mrgsolve: R Workflow

mrgsolve Workshop
March 12, 2016
San Diego, CA

Load mrgsolve

```
library(mrgsolve)
```

I usually work with these packages as well

```
library(dplyr)
library(magrittr)
library(tidyr)
```

Read a model from a file

```
mod <- mread("mymodel", proj)
```

- `<model-name>`, `<project-directory>`, `<code>`
- `mrgsolve` assumes there is a file called `mymodel.cpp` in directory `proj`
- Prefer to keep model code in it's own file
  - Code-reuse
  - Syntax highlighting
- Parse, write `.cpp` file, compile, and load the shared object (`dyn.load`)
- Returns a model object (class `mrgmod`)
  - Contains all of the basic information `mrgsolve` needs to run the model

An example using code argument

```
code <- '
  $PARAM CL = 1, VC=2
  $CMT CENT
  $ODE dxdt_CENT = -(CL/VC)*CENT;
'
```

```
mod <- mread("mycode", tempdir(), code)
```

- `mrgsolve` writes code to `tempdir()/mycode.cpp`, then reads it back in
- Use single quote around code so you can use double quotes inside

mread returns a model object

```
mod <- mread("mymodel", proj)
class(mod)
```

```
. [1] "mrgmod"
. attr(,"package")
. [1] "mrgsolve"
```

```
mod <-
  mread("mymodel", proj) %>%
  update(end=240) %>%
  param(CL = 1.5)
```

Model overview

```
mod
```

```
.
.
. -------- mrgsolve model object (unix) --------
.   Project: /Users/kyleb/CTS/script/models
.   source:        mymodel.cpp
.   shared object: b06f173f72b9 (loaded)
.
.   compile date:  03/12 11:18
.   Time:          start: 0 end: 240 delta: 1
.   >              add: <none>
.   >              tscale: 1
.
.   Compartments:  GUT CENT [2]
.   Parameters:    CL VC KA [3]
.   Omega:         0x0
.   Sigma:         0x0
.
.   Solver:        atol: 1e-08 rtol: 1e-08
.   >              maxsteps: 2000 hmin: 0 hmax: 0
```

Check parameters, compartments, and initial conditions

```
param(mod)
```

```
.
.  Model parameters (N=3):
.  name value . name value
.  CL   1.5   | VC   20
.  KA   1.1   | .    .
```

```
init(mod)
```

```
.
.  Model initial conditions (N=2):
.  name        value . name       value
.  CENT (2)    0     | GUT (1)    0
```

Look at the model code

```r
see(mod)
```

```
.
. Model file:  mymodel.cpp
. $PARAM
. CL = 1, VC=20, KA=1.1
. $ADVAN2
. $CMT GUT CENT
. $MAIN
. pred_CL = CL;
. pred_V = VC;
. pred_KA = KA;
.
. $TABLE
. table(CP) = CENT/VC;
```

Use mrgsim to run the model

- Return is object of class mrgsims
- Pass in items to send to update

```
out <- mod %>% init(CENT=2000) %>% mrgsim(delta=3)
```

```
out
```

```
. Model:  mymodel.cpp
. Date:   Sat Mar 12 11:18:48 2016
. Dim:    81 x 5
. Time:   0 to 240
. ID:     1
.      ID time GUT   CENT     CP
. [1,]  1    0   0 2000.0 100.00
. [2,]  1    3   0 1597.0  79.85
. [3,]  1    6   0 1275.3  63.76
. [4,]  1    9   0 1018.3  50.92
. [5,]  1   12   0  813.1  40.66
. [showing 4 significant digits]
```

Using pipes

- We prefer to use **pipes** (%>%) to configure the model object and run the simulation
- We use functions that have inputs (arguments) and return values
    - Pipes take the return value from one function and sends it as an argument to the next function
- Allows chaining commands to configure simulation and manipulate output
    - Easy to read
    - Many simple functions that do small, specific tasks

```
mod %>% init(GUT=100) %>% Req(CP) %>% mrgsim
```

1. `mod` is piped into `init`
   - The `GUT` initial condition is set to 100
   - `init` returns the updated `mod`
2. Next, `mod` is passed into `mrgsim`
   - The simulation is run
   - `mrgsim` returns an object of class `mrgsims`

```
mod %>% init(.,GUT=100) %>% Req(.,CP) %>% mrgsim(.)
```

What is this: %<>% ???

