

Tutorial 0: GNU MCSim Introductory

Nan-Hung Hsieh

April 18, 2019

Outline

1. GNU MCSim
2. MCSim-related publication
3. MCSim under R
4. MCSim-related R package
5. Workflow



What is GNU MCSim?



What is GNU MCSim?

GNU MCSim is a simulation and statistical inference tool for algebraic or differential equation systems. Other programs, such as *GNU Octave*, have been created to the same end. Still, most available tools are not optimal for performing computer intensive and sophisticated Monte Carlo analyses.

GNU MCSim was created specifically to this end:

to perform Monte Carlo analyses in an optimized, and easy to maintain environment.

<https://www.gnu.org/software/mcsim/mcsim.html>

GNU MCSim Project

- The project started by Don Maszle and [Frederic Y. Bois](#) in UC Berkeley, 1991.
- First public release in 1993 (straight simulations with Monte Carlo modeling).
- Discussions with Stuart Beal at UCSF School of Pharmacy, led the team to investigate the use of **Markov chain Monte Carlo** techniques for PBPK models' calibration.
- The corresponding code was developed by Maszle, during a project in collaboration with [Andrew Gelman](#), then professor at UC Berkeley Statistics Department.
- Additional code written by Ken Revzan allowed the definition and Bayesian calibration of hierarchical statistical models.
- At the time of these developments (around 1996) those capabilities were unique for a freely distributed, easily accessible, efficient and quite versatile software.

[source](#)

GNU MCSim Project

Published to *Journal of Statistical Software* in 1997
(version 4.2.0)

<https://www.jstatsoft.org/article/view/v002i09>

Authors: Frederic Y. Bois, Don R. Maszle

Title: **MCSim: A Monte Carlo Simulation Program**

Abstract: MCSim consists of two pieces, a model generator and a simulation engine. The model generator, *mod*, was created to facilitate the model maintenance and simulation definition, while keeping execution time fast. Other programs have been created to the same end, the Matlab family of graphical interactive programs being some of the more general and easy to use. Still, many available tools are not optimal for performing time and computer intensive Monte Carlo analysis. MCSim was created specifically to this end: to perform Monte Carlo analysis in a highly optimized, and easy to maintain environment.

Page views:: 7512. **Submitted:** 1997-03-07. **Published:** 1997-11-03.

Version history

February 19th, 2019 - Release of GNU MCSim version 6.1.0.

Version 6.1.0 offers an automatic setting of the inverse-temperature scale in simulated tempering MCMC. This greatly facilitates the uses of this powerful algorithm (convergence is reached faster, **only one chain needs** to be run if inverse-temperature zero is in the scale, **Bayes factors** can be calculated for model choice).

- 6.0.1 (05 May 2018)
- 6.0.0 (24 February 2018)
- 5.6.6 (21 January 2017)
- 5.6.5 (27 February 2016)
- 5.6.4 (30 January 2016)
- 5.6.3 (1 January 2016)
- 5.6.2 (24 December 2015)
- 5.6.1 (21 December 2015)
- 5.6.0 (16 December 2015)
- 5.5.0 (17 March 2013)
- 5.4.0 (18 January 2011)
- 5.3.1 (3 March 2009)
- 5.3.0 (12 January 2009)
- 5.2 beta (29 January 2008)
- 5.1 beta (18 September 2006)
- 5.0.0 (4 January 2005)
- 4.2.0 (15 October 2001)
- 4.1.0 (1 August 1997)
- 4.0.0 (24 March 1997)



Types of Simulation

Simple simulation

- Straight simulations (set parameter values and initial conditions).

Used to: *Model testing when building the model (e.g., mass balance)*

Monte Carlo simulations

- Perform repeated (stochastic) simulations across a randomly sampled region of the model parameter space.

Used to: *Check possible simulation (under given parameter distributions) results before model calibration*

SetPoints simulation

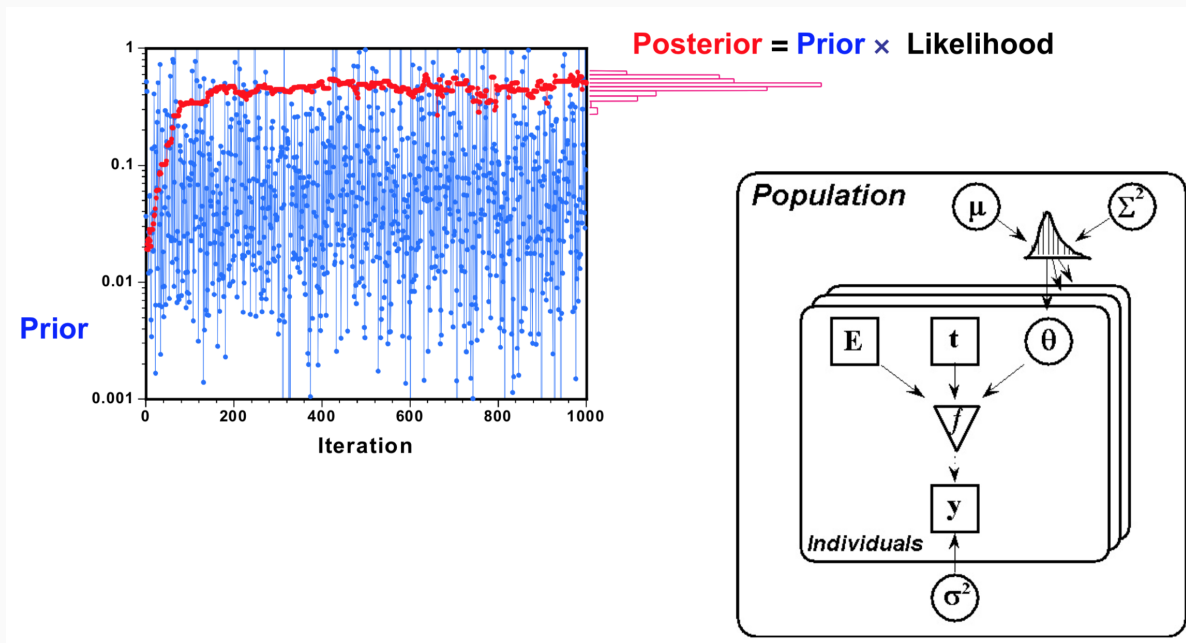
- Solves the model for a series of specified parameter sets. You can create these parameter sets yourself or use the output of a previous Monte Carlo or MCMC simulation.

Used to: *Posterior analysis, Local/global sensitivity analysis*

Types of Simulation

Markov-chain Monte Carlo (MCMC) simulation

- Performs a series of simulations along a Markov chain in the model parameter space.
- They can be used to obtain the Bayesian **posterior** distribution of the model parameters, given a statistical model, **prior** parameter distributions and data for which a **likelihood function** can be computed.
- *GNU MCSim* can handle hierarchical statistical models as well.



Source

Bayesian statistics is a powerful tool, Because...

It gives us the opportunity to understand and quantify the "uncertainty" and "variability" from individuals to **population** through **data** and **model**.



« Surveys are systematically non-representative of the population. So what we do is we adjust for known differences between sample and population. »

ANDREW GELMAN

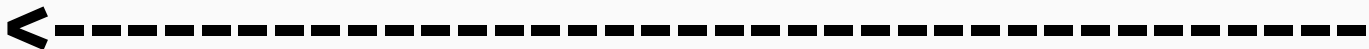


Source

If you have *known* "parameters"



Parameters / Model / Data



If you have *known* "data"

Types of Simulation

"Optimal Design" procedure

Optimizes the number and location of observation times for experimental conditions, in order to minimize the variance of a parameter or an output you specify, given a structural model, a statistical model, and prior distributions for their parameters

Systems Biology Markup Language (SBML)

GNU MCSim can read SBML models, which provides a standard for representing biochemical pathway models. It can code for models of metabolism, cell-signaling, transcription, etc. SBML is maintained since the mid-2000 by an international community of software developers and users.

What GNU MCSim can do in toxicology?

Exchange of ideas between statistics and toxicology

From Statistics to Toxicology

- Bayesian inference for combining prior and data
- Hierarchical models for population variation

From Toxicology to Statistics

- Models for constrained parameters
- Hierarchical prior distributions
- New ideas in understanding and checking models

