Basics of `mrgsolve`

Ashwin Karanam

July 16, 2020

Contents

1. About `mrgsolve`

- `R` package for simulation from ODE-based models
  - Free, OpenSource, GitHub, CRAN
- Language
  - Models written in `C++` inside model specification format
  - General purpose solver: `ODEPACK` / `DLSODA` (`FORTRAN`)
  - Simulation workflow in `R`
- Hierarchical (population) simulation
  - `ID`, η, ε
- Integrated PK functionaility
  - Bolus, infusion, `F`, `ALAG`, `SS` etc, handled under the hood
  - 1- and 2-cmt PK models in closed-form
- `R` is it's natural habitat

2. Installation

This is the trickiest part of mrgsolve. Once you figure it out, the rest will be easy. Please refer to `installation instructions here`. Prerequisits are:

- R: https://cran.r-project.org/ (Currently R >= 3.1.2 is required)
- Seriously consider running in Rstudio if possible. This is not required but is recommended.
- Compilers: C++ and gfortran

```
install.packages("mrgsolve")
```

---

3. Getting started with mrgsolve

There are two main components of simulating with mrgsolve:

- Model specifications in a C++ file
- Simulation specifics in a R Script

3.1. Model specifications

The model file for mrgsolve is written in C++ syntax and saved as a .cpp file. Language and format is quite similar to NONMEM, so NONMEMers will find the transition quite simple. While there are several blocks with in the .cpp file, we will be looking at an example with the most frequently used ones. For more details please refer to the model specification help document.

```
#Begin .cpp file
#Starting with next line, we will write the mrgsolve model in C++esque synntax

//PROBLEM
//Two compartment model + extravascular absportion + nonlinear clearance

$PARAM
TVCL = 1, TVVC = 20, TVQ =2, TVVP = 10, TVKA = 1, TVVMAX = 0, TVKM   =  2

$CMT  @annotated
// similar to $MODEL in NONMEM
// if @annotated is not used, simpley declare compartment names with a space

EV1    : First extravascular compartment (mass)
CENT   : Central compartment (mass)
PERIPH : Peripheral compartment (mass)

$MAIN
//akin to $PK block in NONMEM

double CL   = TVCL*exp(ETA(1));
double VC   = TVVC*exp(ETA(2));
double Q    = TVQ;
double VP   = TVVP;
double KA   = TVKA;
double VMAX = TVVMAX;
double KM   = TVKM;
```

```
$GLOBAL
// declare variables you want to either use further in computation
// OR derive variables you want calculated from computations

//int TEST = 1; // declaring variables needs ";" at the end
#define CP (CENT/VC) // # define does not require ";"
#define CT (PERIPH/VP)
#define CLNL (VMAX/(KM+CP))

$OMEGA @block
// generates a block matrix with the formulation (1,1), (1,2), (2,2)....
// @correlation will specify values to be corrleations instead as variances

0.1 0.02 0.3

$SIGMA
// sigma variances go here

0.01

$ODE
//similar to $DES in NONMEM
//differential equations go here

dxdt_EV1 = -KA*EV1;
dxdt_CENT = KA*EV1 - (CL+CLNL+Q)*CP  + Q*CT;
dxdt_PERIPH = Q*CP - Q*CT;

$TABLE
// Use $TABLE to interact with parameters, compartment values,
// and other user-defined variables after the system advances to the next time
// double CP = (CENT/VC);

double DV = CP * (1 + EPS(1));

$CAPTURE
//This is a block to identify variables that should be captured in the simulated output

CP, DV
```

3.2. Simulation spcifications

3.2.1. Reading the model into R

Let's start with `R` part. The first thing you will want to do is read your model file into `REnv` using `mrgsolve`.

```
# Call libraries
library(tidyverse)
library(mrgsolve)
library(gridExtra)
```

```
# mread("model","file-location")
mod <- mread("2cmtnl") # mread_cache() and mread_file() are variations
```

```
## Building 2cmtnl ... done.
```

```
#?mread()
```