Compound assignment pipe operator (of course!)

```
mod <- mod %>% init(GUT=100)
```

Is the same as this:

```
mod %<>% init(GUT=100)
```

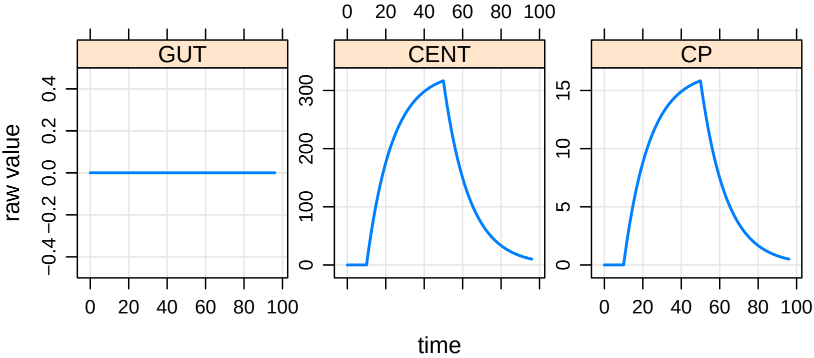
Simulation output objects

mrgsim returns an object with class mrgsims. This is just a matrix of simulated data plus other information about what happened in the simulation.

- We have some special methods to work with mrgsims objects
  - plot
  - as.data.frame, as.matrix, as.tbl
  - head, tail, dim, names, summary, $
  - mod, param, init, events
  - mutate, filter, group_by, do, summarise, summarise_each
- The dplyr-related verbs all return some sort of dplyr data table

Just plot

```
mod %>%
  ev(amt=1000,rate=25,cmt=2,time=10) %>%
  mrgsim(delta=0.1,end=96) %>%
  plot
```



Limit maximum simulated time to 3

```
mod %>%
  Req(CP) %>%
  init(GUT=100) %>%
  #-----------------------
  mrgsim %>%
  #-----------------------
  filter(time < 3)
```

```
. Source: local data frame [3 x 3]
.
.      ID  time       CP
.   (dbl) (dbl)    (dbl)
. 1      1     0 0.000000
. 2      1     1 3.191998
. 3      1     2 4.023880
```

Get the maximum values in GUT and CP

```r
mod %>%
  init(GUT=100) %>%
  mrgsim %>%
  summarise.each(funs(max), GUT:CP)
```

```
. Source: local data frame [1 x 3]
.
.     GUT     CENT       CP
.   (dbl)    (dbl)    (dbl)
. 1   100 81.73623 4.086811
```

Retreive the model object that was used to simulate

```
out <- mod %>% init(GUT=1234) %>% mrgsim
```

```
init(out)
```

```
.
.  Model initial conditions (N=2):
.  name       value . name      value
.  CENT (2)   0     | GUT (1)   1234
```

```
init(mod)
```

```
.
.  Model initial conditions (N=2):
.  name       value . name      value
.  CENT (2)   0     | GUT (1)   0
```

Update the model

Update simulation time grid

- `start`, `end`, `delta`, `add`

```
mod %<>% update(end=240, delta=4, add=seq(0,2,0.1))
```

Update parameters

```
mod %<>% param(CL=1.7, VC=22.5)
```

Other things you can update

- `atol`, `rtol`, `hmax`, `maxsteps`, `mxhnil`,`ixpr`
- `$OMEGA`, `$SIGMA`
- `tscale` (rescale the output time)
- `digits`

Other methods to update parameters

```r
p <- list(CL=2.1, VC=17.2, KYLE = 777)
mod %>% param(p) %>% param
```

```
.
.   Model parameters (N=3):
.   name value . name value
.   CL   2.1   | VC   17.2
.   KA   1.1   | .    .
```

```r
d <- data_frame(CL=c(9,10), VC=c(11,12), KTB=c(13,14))
mod %>% param(d[2,]) %>% param
```

```
.
.   Model parameters (N=3):
.   name value . name value
.   CL   10    | VC   12
.   KA   1.1   | .    .
```