source

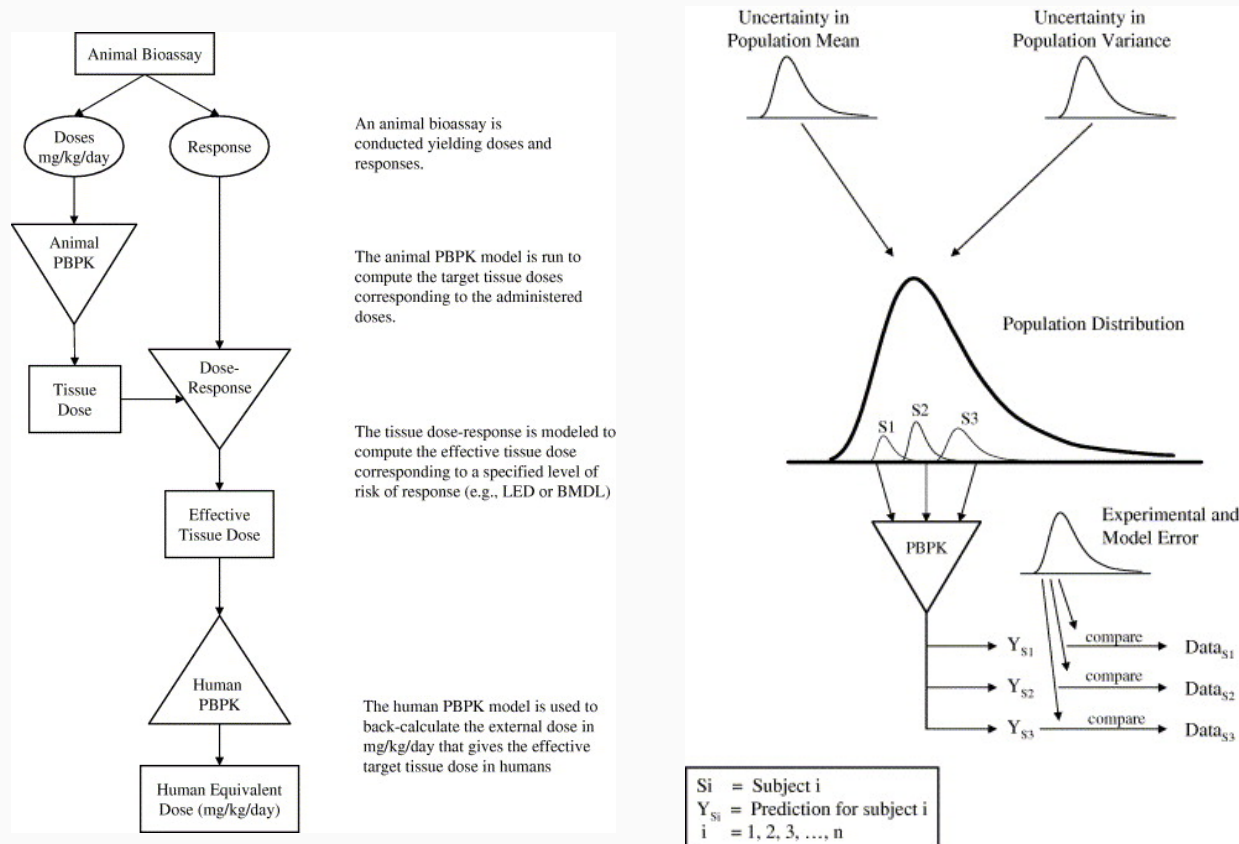
What we need?

1. Physiological pharmacokinetic model
2. Hierarchical population model
3. Prior information
4. Experimental data
5. Bayesian inference
6. Computation
7. Model checking

Related publication

Chiu and Bois (2006) - Revisiting the population toxicokinetics of tetrachloroethylene

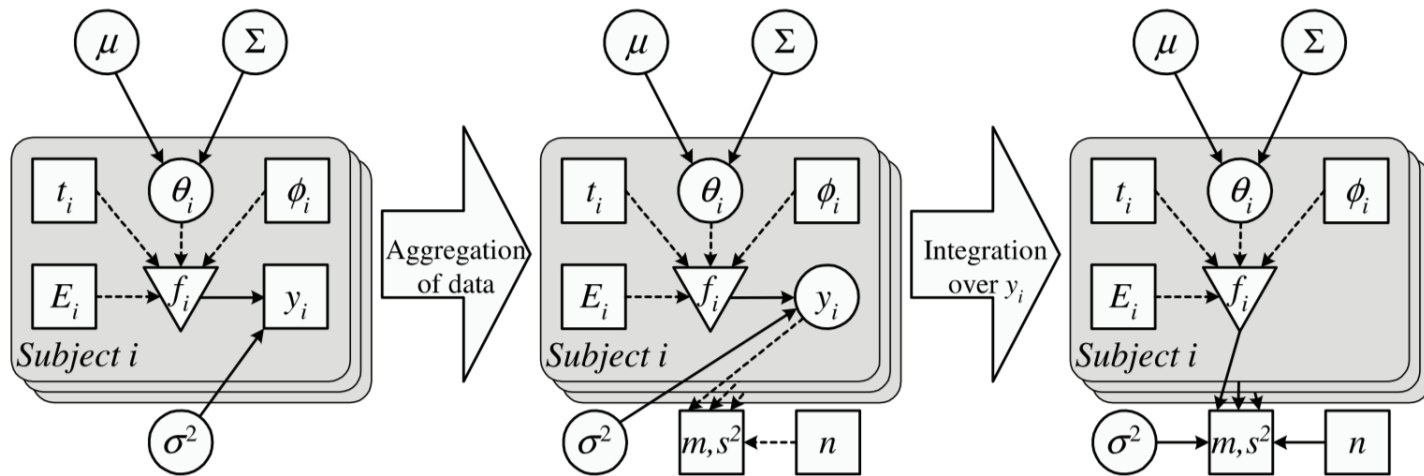
Hack et al. (2006) - Bayesian population analysis of a harmonized physiologically based pharmacokinetic model of trichloroethylene and its metabolites



Related publication

Chiu and Bois (2007) - An approximate method for population toxicokinetic analysis with aggregated data

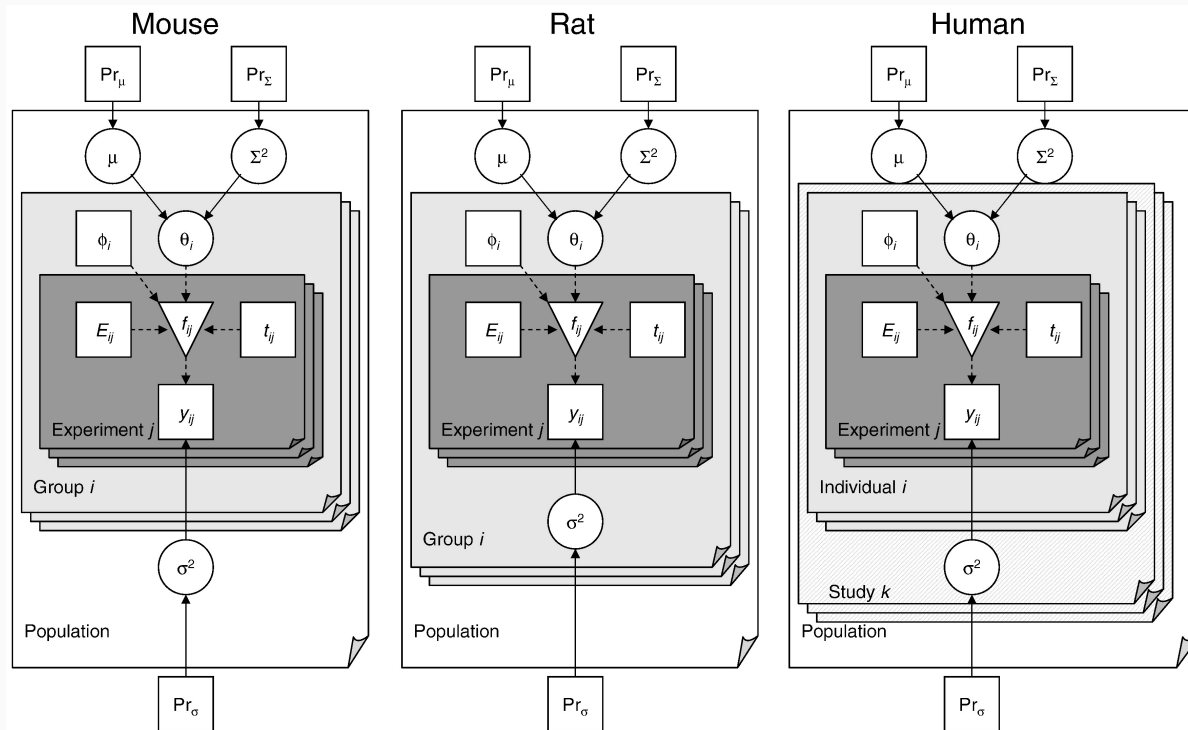
- Applied Hierarchical Bayesian approach to estimate inter-individual variability



Related publication

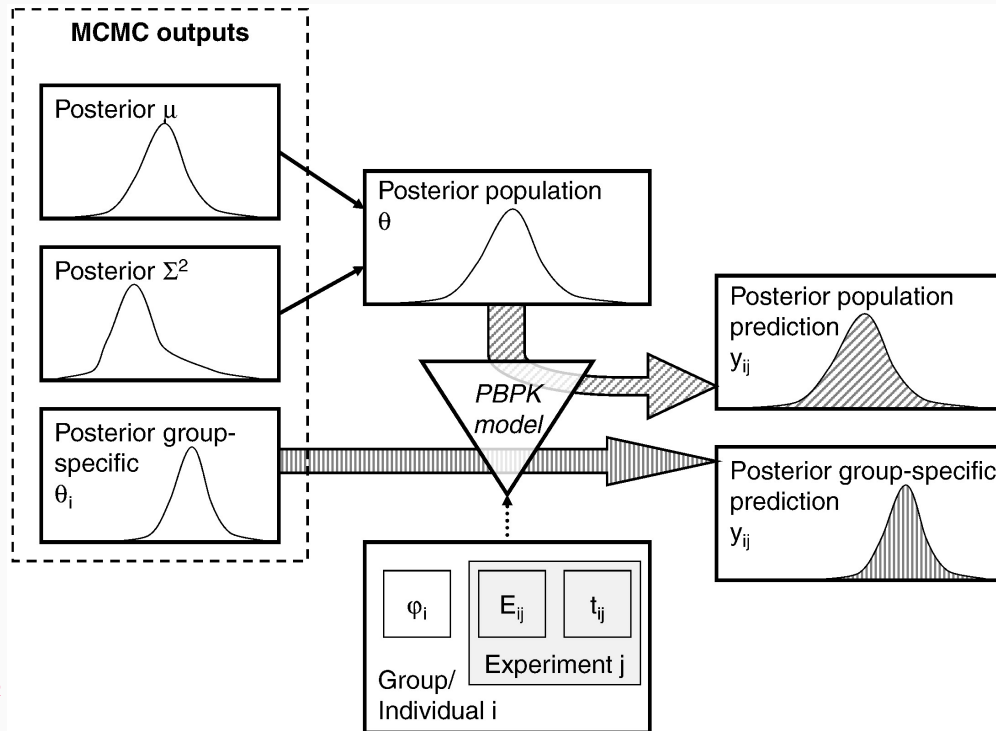
Chiu et al. (2009) - Characterizing uncertainty and population variability in the toxicokinetics of trichloroethylene and metabolites in mice, rats, and humans using an updated database, physiologically based pharmacokinetic (PBPK) model, and Bayesian approach