Let's take a look at the model

```
mod
```

```
##
##
## ----------------  source: 2cmtnl.cpp  ----------------
##
##   project: C:/Users/ashwi/D... 6 - mrgsolve
##   shared object: 2cmtnl-so-3004790d36e0
##
##   time:          start: 0 end: 24 delta: 1
##                  add: <none>
##
##   compartments:  EV1 CENT PERIPH [3]
##   parameters:    TVCL TVVC TVQ TVVP TVKA TVVMAX TVKM [7]
##   captures:      CP DV [2]
##   omega:         2x2
##   sigma:         1x1
##
##   solver:        atol: 1e-08 rtol: 1e-08 maxsteps: 20k
## -------------------------------------------------------
```

```
param(mod)
```

```
##
##  Model parameters (N=7):
##  name value . name   value
##  TVCL 1     | TVVC   20
##  TVKA 1     | TVVMAX 10
##  TVKM 2     | TVVP   10
##  TVQ  2     | .      .
```

```
init(mod)
```

```
##
##  Model initial conditions (N=3):
##  name       value . name         value
##  CENT (2)   0     | PERIPH (3)   0
##  EV1 (1)    0     | . ...        .
```

3.2.2. Sequence of simulation

Let's now see how to simulate. The sequence for simulation is quite simple. Pipes come in handy. Simulations will a variation of one of the following:

- `model` %>% `intervention` %>% `options` %>% Go! %>% ...
- `model` %>% `intervention` %>% `population` %>% Go! %>% ...
- `model` %>% `data-set` %>% Go! %>% ...
  - where `data-set` = `intervention + population`

3.2.3. Interventions/Dosing Events

Simulations are run with `mrgsim()`. Details will be described later. Let us now look at creating interventions. They are called dosing events and the function used is `ev()`. Things to include in `ev()`:

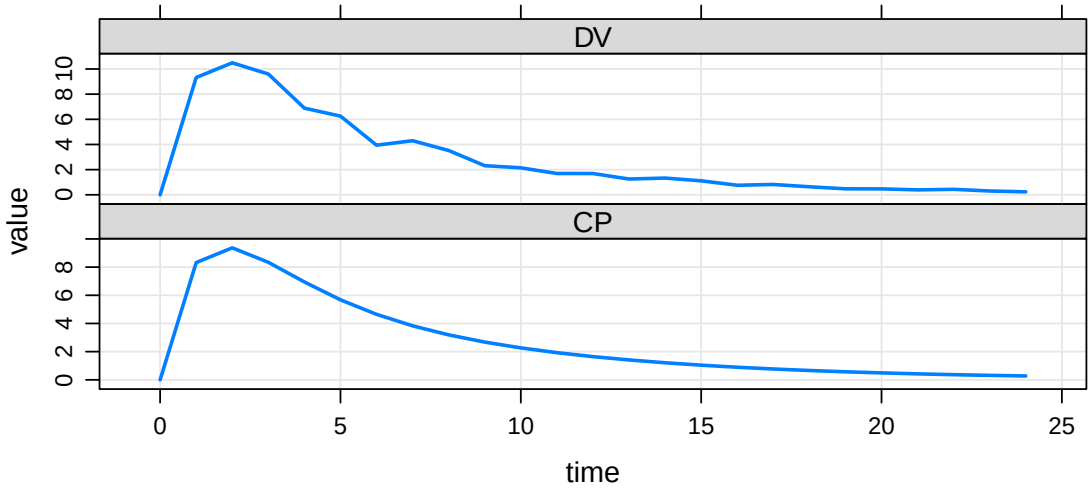
- `time` event time
- `cmt` Event compartment
- `amt` Dose amount
- `ii` Inter-dose interval
- `addl` Additional doses to administer
- `rate` Infusion rate
  - rate = 0 is bolus
  - rate > 0 is zero order infusion (`evid` 1)
- `ss` Set to 1 to advance to steady state
- `evid` Event id.
  - `evid` 2 is for "other"
  - `evid` 3 is for compartment reset
  - `evid` 4 is reset and dose
- `ID` Subject ID (use multiple ids - ID = 1:10)

