing 4 The parameter definition of the toy kappa model .

```
                                                                    1
# Reaction kinetic parameters                                       2
%var: 'vol_-' 0.1                                                   3
%var: 'MOD_-' 0.0005                                                4
%var: 'BRK_-' 0.1                                                   5
```

We are going to change shell templates to be able to run it locally on average Linux or Mac system. There are three shell script templates in the project:

run.sh.templ the script defining some basic steps to evaluate one parameter set. The script repeat simulation of the defined model several times and store results in predefined set of folders. It is this script that defines the simulation engine and it has to be modified for use with different version of KaSim or other simulation engine (NFsim for example).

job.sh.templ the whole job execution script. The template provided with the package is designed for execution with the Sun Grid Engine batch-queuing system and was tested on ECDF Cluster in the Edinburgh University. Execution of the job with other system like HTCondor or in AmazonEC2 will require modification of the template.

jobConc.sh the concurrent sensitivity job execution script. The template provided with the package is designed for execution with the Sun Grid Engine batch-queuing system and was tested on ECDF Cluster in the Edinburgh University. Execution of the job with other system like HTCondor or in AmazonEC2 will require modification of the template.

To optimize our project for local execution we need to modify run.sh.templ and job.sh.templ. In the run.sh.templ we first decrease simulation time (replace 1000 seconds with 1000 events in lines 16 and 21 of the template). Another way to run simulation faster is to decrease number of model evaluation runs. Each parameter set is evaluated several times (by default 10 times) to moderate the influence of stochastic nature of the kappa simulation engine. At the analysis stage we are averaging results for calculation of the sensitivity coefficients. Here with the toy model we are going to degrease number of evaluation runs to two ($numEv = 2$):

```
abcProj$shLines[["run.sh.templ"]][2]<-"numEv=2"
abcProj$shLines[["run.sh.templ"]][16]<-
"$KASIM_EXE _- -e $time -p 100 -d \"./pset***/try$i\" -make-sim prom.kasim"
abcProj$shLines[["run.sh.templ"]][21]<-
"$KASIM_EXE -e $time -p 100 -d \"./pset***/try$i\" -load-sim  ./pset***/try1/prom.kasim"

 abcProj$shLines[["run.sh.templ"]]


##  [1] "#!/bin/bash"
##  [2] "numEv=2"
##  [3] "time=1000"
##  [4] "if [ \"$1\" != \"\" ]; then"
##  [5] "numEv= $1"
##  [6] "echo \"number of events to simulate=$numEv\" "
##  [7] "fi"
##  [8] "if [ \"$2\" != \"\" ]; then"
##  [9] "time= $2"
## [10] "echo \"number of seconsd to simulate=$time\" "
## [11] "fi"
```

```
## [12] ""
## [13] "i=1"
## [14] "echo $i"
## [15] "mkdir -p \"./pset***/try$i\""
## [16] "$KASIM_EXE _- -e $time -p 100 -d \"./pset***/try$i\" -make-sim prom.kasim"
## [17] "while [ $i -lt $numEv ]"
## [18] "do"
## [19] "i=$[$i+1]"
## [20] "mkdir -p \"./pset***/try$i\""
## [21] "$KASIM_EXE -e $time -p 100 -d \"./pset***/try$i\" -load-sim  ./pset***/try1/prom.kasim"
## [22] "done"
```