Chiu and Ginsberg (2011) - Development and evaluation of a harmonized physiologically based pharmacokinetic (PBPK) model for perchloroethylene toxicokinetics in mice, rats, and humans



Related publication

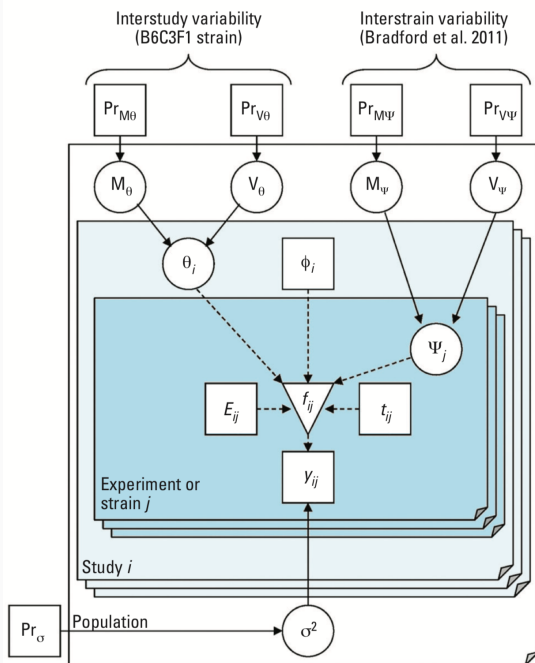
- Characterizing PBPK parameters from individuals to population
- Evaluating population **variability** and parameter **uncertainty**
- Cross-species comparisons of metabolism in **risk assessment**



Related publication from our group

Inter-strain & inter-individual variability

Chiu et al. (2014) - Physiologically-Based Pharmacokinetic (PBPK) Modeling of Inter-strain Variability in Trichloroethylene Metabolism in the Mouse



	Ratio of 95th percentile/50th percentile individual or strain*	
	Human inter-individual variability	Mouse inter-strain variability
TCE oxidized by P450	1.11 (1.05, 1.22)	1.05 (1.01, 1.27)
Total TCA produced	2.09 (1.81, 2.51)	1.77 (1.36, 2.99)
TCE conj. with GSH	6.61 (3.95, 11.17)	7.12 (3.43, 20.7)

*Median and 95% confidence interval

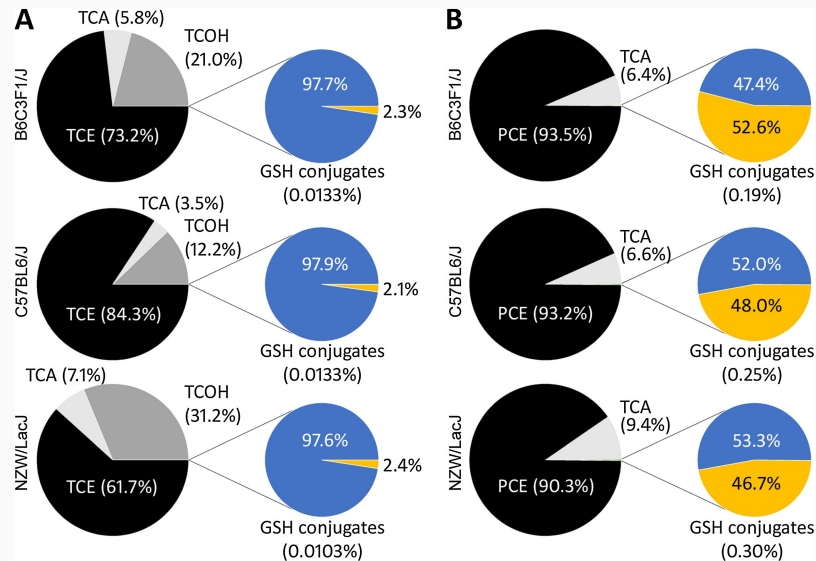
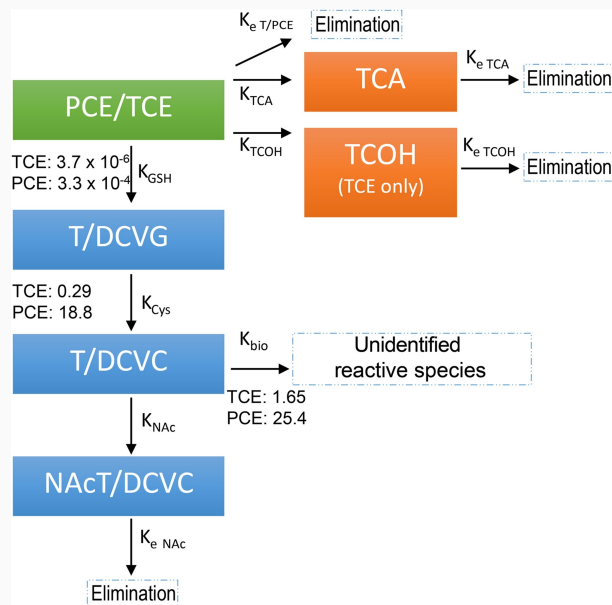
Estimates of mouse population variability from multi-strain experiments are consistent with estimates of human population variability from controlled human exposure studies.

Related publication

Dalaijamts et al. (2018) - Incorporation of the glutathione conjugation pathway in an updated physiologically-based pharmacokinetic model for perchloroethylene in mice

Luo et al. (2018) - Comparative analysis of metabolism of trichloroethylene and tetrachloroethylene among mouse tissues and strains

Luo et al. (Accepted) - Using Collaborative Cross Mouse Population to Fill Data Gaps in Risk Assessment A Case Study of Population-based Analysis of Toxicokinetics and Kidney Toxicodynamics of Tetrachloroethylene



Related publication

U.S. Food and Drug Administration. Enhancing the reliability, efficiency, and usability of Bayesian population PBPK modeling - Create, implement, and evaluate a robust Global Sensitivity Analysis algorithm to reduce PBPK model parameter dimensionality.

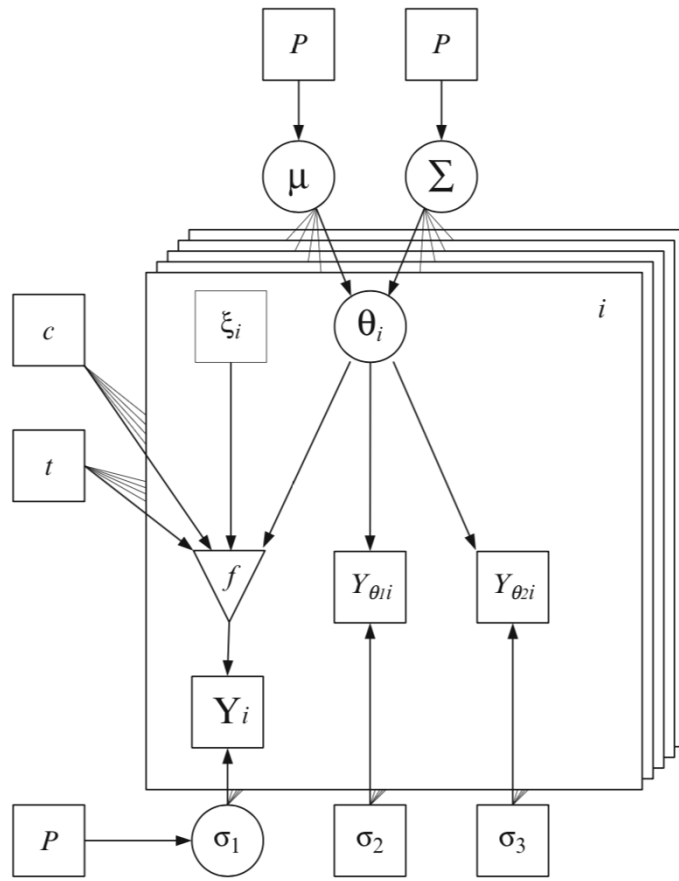
Hsieh et al. (2018) - [Applying a Global Sensitivity Analysis Workflow to Improve the Computational Efficiencies in Physiologically-Based Pharmacokinetic Modeling](#)

Bois et al. (Submitted) - Well tempered MCMC simulations for population pharmacokinetic models

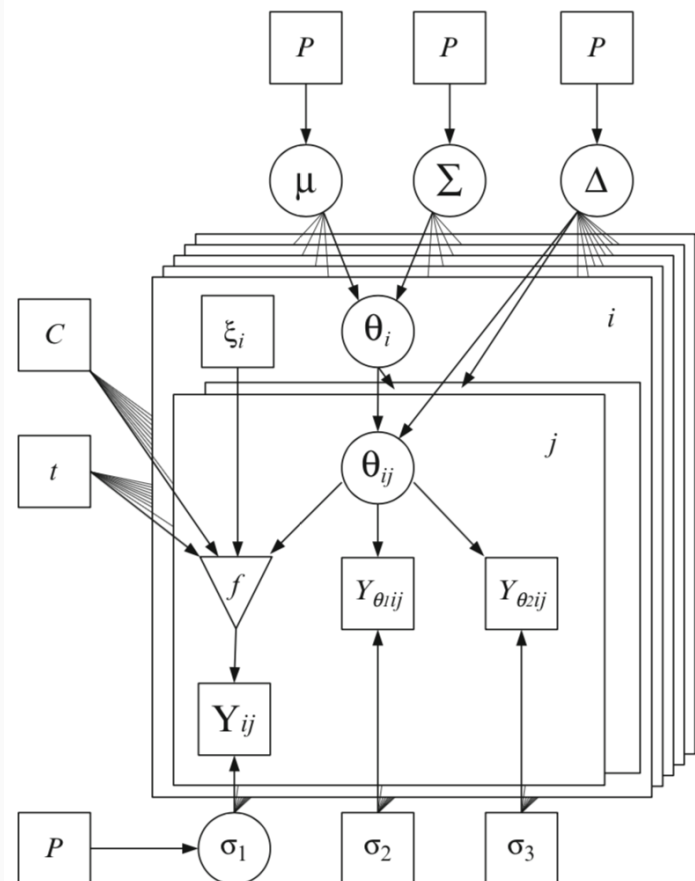
- Based on the latest version 6.1.0

Inter & intra-variability modeling

Inter-individual variability model



Inter- & intra-individual variability model



Why we use GNU MCSim with R(Studio)?