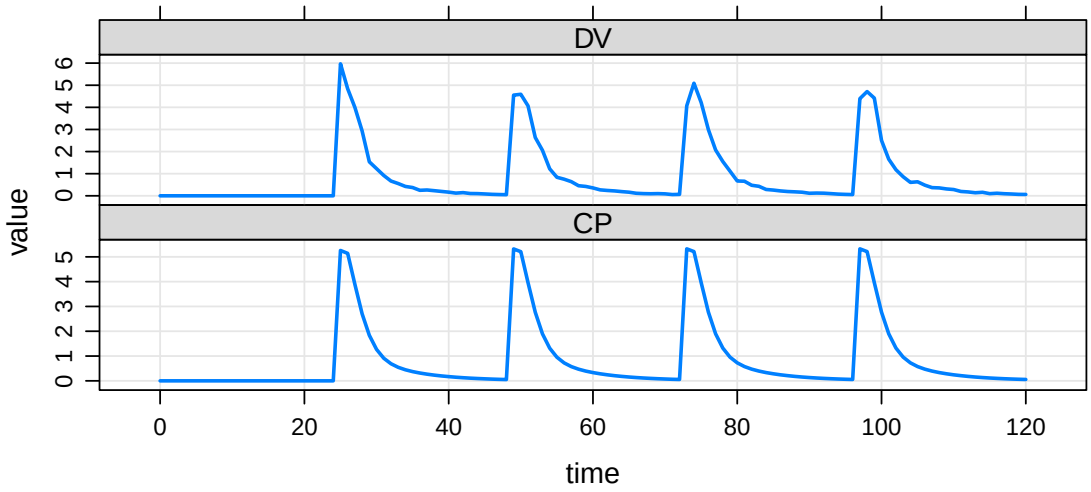
Let's look at a few interventions:

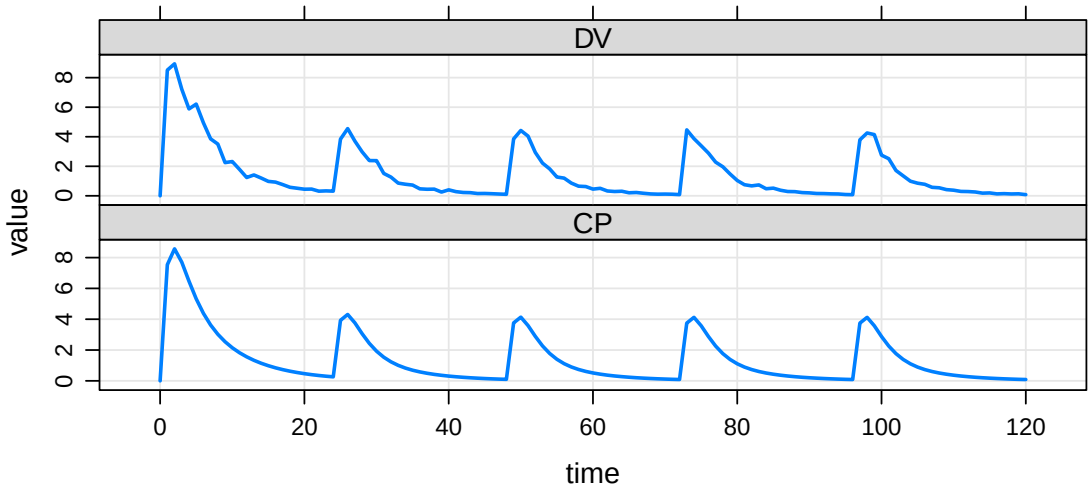
```
e1 <- ev(amt = 200)
e2 <- ev(amt = 100, time = 24, ii = 24, addl = 4)
e3 <- c(e1,e2)

set.seed(07192020)
p1 <- mod %>% req(CP,DV) %>% ev(e1) %>% mrgsim() %>% plot()
p2 <- mod %>% req(CP,DV) %>% ev(e2) %>% mrgsim(end=24*5) %>% plot()
p3 <- mod %>% req(CP,DV) %>% ev(e3) %>% mrgsim(end=24*5) %>% plot()

grid.arrange(p1,p2,p3,ncol=1)
```