After modifications made in run.sh.template execution time come down to several seconds and nowe we are ready to run it locally. There is no need for batch-queuing system on the local machine so cannot rely on it in execution all our jobs. So we are going to modify job.sh.templ to make it loop through all jobs and execute them one by one:

```
 abcProj$shLines[["job.sh.templ"]][20]<-"#N=$SGE_TASK_ID"
 abcProj$shLines[["job.sh.templ"]][21]<-"for runN in $(ls run*.sh)"
 abcProj$shLines[["job.sh.templ"]][22]<-"do"
 abcProj$shLines[["job.sh.templ"]][23]<-"echo $runN"
 abcProj$shLines[["job.sh.templ"]][24]<-"./$runN"
 abcProj$shLines[["job.sh.templ"]][25]<-"done"

 abcProj$shLines[["job.sh.templ"]]


##  [1] "#!/bin/bash"
##  [2] "##########################################"
##  [3] "#                                        #"
##  [4] "# SGE MPI job script for ECDF Cluster  #"
##  [5] "#                                        #"
##  [6] "# by ECDF System Team                   #"
##  [7] "# ecdf-systems-team@lists.ed.ac.uk      #"
##  [8] "#                                        #"
##  [9] "##########################################"
## [10] ""
## [11] "# Grid Engine options"
## [12] ""
## [13] "#$ -N jsim "
## [14] "#$ -cwd"
## [15] "#$ -l h_rt=24:00:00"
## [16] ""
## [17] "export KASIM_EXE=KKKKKK"
## [18] "# Initialise environment module"
## [19] ""
## [20] "#N=$SGE_TASK_ID"
## [21] "for runN in $(ls run*.sh)"
## [22] "do"
## [23] "echo $runN"
## [24] "./$runN"
## [25] "done"
```

To run the project we need to create set of executables in the local or remote machine:

```
write.kproject(abcProj)
```

That command would create folder with name of your project$name and write all required files there. The next step is to execute the simulator with all model parameter sets. call the system command for execution job.sh:

```
system('./abc/job.sh')
```

The results of the simulation could be then loaded to the R system with command:

```
abcObs<-read.observables(abcProj,dir='abc')
```

Now we can plot time series for our observables at different parameter sets (graph is shown on fig 2):

```
library(lattice)
par.settings <- list(superpose.symbol = list(col = c("red",
"blue", "green"), fill = c("red", "blue", "green")), superpose.line = list(col =
c("red", "blue", "green")) )

grep('AB[0-9]+$',names(abcObs$resPar))->ind
tabPlot<-abcObs$resPar[,c(1,ind[1:9])]
t<-cbind(data.frame(time=tabPlot$time),stack(tabPlot,select=-time))
xyplot(values~time|ind,t)
```

The PRCC sensitivity value could be calculated with command:

```
tcs<-timed.concurrent.sensitivity(abcProj$paramSet,
           abcObs$resPar, outName='AB[0-9]+', nboot=0)


## Loading required package:  sensitivity
##
## Attaching package:  'sensitivity'
##
## The following object is masked from 'package:randtoolbox':
##
##    sobol
```

the result is a data.frame with time dependent sensitivity value of selected variable against all parameters. Summary of that data.frame is shown in the table 1. For each observable there are sensitivity coefficient in the column with prefix 'sc', p-value of accepting the null hypothesis that sensitivity value is equal to zero in the column with prefix 'pval', and value of T-statistics in the column with prefix 'T'.

```
xts<-xtable(summary(tcs), label='tab:tcs.sum',
            caption='Summary of timed sensitivity analysis results')
 print(xts, floating.environment = 'sidewaystable')
```

The last thing is to plot the time series for the sensitivity coefficient:

```
xyplot(sc.AB  ~time|param,tcs,type='l',title='abc binding'
       , par.settings = par.settings)
```

We can see that initially when concentration of AB complex is small and fluctuating sensitivity coefficients are close to zero and only whith time it achieves its real value. It is even better seen on the significance plot (fig 4), where we plot the probability that real sensitivity is equals to zero.