Advantages in MCSim and R

"Free and Open Source Software" under GNU General Public License

- The freedom to run the program as you wish, for any purpose (freedom 0).
- The freedom to study how the program works, and change it so it does your computing as you wish (freedom 1). Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help others (freedom 2).
- The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.



Advantages in MCSim and R

GNU MCSim

Simulation package, which allows you to:

- design and run simulation models (using algebraic or differential equations)
- perform Monte Carlo stochastic simulations
- do Bayesian inference through Markov Chain Monte Carlo simulations
- has faster computing speed than other simulation software/packages (e.g., Asclx, Berkeley Madonna, RStan)

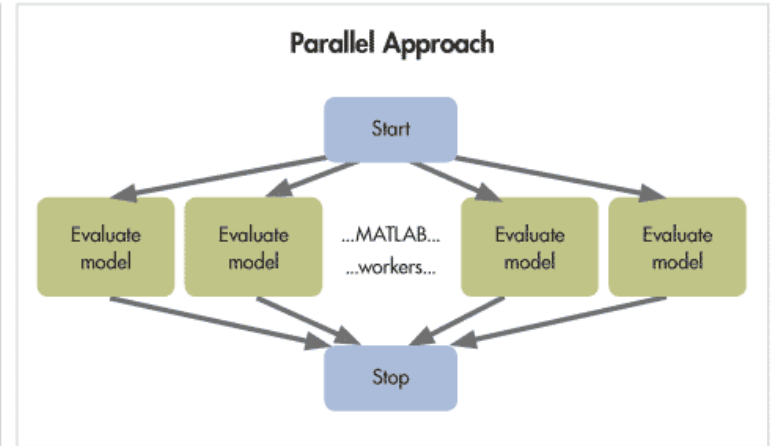
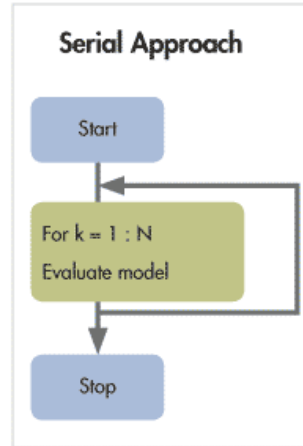
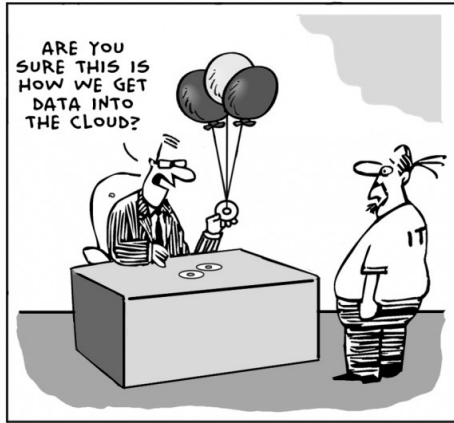
R

Programming language that allows you to:

- conduct statistical analysis (summarization, estimation)
- visualize simulation results
- use various packages to analyze results (e.g., `CODA`, `BOA`, `rstan`)
- perform sensitivity analysis (e.g., `sensitivity`, `pksensi`)
- access community support (e.g., Stack Overflow, R User groups)

Advantages in MCSim and R

"Parallel computing"



- Run multiple MCMC chains with multiple CPUs
- High performance (cloud) computing

source

Population pharmacokinetic reanalysis of a Diazepam PBPK model: a comparison of *Stan* and *GNU MCSim*

Periklis Tsiros, Frederic Y. Bois, Aristides Dokoumetzidis, Georgia Tsiliki, Haralambos Sarimveis

- The aim of this study is to benchmark two Bayesian software tools, namely *Stan* and *GNU MCSim*, that use different Markov chain Monte Carlo (MCMC) methods for the estimation of physiologically based pharmacokinetic (PBPK) model parameters.
- The software tools were applied and compared on the problem of updating the parameters of a Diazepam PBPK model, using time-concentration human data. Both tools produced very good fits at the individual and population levels, despite the fact that *GNU MCSim* is not able to consider multivariate distributions.
- *Stan* outperformed *GNU MCSim* in sampling efficiency, due to its almost uncorrelated sampling.
- However, *GNU MCSim* exhibited much **faster convergence** and performed better in terms of effective samples produced per unit of time.

Journal of Pharmacokinetics and Pharmacodynamics; <https://doi.org/10.1007/s10928-019-09630-x>

Original Paper; First Online: 04 April 2019

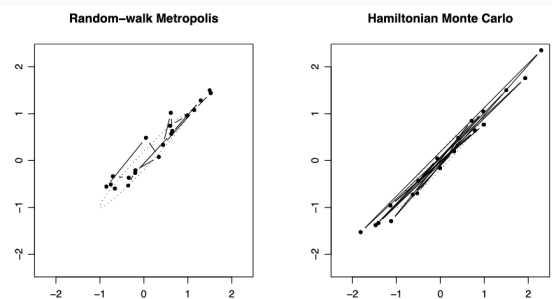


Table 3

Comparison of computational efficiency between *Stan* and *GNU MCSim*

Platform	Time (s)	$\overline{N_{\text{eff}}}$	$\overline{N_{\text{eff}}/s}$	$N_{\text{eff}_{\text{min}}}$	$N_{\text{eff}_{\text{min}}/s}$
<i>Stan multivariate</i>	4850	19,926	4.11	16,240	3.35
<i>Stan univariate</i>	4800	19,935	4.15	15,109	3.15
<i>GNU MCSim</i>	160	2533	15.8	1221	7.63

MCSim-related R packages

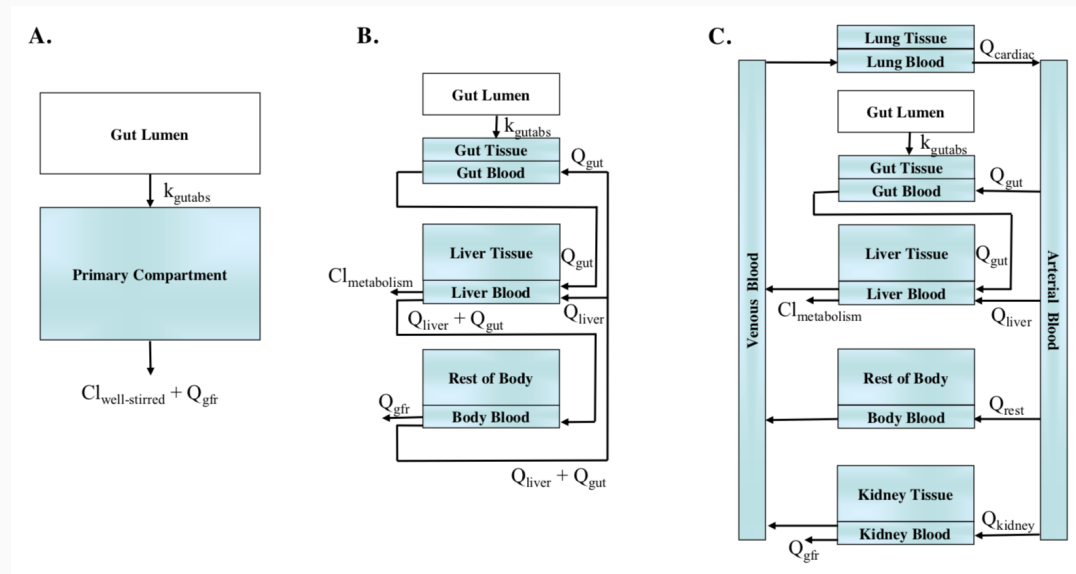
httk: R Package for High-Throughput Toxicokinetics

Robert G. Pearce, R. Woodrow Setzer, Cory L. Strobe, Nisha S. Sipes, John F. Wambaugh

MCSim (Bois and Maszle 1997) was used for converting the model equations into C code, which is used with **deSolve** (Soetaert et al. 2016) in solving each system of equations.

Journal of Statistical Software; <http://dx.doi.org/10.18637/jss.v079.i04>

GNU *MCSim* model code → C code → **deSolve** → Prediction



MCSim-related R packages

pk sensi: R Package for Global Sensitivity Analysis in Pharmacokinetic Modeling

Nan-Hung Hsieh, Brad Reisfeld, Weihsueh A. Chiu

pk sensi implements the global sensitivity analysis workflow to investigate the parameter uncertainty and sensitivity in pharmacokinetic (PK) models, especially the physiologically based pharmacokinetic (PBPK) model with multivariate outputs. The package also provides some functions to check the convergence and sensitivity of model parameters.