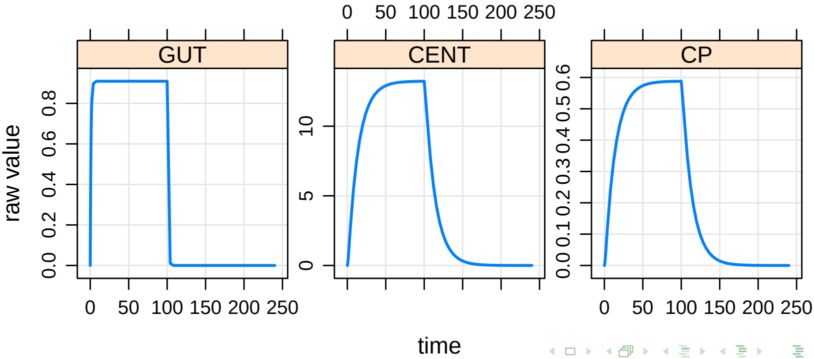
Two ways to introduce events

- A NONMEM-like data set
  - Every individual is represented in the data set
  - Different individuals may have different interventions
  - The data set may or may not have observation records
  - If no observation records (`evid==0`), `mrgsolve` will fill in with it's internal time grid
    - "condensed" data set
- An events object (`ev`)
  - The event object gets applied to every individual
  - Observations are determined by `start`/`end`/`delta`/`add`
  - `mrgsolve` turns this in to a NONMEM-like data set
    - Default `cmt`, `time`
    - `evid 0` is prohibited

Run the model with an event

```
out <- mod %>%
  ev(amt=100,rate=1) %>%
  mrgsim
```

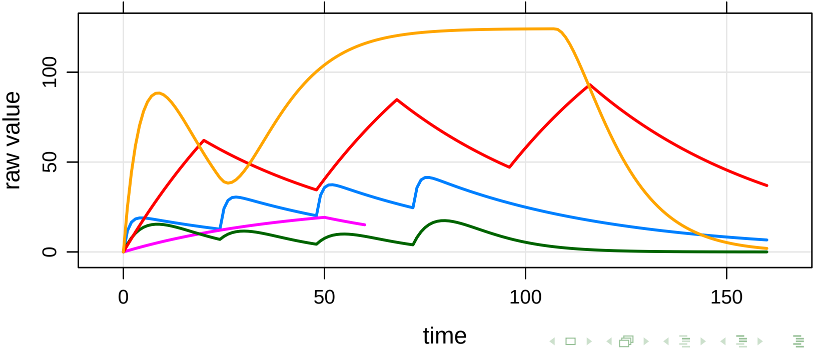
```
plot(out)
```



Run the model with a data set

```
data(extran3) ## ?exdatasets
```

```
mod %>%
  data_set(extran3) %>%
  mrgsim %>%
  plot(CP~.)
```



```r
head(extran3)
```

```
.   ID time cmt evid  amt addl ii rate   CL   VC    KA
. 1  1    0   1    1 1000    3 24    0 1.05 47.8 0.839
. 2  1    0   0    0    0    0  0    0 1.05 47.8 0.839
. 3  1    1   0    0    0    0  0    0 1.05 47.8 0.839
. 4  1    2   0    0    0    0  0    0 1.05 47.8 0.839
. 5  1    3   0    0    0    0  0    0 1.05 47.8 0.839
. 6  1    4   0    0    0    0  0    0 1.05 47.8 0.839
```

Reserved data set columns

- `ID`
- `time`
- `cmt`
- `amt`
- `ii`
- `addl`
- `rate`
- `ss`

see `?lctran`

Available interventions and corresponding evid

- Bolus dosing (evid 1, with rate==0)
- Zero order infusion (evid 1, with rate > 0)
- Other type event (evid 2)
  - This also forces solver reset
- Compartment reset (evid 3)
- Reset and dose (evid 4)
- Replace the amount in a specific compartment (evid 8)

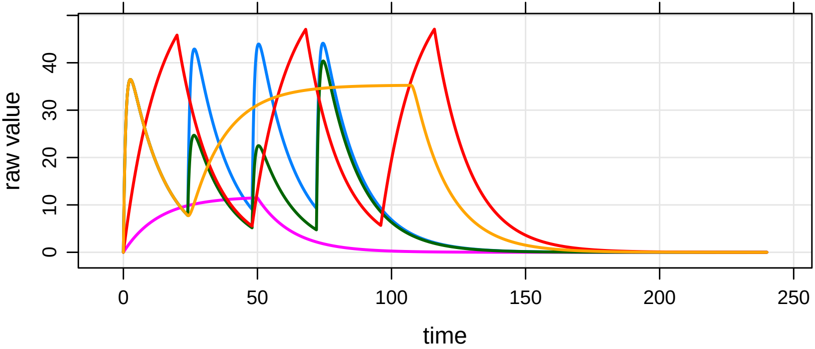
Condensed data set

```
data(extran1)
```