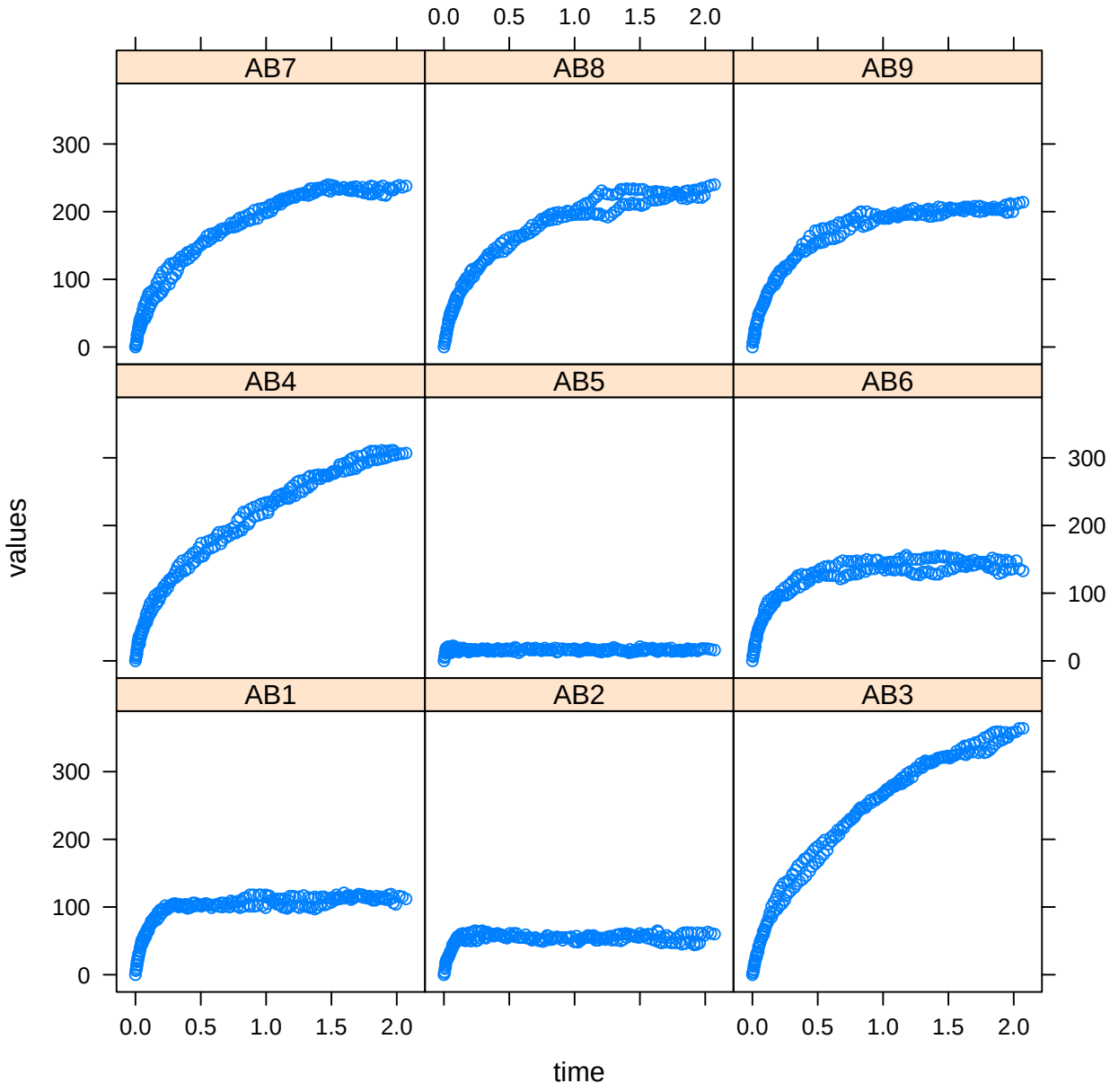

Figure 2: Concentration of AB complex at different parameter sets