CRAN 1.0.1 – 2019-01-29

Two types of model solver

`solve_fun()`

GNU MCSim model code → C code → **deSolve** → Prediction

`solve_mcsim()`

GNU MCSim model code → Prediction

Note: `solve_mcsim()` is faster than `solve_fun()`

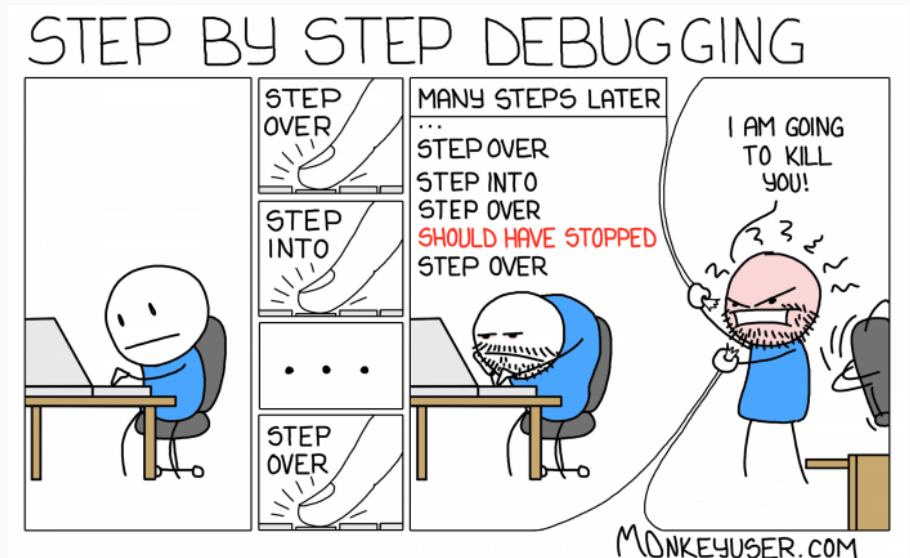
Disadvantages in MCSim and R

Difficult learning curve (command line interface-based)

Requires coding/programming skill

Requires installation of extra program or package

Requires "**debugging**"



Run GNU MCSim in Windows

GNU MCSim is a simulation package, written in C, which can run under different platforms (GNU/Linux, MacOS, and Windows).

However, the basic tools in the Windows system are not available to build and run GNU MCSim. It needs a bit more extra steps to install and execute it.

Therefore...



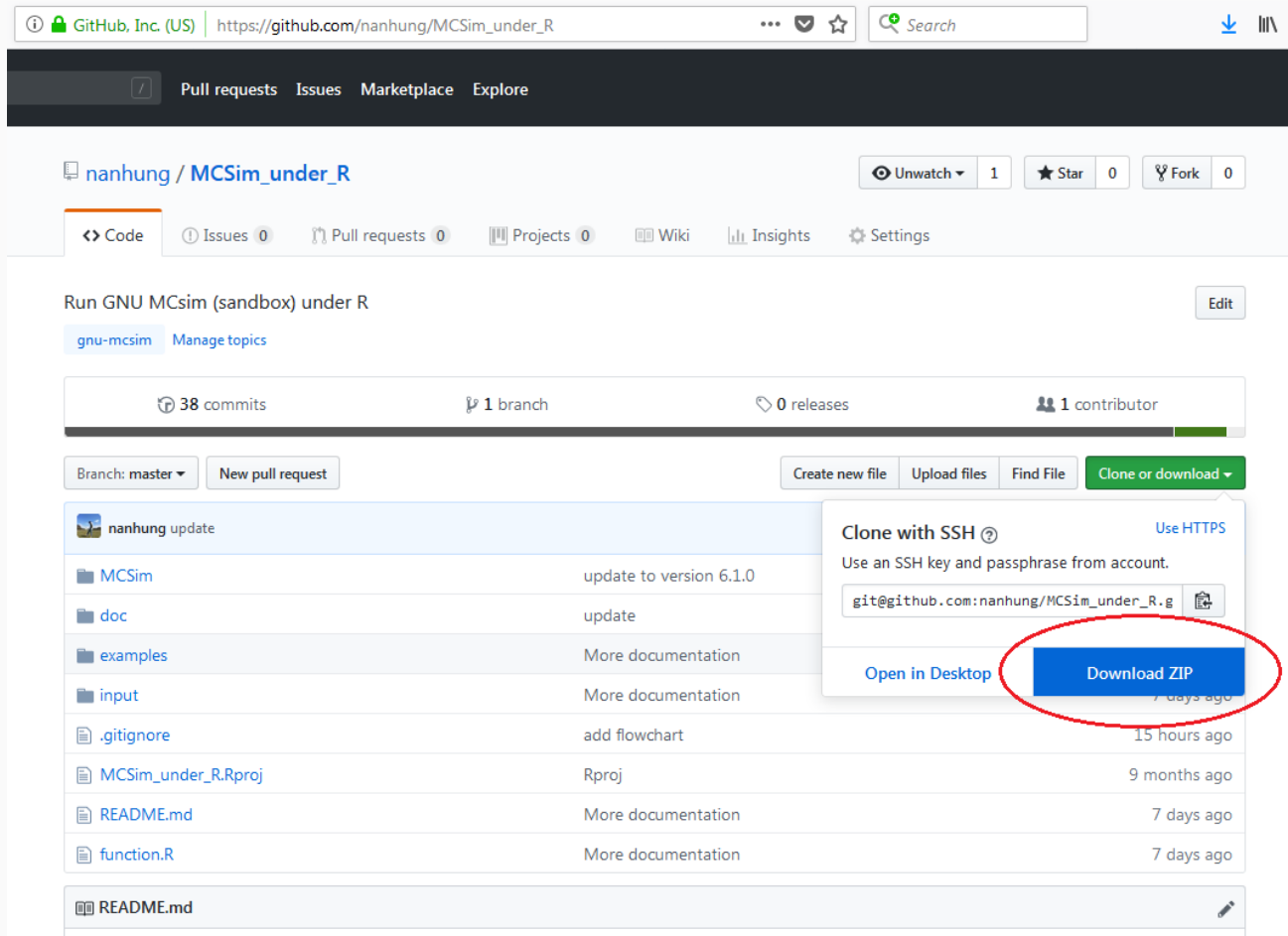
Run GNU MCSim in Windows

- Prof. Weihsueh Chiu proposed a practical method using **minGW** to install it (https://www.gnu.org/software/mcsim/mcsim_with_minGW.pdf).
- Dr. Frederic Bois also provide an alternative method to compile and run GNU MCSim through Rtools. This concept called "MCSim under R".
- Dr. Nan-Hung Hsieh developed an installation function in R package **pksensi** (<https://cran.r-project.org/web/packages/pksensi/index.html>) to help users easily install GNU MCSim through this function across platforms.

Here, we proposed an additional idea to run GNU MCSim in **RStudio**, the integrated development environment (IDE) for R user. This project aims to help beginner write and run GNU MCSim's model code in RStudio IDE.

GNU MCSim under R

An open R project that aims to help beginner (especially Windows user) run GNU MCSim in **RStudio IDE**. All resources are stored in GitHub repo (https://github.com/nanhung/MCSim_under_R).



The screenshot displays the GitHub repository page for 'nanhung / MCSim_under_R'. The repository is on the 'master' branch and has 38 commits, 1 branch, 0 releases, and 1 contributor. The repository structure is as follows:

File/Folder	Description	Last Update
MCSim	update to version 6.1.0	7 days ago
doc	update	7 days ago
examples	More documentation	7 days ago
input	More documentation	7 days ago
.gitignore	add flowchart	15 hours ago
MCSim_under_Rproj	Rproj	9 months ago
README.md	More documentation	7 days ago
function.R	More documentation	7 days ago

The 'Clone or download' dropdown menu is open, showing the following options:

- Clone with SSH (Use an SSH key and passphrase from account. `git@github.com:nanhung/MCSim_under_R.g`)
- Use HTTPS
- Open in Desktop
- Download ZIP (Circled in red)

RStudio

Free and open-source integrated development environment

Powerful and user friendly programming interface

Designed to make it easy to write scripts

Easy to view and interact with the objects

R project with version control (e.g., git)

Support cloud computing <https://rstudio.cloud/>



Studio Cloud ⊗ Your Workspace / MCSim under R ⚙️ ⋮ NH Nan-Hung Hsieh

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function Addins R 3.5.2

Console Terminal Jobs

```

/cloud/project/MCSim_under_R/
under certain conditions; see the GNU General PUBLIC License.

* Using `modeling/linear.model.R' model in file "linear.model.R.c" crea
ted by ./MCSim/mod.exe v6.1.0

Reading experiment 1.

Doing analysis - 1 normal experiment
1
Wrote output file "sim.out"
Done.

> out <- mcsim("linear.model.R", "linear.in.R")

-----

MCSim v6.1.0

Copyright (c) 1993-2019 Free Software Foundation, Inc.

MCSim comes with ABSOLUTELY NO WARRANTY;
This is free software, and you are welcome to redistribute it
under certain conditions; see the GNU General Public License.

* Using `modeling/linear.model.R' model in file "linear.model.R.c" crea
ted by ./MCSim/mod.exe v6.1.0

Reading experiment 1.

Doing analysis - 1 normal experiment
1
Wrote output file "sim.out"
Done.

> plot(out$Time, out$y)
> abline(1,2)
> |

```

Environment History Connections Git

Import Dataset

Global Environment

Data

out 11 obs. of 2 variables

Functions

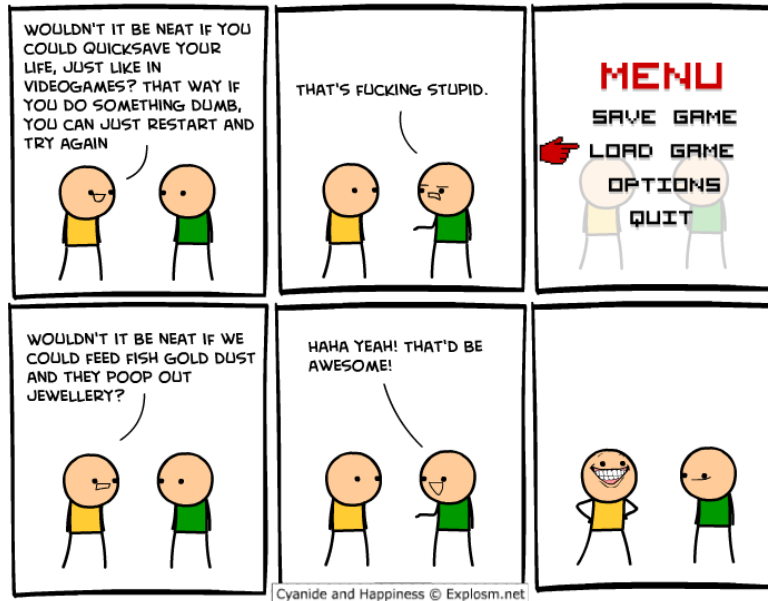
clear	function ()
makemcsim	function (model)
makemod	function ()
mcsim	function (model, input)
mcsim_report	function ()
plotmcsim	function (filename, sim = 1, ...)