DV
CP
value
time
0 2 4 6 8 10
0 2 4 6 8
0 5 10 15 20 25


DV
CP
value
time
0 1 2 3 4 5 6
0 1 2 3 4 5
0 20 40 60 80 100 120

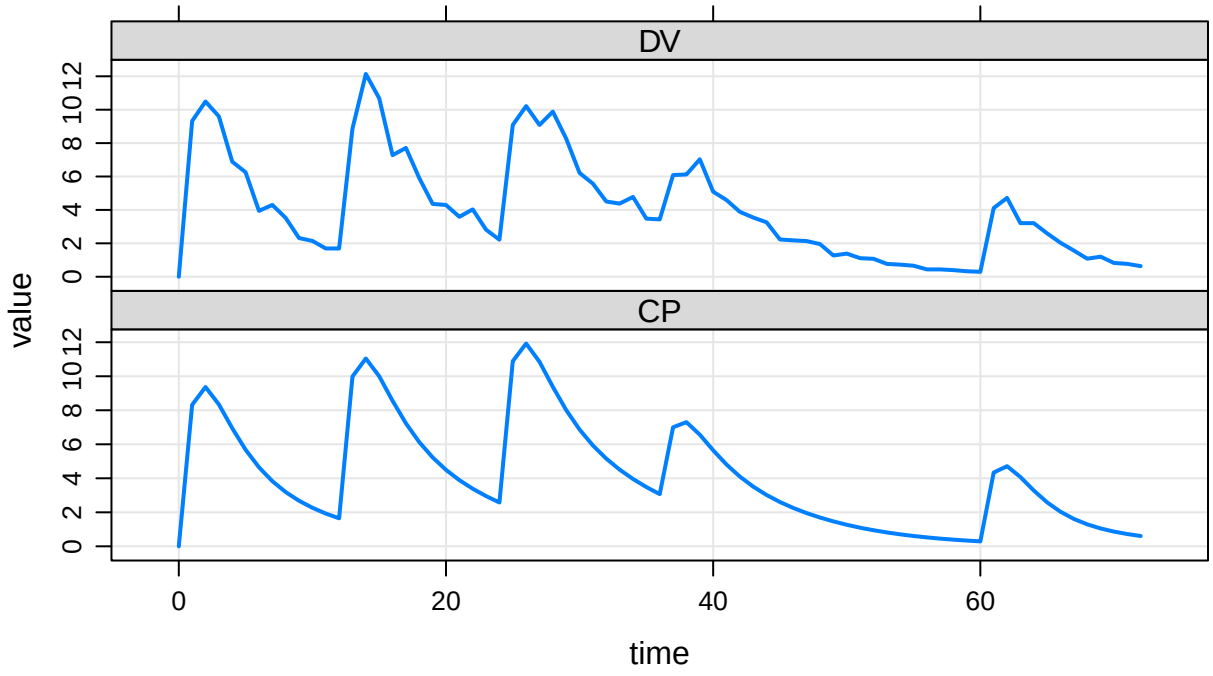

DV
CP
value
time
0 2 4 6 8
0 20 40 60 80 100 120

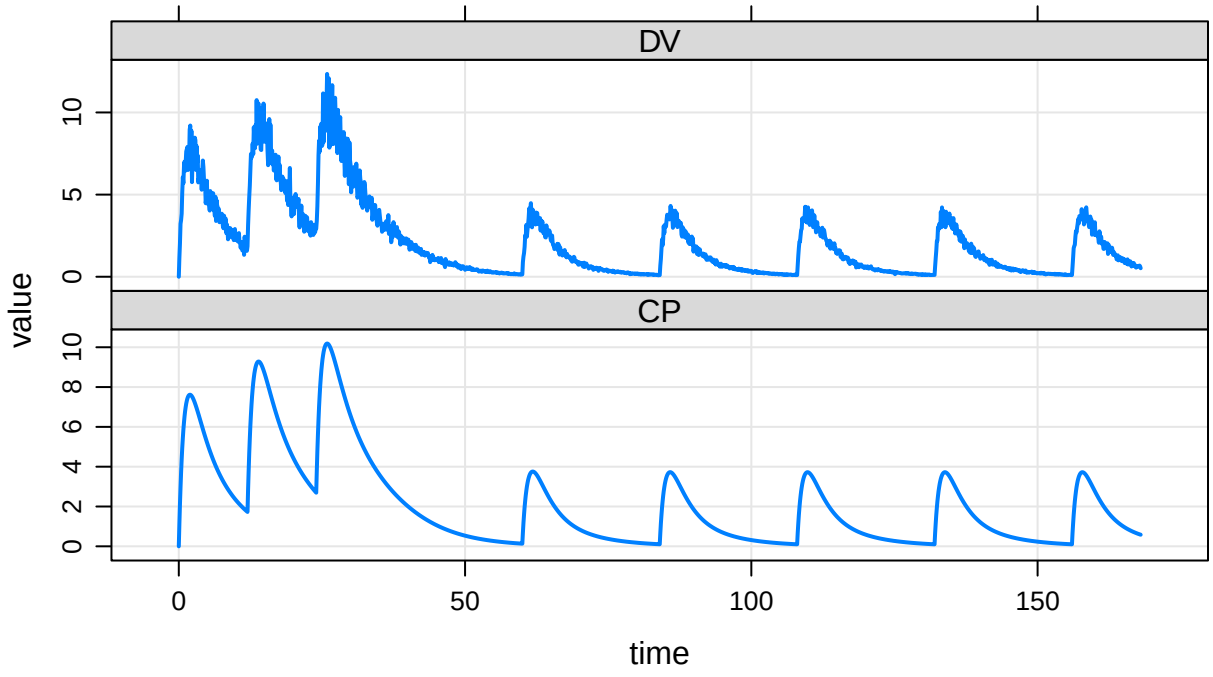
```
# You could also add them in sequence

e1 <- ev(amt = 200, ii = 12, addl = 2)
e2 <- ev(amt = 100, ii = 24, addl = 4)
e3 <- seq(e1, e2)
e4 <- seq(e1, wait = 24, e2)

set.seed(07192020)
p1 <- mod %>% req(CP,DV) %>% ev(e3) %>% mrgsim(end=24*3,delta=1) %>% plot()
p2 <- mod %>% req(CP,DV) %>% ev(e4) %>% mrgsim(end=24*7,delta=0.1) %>% plot()

grid.arrange(p1,p2,ncol=1)
```