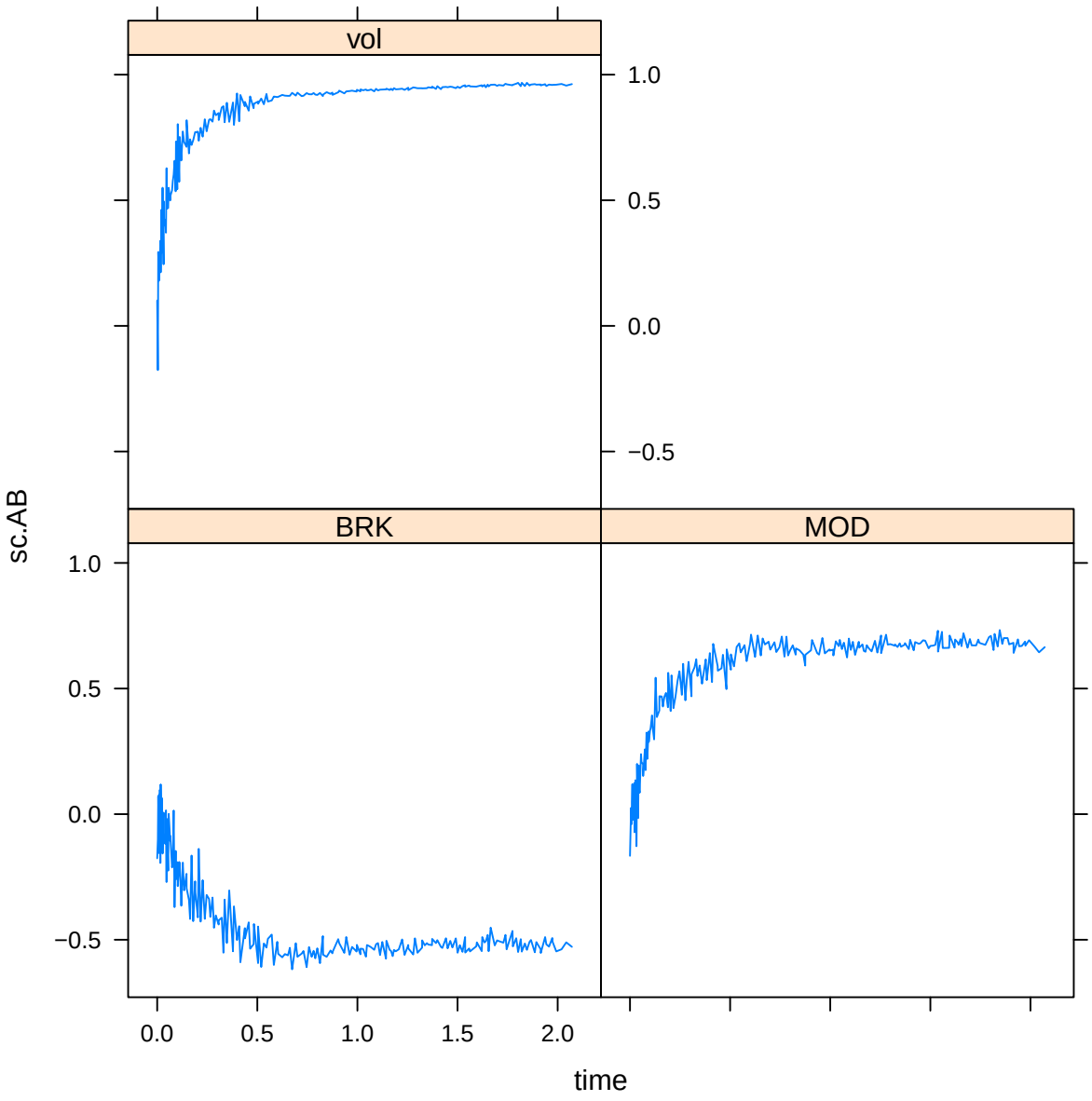

Figure 3: Sensitivity coefficients of three parameters relative to concentration of AB complex