Files Plots Packages Help Viewer

Zoom Export

Git

- Manage your code (model, input, R script)
- Transfer your file (through GitHub or GitLab)
- Collaboration work



How to use GNU MCSim with R(Studio)?

Overview

The *GNU MCSim* consists in two pieces, a **model generator** and a **simulation engine**:

The model generator, "**mod**"

- Created to facilitate structural model definition and maintenance, while keeping execution time short. You can code your model using a simplified syntax and use `mod` to translate it to C (`model.c`).

The simulation engine, "**sim**"

- A set of routines which are linked to your model during compilation to produce executable program (`mcsim.model.exe`). After that, you can run simulations of your model under a variety of conditions, specify an associated statistical model, and perform simulations.

Workflow (GNU MCSim under R)

The source code of *GNU MCSim* are put into **"mod"** and **"sim"** folders. In addition, we need two types of files a **"model-file"** (model structure and default parameter values) and an **"input-file"** (run specific parameter values/distributions and/or observation data)

The workflow include three steps:

1. Making GNU MCSim program

Use the files in **"mod"** folder to

- build MCSim program named `mod.exe` (This only needs to be done once)

2. Build model program

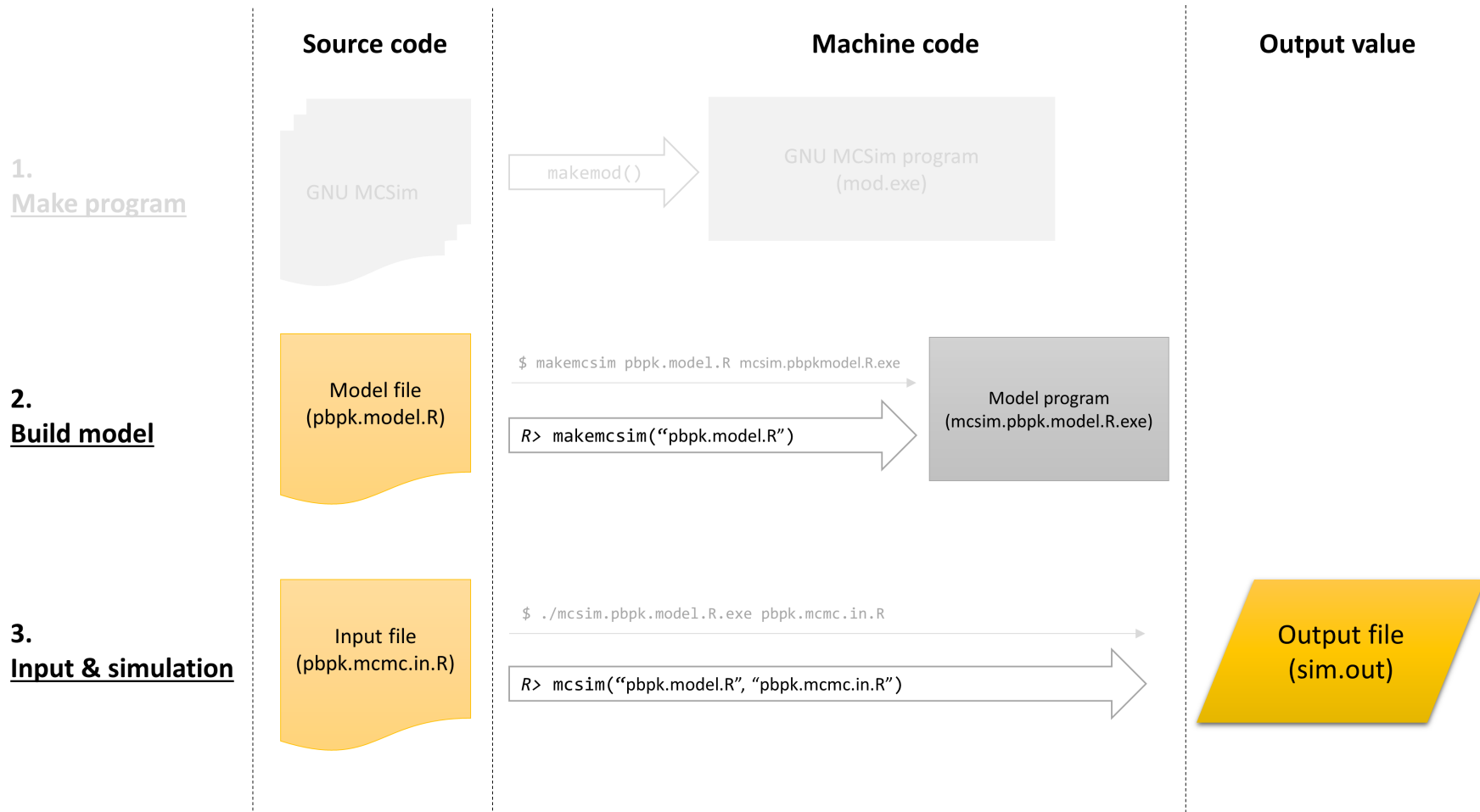
Use the files in **"sim"** folder to

- build model program (*.exe) with (1) `mod.exe` and (2) **"model-file"**

3. Run simulation

- Use model program and **"input-file"** to run simulation and generate result.

Overall Workflow



Syntax of the model description file

```
# Model description file (this is a comment)

<Global variable specifications>

States = {
  <state variables for the model, such as quantity>
}
Outputs = {
  <output variables, such as concentration>
}
Inputs = {
  <input variables, such as exposure dose>
}
Initialize {
  <Equations for initializing or scaling model parameters>
}
Dynamics {
  <Equations for computing derivatives of the state variables>
}
CalcOutputs {
  <Equations for computing output variables>
}
End. # mandatory ending keyword
```

Example of model description file

```
# Unit (V_: liter; A_: mg; k_: /hr)

States = {A_central, A_periph}
Inputs = {Dose}
Outputs = {C_central}

# Structural model parameters
k_12 = 1.02;
k_21 = 0.15;
k_10 = 0.18;
V_central = 58.2;

# Measurement error
Ve_C_central = 1;

# Initialization
Initialize {
  A_central = Dose;
}

# Dynamics
Dynamics {
  # Central compartment quantity
  dt(A_central) = k_21 * A_periph - k_12 * A_central - k_10 * A_central;
  # Peripheral compartment quantity
  dt(A_periph) = k_12 * A_central - k_21 * A_periph;
}

CalcOutputs {
  C_central = A_central / V_central ;
}

End.
```

General syntax

Variable assignments

```
<variable-name> = <constant-value-or-expression> ;
```

Colon conditional assignments

```
<variable-name> = (<test> ? <value-if-true> : <value-if-false>);
```

For example

```
Adjusted_param = (Input_var > 0.0 ? Param * 1.1 : Param);
```

Some other assignments can be found in [GNU MCSim's user manual 5.3.1](#)

Comments on style

For your model file to be **readable** and **understandable**.

- All variable names begin with a capital letter followed by meaningful lower case subscripts.
- Where two subscripts are necessary, they can be separated by an underscore, such as in `PC_fat`.
- Where there is only one subscript an underscore can still be used to increase readability as in `Q_fat`.
- Where two words are used in combination to name one item, they can be separated visually by capitalizing each word, as in `BodyWt`.

Some other contents can be found in [GNU MCSim's user manual 5.3.11](#)

Comments on style (R)

Using = instead of <- for assignment

```
# Good  
x ← 5  
# Bad  
x = 5
```

Without spacing

```
# Good  
average ← mean(feet / 12 + inches, na.rm = TRUE)  
# Bad  
average←mean(feet/12+inches,na.rm=TRUE)
```

Put more than one statement (command) per line

```
# Good  
x ← 1  
x ← x + 1  
# Bad  
x ← 1; x ← x + 1
```

“Good coding style is like using correct punctuation.
You can manage without it, but it sure makes things easier to read.”
— Hadley Wickham, Chief Scientist @RStudio

Using = instead of <- for assignment



Syntax of (simulation) input-file

For the basic simulation

```
# Input-file (text after # are comments)
<Global assignments and specifications>
Simulation {
  <Specifications for first simulation>
}
Simulation {
  <Specifications for second simulation>
}
# Unlimited number of simulation specifications
End. # Mandatory End keyword. Everything after this line is ignored
```

Syntax of (simulation) input-file

For MCMC simulation

```
# Input-file
<Global assignments and specifications>
Level {
  # Up to 10 levels of hierarchy
  Simulation {
    <Specifications and data for first simulation>
  }
  Simulation {
    <Specifications and data for second simulation>
  }
  # Unlimited number of simulation specifications
} # end Level
End. # Mandatory keyword.
```

MCMC() specification

Example of (simulation) input-file

```
# File name: dogoxin.in.R
# ./mcsim.digoxin dogoxin.in.R

Simulation {
  Dose = 509;
  Print (C_central, 0.5, 1, 2, 3, 4, 5, 6, 7, 8, 23);
}

End.
```

Here is the simulation output:

```
mcsim(model = "digoxin.model.R", input = "digoxin.in.R")
```

```
##      Time C_central
## 1    0.5  4.910300
## 2    1.0  2.933400
## 3    2.0  1.383300
## 4    3.0  0.960978
## 5    4.0  0.837285
## 6    5.0  0.792841
## 7    6.0  0.769599
## 8    7.0  0.752196
## 9    8.0  0.736565
## 10  23.0  0.543027
```