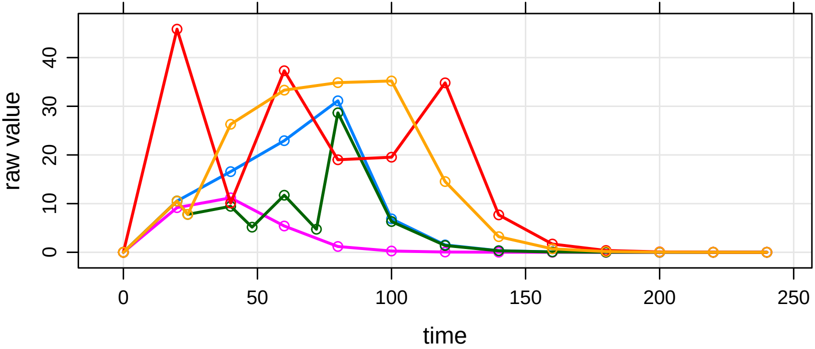
```
head(extran1)
```

```
.    ID  amt cmt time addl ii rate evid
. 1   1 1000   1    0    3 24    0    1
. 2   2 1000   2    0    0  0   20    1
. 3   3 1000   1    0    0  0    0    1
. 4   3  500   1   24    0  0    0    1
. 5   3  500   1   48    0  0    0    1
. 6   3 1000   1   72    0  0    0    1
```

```
mod %>%
  data_set(extran1) %>%
  Req(CP) %>%
  mrgsim(delta=0.1) %>%
  plot
```



```
mod %>%
  data_set(extran1) %>%
  Req(CP) %>%
  mrgsim(delta=20, add=numeric(0)) %>%
  plot(type='b')
```



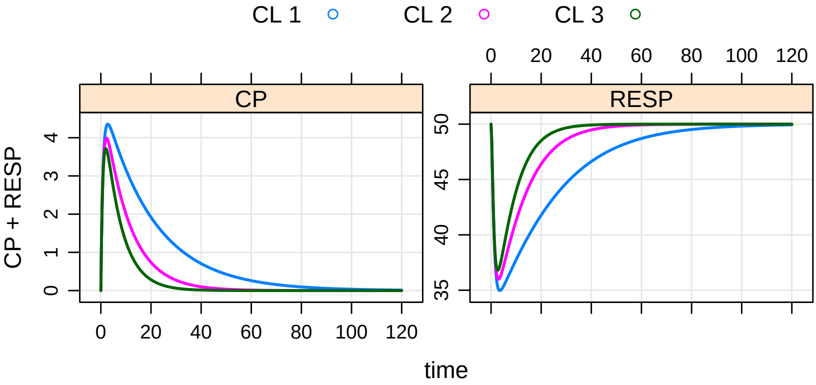
Summary: simulate with events and data sets

1. `mod %>% ev(...)  %>% ...`
   - One `ID` gets events in `ev`
   - But see what happens when you use `idata` set with `ev`
   - Simulation times from `mod` (`start/end/delta/add`)
2. `mod %>% data_set(...)  %>% ...` [`evid != 0` only]
   - As many `ID`s as found in the data set
   - Simulation times from `mod`
3. `mod %>% data_set(...)  %>% ...` [includes > 0 `evid`=0]
   - As many `ID`s as in the data set
   - Only simulation times that are coded into the data set
   - But see `obsaug` argument to `mrgsim`

knobs

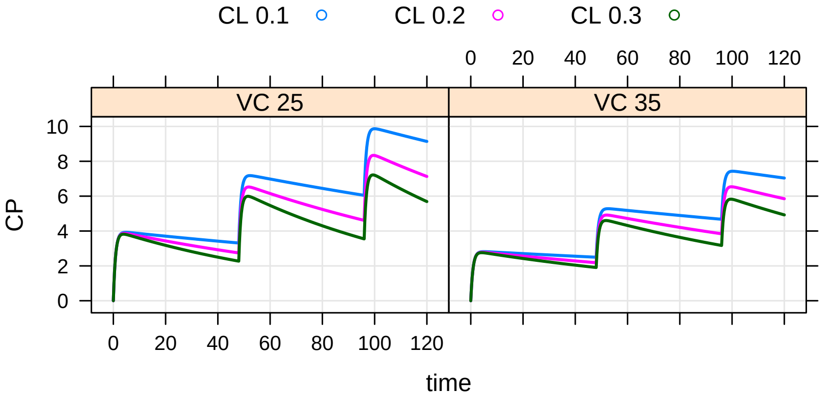
- A simple way of testing all combinations of inputs

```
mod <- mrgsolve:::house()
out <- mod %>% knobs(CL = c(1,2,3),amt=100,cmt=1)
```