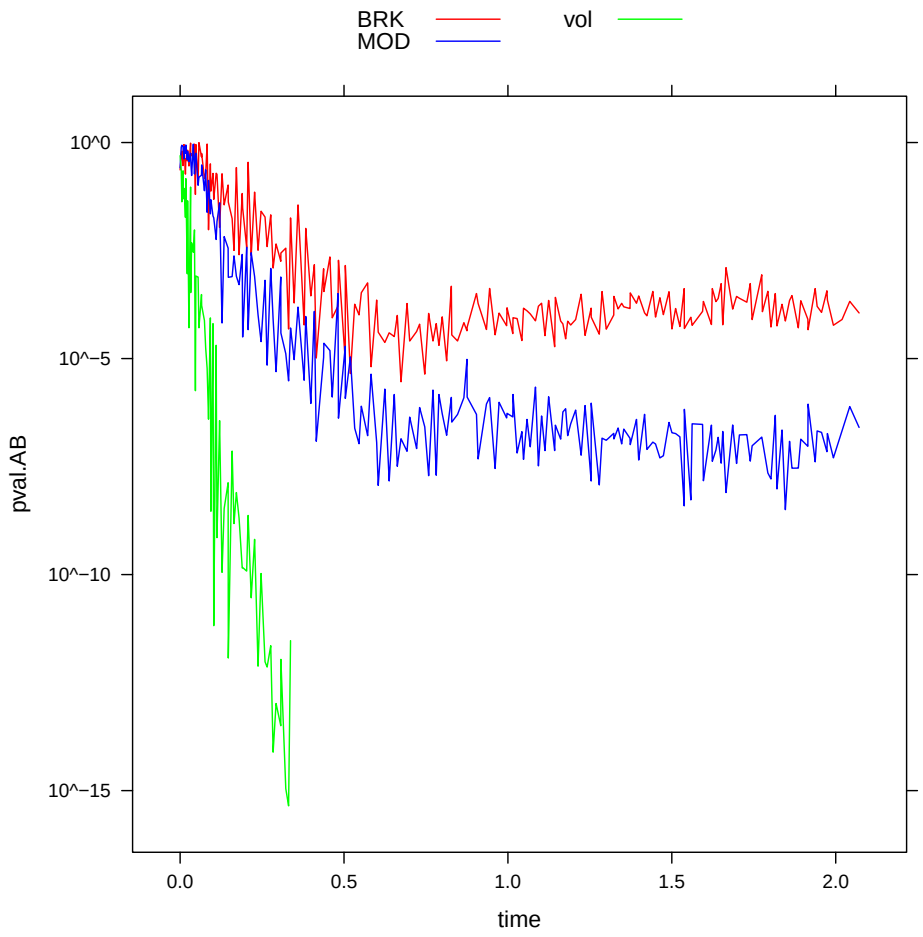

Figure 4: Significance of the sensitivity coefficients of three parameters relative to concentration of AB complex

```
xyplot(pval.AB  ~time,tcs,groups=param,type='l',title='abc binding'
       , par.settings = par.settings
       ,scales=list(y = list(log = TRUE))
       ,auto.key = list(
               text = paste(sort(unique(tcs$param))),
               lines = TRUE,
               points = FALSE,
               columns=2
       )
)
```

It is also possible to estimate the value below which sensitivity coefficient is insignificant at certain value:

```
N<-attr(tcs,'N')
p<-attr(tcs,'p')
threshold<-prccConfidenceInterval(c(1e-3,5e-3,1e-2,5e-2,1e-1),N,p)
xths<-xtable(threshold, label='tab:sig.level',
            caption='Sensitivity value thresholds for different significance levels')
 print(xths, floating.environment = 'table')
```

	time	param	sc.AB_fraction	T.AB_fraction	pval.AB_fraction	sc.AB	T.AB	pval.AB
1	Min. :0.000	BRK:201	Min. :-0.979	Min. :-32.54	Min. :0.0000	Min. :-0.618	Min. :-5.33	Min. :0.0000
2	1st Qu.:0.259	MOD:201	1st Qu.:-0.801	1st Qu.: -9.08	1st Qu.:0.0000	1st Qu.:-0.387	1st Qu.:-2.85	1st Qu.:0.0000
3	Median :0.829	vol:201	Median :-0.504	Median : -3.95	Median :0.0000	Median : 0.617	Median : 5.31	Median :0.0000
4	Mean :0.874		Mean :-0.214	Mean : -3.35	Mean :0.0426	Mean : 0.318	Mean : 5.51	Mean :0.0488
5	3rd Qu.:1.424		3rd Qu.: 0.635	3rd Qu.: 5.58	3rd Qu.:0.0001	3rd Qu.: 0.811	3rd Qu.: 9.41	3rd Qu.:0.0003
6	Max. :2.071		Max. : 0.887	Max. : 13.05	Max. :0.9942	Max. : 0.968	Max. :26.08	Max. :0.9954

Table 1: Summary of timed sensitivity analysis results

	sig.level	lower	upper
1	0.1%	-0.46	0.46
2	0.5%	-0.40	0.40
3	1%	-0.37	0.37
4	5%	-0.28	0.28
5	10%	-0.24	0.24

Table 2: Sensitivity value thresholds for different significance levels