Example of (simulation) input-file

```
# File name: dogoxin.mcmc.in.R
# ./mcsim.digoxin dogoxin.mcmc.in.R

MCMC("sim.out","", # name of output and restart file
     "",           # name of data file
     2000,0,       # iterations, simTypeFlag,
     10,2000,     # printing frequency, iters to print
     10101010);  # random seed (default)

Level { # top level
  Distrib(k_12, LogUniform, 0.01, 10);
  Distrib(k_21, LogUniform, 0.01, 10);
  Distrib(k_10, LogUniform, 0.01, 10);
  Distrib(V_central, TruncNormal, 50, 5, 40, 60);
  Distrib(Ve_C_central, LogUniform, 0.01, 0.5); # 10% to 70% residual error

  Likelihood(C_central , Normal, Prediction(C_central) , Ve_C_central);

  Simulation {
    Dose = 509;
    Print (C_central, 0.5, 1, 2, 3, 4, 5, 6, 7, 8, 23);
    Data (C_central, 4.6244, 2.7654, 1.3224, 0.9563, 0.8843, 0.8648, 0.8363, 0.7478, 0.7232, 0.5655)
  }
}
End.
```

```
out ← mcsim(model = "digoxin.model.R", input = "digoxin.mcmc.in.R")
```

```
## * Create 'chk.out' from the last iteration.
```

```
head(out)
```

```
##   iter k_12.1.  k_21.1.  k_10.1. V_central.1. Ve_C_central.1.  LnPrior
## 1    0 0.512176 1.591910 4.578250      47.3817      0.197833 -8.507159
## 2   10 0.663034 1.053850 0.920288      53.8313      0.455026 -7.737837
## 3   20 0.432514 0.157149 0.831279      55.2544      0.329552 -5.241873
## 4   30 0.527456 0.141271 0.772024      59.0265      0.471704 -6.695875
## 5   40 1.052430 0.394778 0.772024      51.4652      0.443793 -6.766683
## 6   50 0.744651 0.499311 0.772024      49.3481      0.405070 -6.529911
##           LnData LnPosterior
## 1 -322.241700 -330.748900
## 2  -5.697733  -13.435570
## 3  -8.420752  -13.662630
## 4  -6.748264  -13.444140
## 5  -4.156501  -10.923180
## 6  -3.172368  -9.702279
```

```
tail(out)
```

```
##   iter k_12.1.  k_21.1.  k_10.1. V_central.1. Ve_C_central.1.  LnPrior
## 196 1950 1.08731 0.182626 0.197338      59.1769      0.0271477 -3.511598
## 197 1960 1.08333 0.171446 0.184968      59.6128      0.0277980 -3.567542
## 198 1970 1.08763 0.171331 0.170108      59.8496      0.0355800 -3.826062
## 199 1980 1.08763 0.172720 0.183829      59.0764      0.0262816 -3.316123
## 200 1990 1.09268 0.176052 0.183829      59.4927      0.0270974 -3.525014
## 201 2000 1.09235 0.172909 0.183829      59.3599      0.0375671 -3.783332
##           LnData LnPosterior
## 196 19.32131    15.80971
## 197 21.13345    17.56590
## 198 20.37592    16.54985
## 199 18.81788    15.50176
## 200 20.96801    17.44300
## 201 20.56532    16.78199
```

Distribution functions

- **InvGamma** (inverse gamma distribution), needs two strictly positive real parameters: the shape and the scale.
- **LogNormal**, takes two real numbers as parameters: the geometric mean (exponential of the mean in log-space) and the geometric standard deviation (exponential, strictly superior to 1, of the standard deviation in log-space).
- **LogUniform** with two shape parameters: the minimum and the maximum of the sampling range (real numbers) in natural space.
- **Normal** takes two real numbers as parameters: the mean and the standard deviation, the latter being strictly positive.
- **Normal_v** is also the normal distribution with the variance instead of the standard deviation as second parameter.
- **TruncNormal** (truncated normal distribution), takes four real parameters: the mean, the standard deviation (strictly positive), the minimum and the maximum.
- **StudentT**, requires three parameters: its number of degrees of freedom (an integer), its mean, and its standard deviation.
- **TruncLogNormal** (truncated lognormal distribution), uses four real numbers: the geometric mean and geometric standard deviation (strictly superior to 1)
- **LogNormal_v**, is the lognormal distribution with the variance (in log space!) instead of the standard deviation as second parameter.

More functions in [GNU MCSim User's Manual](#)

Input functions

These functions can use to different exposure types

- `PerDose()`: # specifies a periodic input of constant

```
PerDose(<magnitude>, <period>, <initial-time>, <exposure-time>);
```

- `PerExp()`: # specifies a periodic exponential input.

```
PerExp(<magnitude>, <period>, <initial-time>, <decay-constant>);
```

- `PerTransit()`: models a delayed input mechanism

```
PerTransit(<magnitude>, <period>, <initial-time-in-period>,  
           <decay-constant>, <number-of-input-compartments>);
```

- `NDoses()`: specifies a number of stepwise inputs of variable magnitude and their starting times

```
NDoses(<n>, <list-of-magnitudes>, <list-of-initial-times>);
```

- `Spikes()`: specifies a number of instantaneous inputs of variable magnitude and their exact times of occurrence.

```
Spikes(<n>, <list-of-magnitudes>, <list-of-times>);
```

Main functions

Here are the R functions that can help you run GNU MCSim more easily. All R functions are defined in `function.R` in MCSim folder.

```
makemcsim(model, deSolve = F)
```

- Preprocessing and compiling the model-file to the executable file as `makemcsim` in GNU MCSim. The `model` assignment is a string giving the name of the model-file (e.g., `"pbpk.model.R"`). The `deSolve` assignment is a logical factor to use **deSolve** package as an ODE solver.

```
mcsim(model, input)
```

- Using the compiled program with the input-file to run the simulation. The `input` assignment is a string giving the name of the input-file (e.g., `"pbpk.in.R"`)
- This function can also automatically compile the model, if you forgot to use `makemcsim()` to create model program.

Example of using deSolve

deSolve

```
library(deSolve)
model ← "digoxin.model.R"
makemcsim(model = model, deSolve = T)
parms ← initParms() # Define parameter value
newParms ← c(parms, Dose = 509) # Define input
Y ← initState(parms = newParms) # Initial condition
times ← c(0, 0.5, 1, 2, 3, 4, 5, 6, 7, 8, 23)
out ← ode(Y, times, func = "derivs", parms = parms,
          dllname = model,
          initfunc = "initmod", nout = 1,
          outnames = "C_central")

out
```

```
##      time A_central A_periph C_central
## 1    0.0 509.00000    0.0000 8.7457045
## 2    0.5 285.78053 188.5568 4.9103185
## 3    1.0 170.72485 283.6368 2.9334166
## 4    2.0  80.50653 352.9739 1.3832736
## 5    3.0  55.92820 365.7309 0.9609657
## 6    4.0  48.72975 363.6315 0.8372809
## 7    5.0  46.14326 357.7117 0.7928394
## 8    6.0  44.79060 350.8890 0.7695980
## 9    7.0  43.77781 343.9332 0.7521961
## 10   8.0  42.86809 337.0456 0.7365651
## 11  23.0  31.60418 248.5552 0.5430271
```

MCSim

```
mcsim("digoxin.model.R", "digoxin.in.R")
```

```
##      Time C_central
## 1    0.5 4.910300
## 2    1.0 2.933400
## 3    2.0 1.383300
## 4    3.0 0.960978
## 5    4.0 0.837285
## 6    5.0 0.792841
## 7    6.0 0.769599
## 8    7.0 0.752196
## 9    8.0 0.736565
## 10  23.0 0.543027
```

Other functions

These functions are used to perform basic setting.

```
set_PATH(PATH)
```

- Detecting, checking, and setting the C compiler in your computer. This process will automatically execute. The default PATH setting is `"c:/Rtools/mingw_32/bin"`.

```
makemod()
```

- Creating MCSim program `mod.exe`.

```
clear()
```

- Removing all executable and output files with extension `.exe`, `.out`, and `.perks` in the working directory. This function is used to ensure that all simulation process can be reproduced without misusing the old version of the file with the same file name. This function will not remove `mod.exe`.

```
report()
```

- Reporting the system setting.

Example: linear modeling

model-file

```
# File name: linear.model.R

Outputs = {y}

# Model Parameters
A = 0; # Default value of intercept
B = 1; # Default value of slope

# Statistical parameter
SD_true = 0;

CalcOutputs {
  y = A + B * t + NormalRandom(0,SD_true);
}

End.
```

input-file

```
# File name: linear.in.R
# $ ./mcsim.linear.model.R.exe linear.in.R

Simulation { # 1 simple simulation

  A = 1; # given value of intercept
  B = 2; # given value of slope
  SD_true = 2; # given SD of noise

  PrintStep (y, 0, 10, 1);
}

END.
```

Example: linear modeling

A simple way to store the results in the output variable and check it,

```
out ← mcsim(model = "linear.model.R", input = "linear.in.R")  
out
```

```
##      Time      y  
## 1      0 -0.415609  
## 2      1 -1.430340  
## 3      2  4.144040  
## 4      3  7.535080  
## 5      4  9.517630  
## 6      5 10.871900  
## 7      6 13.735700  
## 8      7 13.075000  
## 9      8 16.095100  
## 10     9 16.816300  
## 11    10 21.300200
```

Example: linear modeling

If you forgot to assign a variable to store your output, you can use `read.delim()` to read the output file as,

```
out ← read.delim(file = "sim.out", skip = 1)
out
```

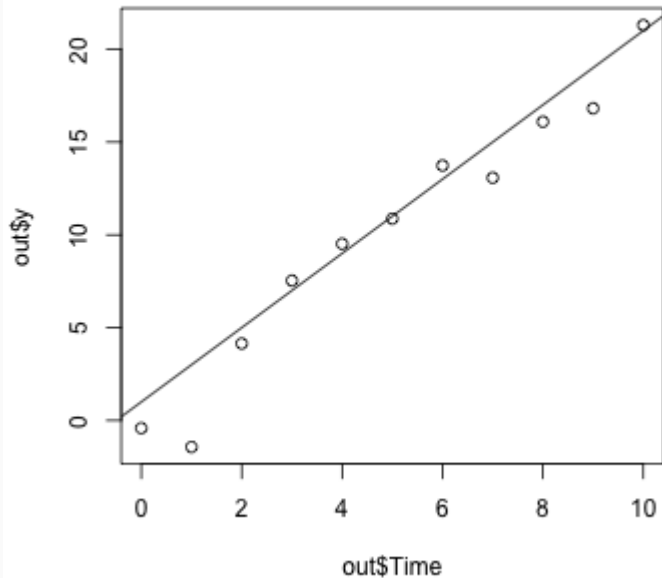
```
##      Time          y
## 1      0 -0.415609
## 2      1 -1.430340
## 3      2  4.144040
## 4      3  7.535080
## 5      4  9.517630
## 6      5 10.871900
## 7      6 13.735700
## 8      7 13.075000
## 9      8 16.095100
## 10     9 16.816300
## 11    10 21.300200
```

Question: why `skip = 1`?