These are for individual simulations. Now let's take a look at simulating populations.

3.2.4. Population level simulations

```
# If individual level parameters avaliable

idata <- expand.idata(TVCL = seq(0.5, 1.5, 0.25))
idata
```

```
##   ID TVCL
## 1  1 0.50
## 2  2 0.75
## 3  3 1.00
## 4  4 1.25
## 5  5 1.50
```

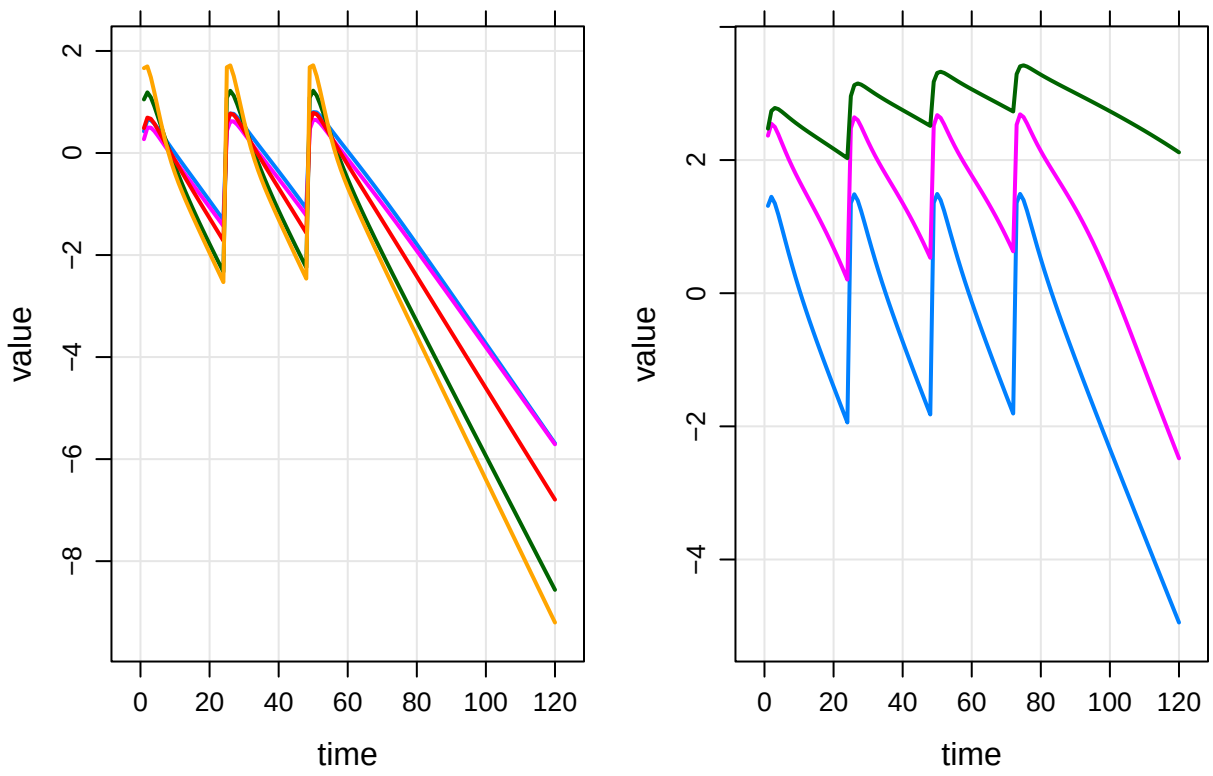
```
# Another formulation
# Can also carry parameters
data <- expand.ev(amt = c(100, 300, 1000), ii = 24, addl = 3)
head(data)
```

```
##   ID time  amt ii addl cmt evid
## 1  1    0  100 24    3   1    1
## 2  2    0  300 24    3   1    1
## 3  3    0 1000 24    3   1    1
```

```
# Let's simulate both
set.seed(07192020)
p1 <- mod %>%
  idata_set(idata) %>%
  ev(amt = 100, ii = 24, addl = 2) %>%
  mrgsim(end = 120) %>% plot(log(CP) ~ .)

p2 <- mod %>%
  data_set(data) %>%
  mrgsim(end = 120) %>% plot(log(CP) ~ .)

grid.arrange(p1,p2,ncol=2)
```