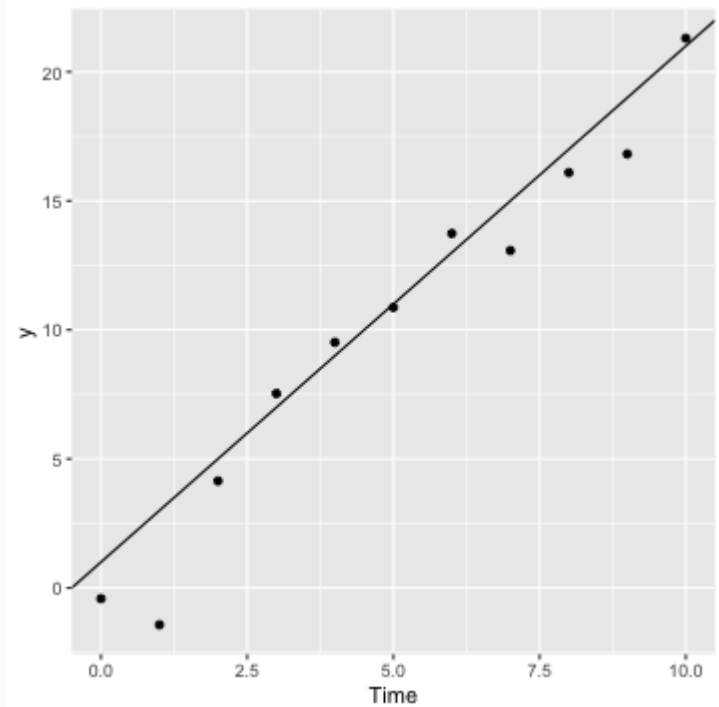
Example: linear modeling

Visualization the simulation result by R base plot and ggplot

```
plot(x=out$Time, y=out$y)  
abline(a=1, b=2)
```



```
library(ggplot2)  
ggplot(out, aes(x=Time, y=y)) +  
  geom_point() +  
  geom_abline(intercept = 1, slope = 2)
```



Additional R packages for analysis

data.table - Fast reading of large output file, especially from MCMC with long iterations and high dimensional parameter space.

tidyverse - Powerful data science toolbox that can use to manipulate (`dplyr`) and plot (`ggplot`) the simulation result.

sensitivity - A collection of functions for factor screening, global sensitivity analysis and reliability sensitivity analysis.

pk sensi - Applying the global sensitivity analysis (eFAST) workflow to investigate the parameter uncertainty and sensitivity in pharmacokinetic (PK) models.

bayesplot - Plotting functions for posterior analysis.

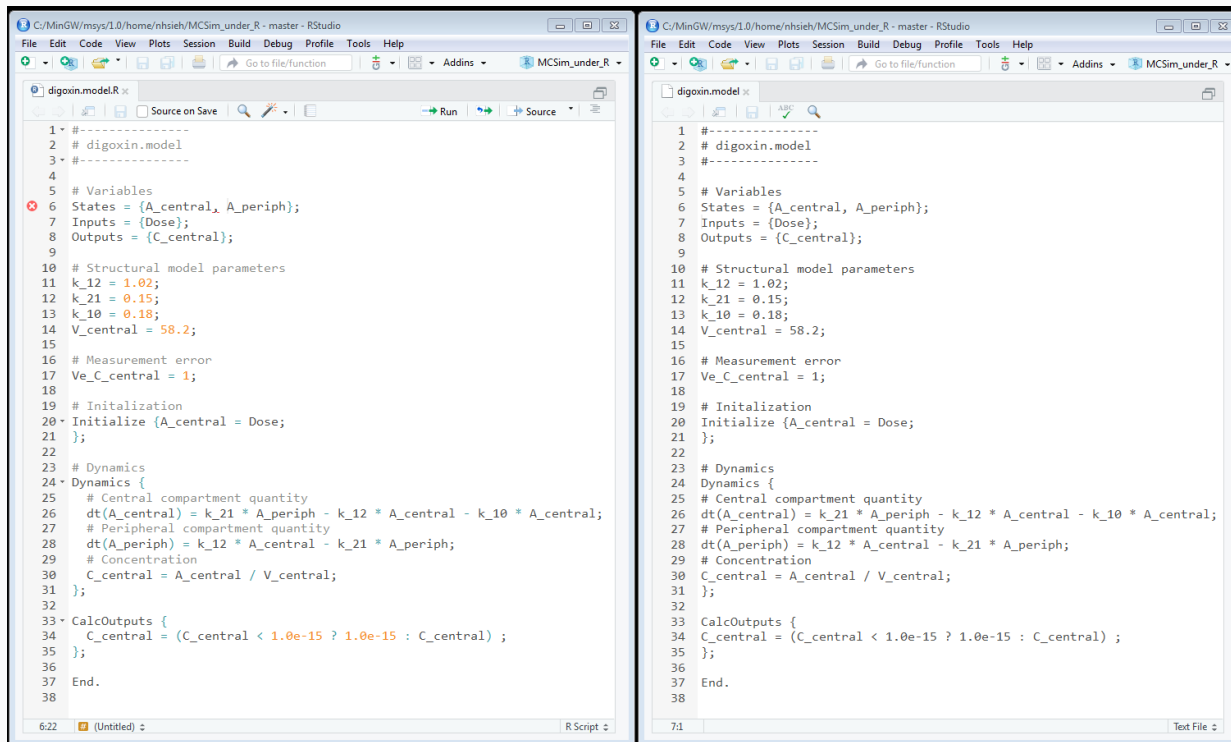
rstan - Diagnose the simulation result from MCMC.

```
# Quick installation
pkgs ← c("bayesplot", "data.table", "pk sensi", "rstan", "sensitivity", "tidyverse")
install.packages(pkgs)
```

Tips of MCSim under R(Studio)

Code edit

Generally, the GNU MCSim used `.model` and `.in` as the extension to name the model- and input-file. However, RStudio doesn't support the syntax highlight for these extensions. You can add `.R` as the extension for these files to help you edit your model or input in RStudio with syntax highlighting. Also, it can help you format your code in the code bracket.



```
1 #-----
2 # digoxin.model
3 #-----
4
5 # Variables
6 States = {A_central, A_periph};
7 Inputs = {Dose};
8 Outputs = {C_central};
9
10 # Structural model parameters
11 k_12 = 1.02;
12 k_21 = 0.15;
13 k_10 = 0.18;
14 V_central = 58.2;
15
16 # Measurement error
17 Ve_C_central = 1;
18
19 # Initialization
20 Initialize {A_central = Dose;
21 };
22
23 # Dynamics
24 Dynamics {
25   # Central compartment quantity
26   dt(A_central) = k_21 * A_periph - k_12 * A_central - k_10 * A_central;
27   # Peripheral compartment quantity
28   dt(A_periph) = k_12 * A_central - k_21 * A_periph;
29   # Concentration
30   C_central = A_central / V_central;
31 };
32
33 CalcOutputs {
34   C_central = (C_central < 1.0e-15 ? 1.0e-15 : C_central) ;
35 };
36
37 End.
38
```

Tips for MCSim under R(Studio)

Example & input

Some example R scripts will put into the `example` folder. To run *GNU MCSim* in model simulation, we need to have two types of file (1) **model** and (2) **input** files. The syntax of the model description file can find in *GNU MCSim User's Manual*. All example `model` and `input` files are located in the `modeling` folder.

[*] You don't need to move your model- or input-file to *modeling* folder, manually.

Just run `mcsim(model-file, input-file)` after you finish your code, then they will be moved to **modeling** folder.

```
— MCSim
— doc
— examples
  — linear.R
  — test_script.R
— modeling
  — digoxin.mcmc.in.R
  — digoxin.model.R
  — linear.in.R
  — linear.mcmc.in.R
  — linear.model.R
  — simple.in.R
  — simple.model.R
```

Following courses

- Tutorial 1: **Walk-through of working models** (4/25)
- Tutorial 2: **MC simulation and sensitivity analysis** (5/16)
- Tutorial 3: **Bayesian MCMC calibration** (5/23)

It's your turn to **run** it!



Take away

- **Bayesian statistics** is a powerful tool in toxicology
 - *GNU MCSim* & *R* are powerful tool in statistical computing and modeling
 - We provide an alternative way to run *GNU MCSim* in **RStudio** that aim to help user learn and familiar with *GNU MCSim* workflow
-

Meet problem? Please submit it to

- MCSim under R GitHub issues https://github.com/nanhung/MCSim_under_R/issues
- My email: nhsieh@cvm.tamu.edu

Slide at: bit.ly/190418_mcsim