Let's look at generating more population datasets

```
# Combinations using exapnd.ev
expand.ev(ID = 1:2, amt = c(100,200))
```

```
##   ID time amt cmt evid
## 1  1    0 100   1    1
## 2  2    0 100   1    1
## 3  3    0 200   1    1
## 4  4    0 200   1    1
```

```
# More datasets
as_data_set(
  ev(amt = 100, ii = 12, addl = 19, ID = 1:2),
  ev(amt = 200, ii = 24, addl = 9,  ID = 1:3),
  ev(amt = 150, ii = 24, addl = 9,  ID = 1:4)
)
```

```
##   ID time cmt evid amt ii addl
## 1  1    0   1    1 100 12   19
## 2  2    0   1    1 100 12   19
## 3  3    0   1    1 200 24    9
## 4  4    0   1    1 200 24    9
## 5  5    0   1    1 200 24    9
## 6  6    0   1    1 150 24    9
```

```
## 7  7    0   1    1 150 24    9
## 8  8    0   1    1 150 24    9
## 9  9    0   1    1 150 24    9
```

3.4.5. Updating parameters in the model

Now, let's take a look at updating typical parameters in the model rather than for individual

```
#Use param() to update $param block elements
param(mod)
```

```
##
##  Model parameters (N=7):
##  name value . name   value
##  TVCL 1     | TVVC   20
##  TVKA 1     | TVVMAX 10
##  TVKM 2     | TVVP   10
##  TVQ  2     | .      .
```

```
mod2 <- mod %>% param(TVVC = 30)
param(mod2)
```

```
##
##  Model parameters (N=7):
##  name value . name   value
##  TVCL 1     | TVVC   30
##  TVKA 1     | TVVMAX 10
##  TVKM 2     | TVVP   10
##  TVQ  2     | .      .
```

```
#Use omat() to update $omega block elements
omat(mod)
```

```
## $...
##     [,1] [,2]
## 1:  0.10 0.02
## 2:  0.02 0.30
```

```
mod3 <- mod %>% omat(bmat(0.2,0.01,0.3)) #use damt() for diagonal matrix
omat(mod3)
```

```
## $...
##     [,1] [,2]
## 1:  0.20 0.01
## 2:  0.01 0.30
```

```
#Use smat() to update $sigma block elements
smat(mod)
```

```
## $...
##     [,1]
## 1:  0.01
```

```r
mod4 <- mod %>% smat(dmat(0.2)) #use bamt() for block diagonal matrix
smat(mod4)
```

```
## $...
##     [,1]
## 1:   0.2
```

Useful Resources

- https://github.com/mrgsolve/learn
- GitHub site: https://github.com/metrumresearchgroup/mrgsolve
- mrgsolve website: https://mrgsolve.github.io
- User Guide: https://mrgsolve.github.io/user_guide