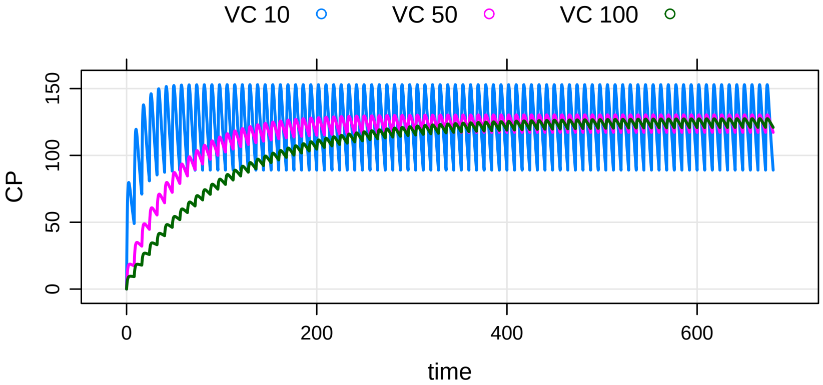
Sensitivity analysis on two parameters

```
out <- mod %>% knobs(CL = c(1,2,3)/10,
                     VC=c(25,35),
                     amt=100,cmt=1, ii=48, addl=2)
```



A run using events

```
out <- mod %>%
  ev(amt=1000,ii=8,addl=100) %>%
  knobs(VC=c(10,50,100), end=680, delta=0.25)
```



Sensitivity analysis through idata

Remember what our **parameter names** are

```
param(mod)
```

```
.
.  Model parameters (N=13):
.  name value . name  value
.  CL   1     | SEXCL 0.7
.  F1   1     | SEXVC 0.85
.  IC50 10    | VC    20
.  KA   1.2   | WT    70
.  KIN  100   | WTCL  0.75
.  KOUT 2     | WTVC  1
.  SEX  0     | .     .
```

Create a bunch of combinations of parameters

```r
pars <- expand.idata(CL=seq(1,5,1), VC=seq(10,40,10))
```

```r
head(pars)
```

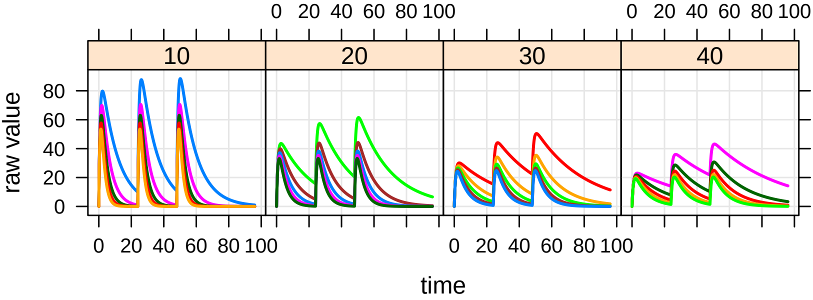
```
.   ID CL VC
. 1  1  1 10
. 2  2  2 10
. 3  3  3 10
. 4  4  4 10
. 5  5  5 10
. 6  6  1 20
```

NOTE

- There is an `ID` column
- The names of the other columns correspond to names in `$PARAM`
- `mrgsolve` will update parameter list right before `ID` is started

We can run all of these IDs in one run

```
out <-
  mod %>%
  idata_set(pars) %>%
  carry.out(CL,VC) %>%
  ev(amt=1000,ii=24,addl=2) %>%
  mrgsim(end=96,delta=0.1)
```



idata data sets

- One ID per row; ID should be unique
- When mrgsolve finds a parameter name in an idata column, that parameter will get updated right before starting to simulate that ID
- Columns with compartment initial name (CMT_0) will set that initial condition
- When a data_set is **not** used, the number of individuals in idata determine the size if the population
- Do not put dosing records / information in idata ... that goes in data only

How to set up your simulations (1)

1. `mod %>% ev(...)  %>% ...`
   - One `ID` gets events in `ev`
   - Simulation times from `mod` (`start/end/delta/add`)
   - Parameters from the base parameter list only
2. `mod %>% data_set(...)  %>% ...` [`evid != 0` only]
   - As many `IDs` as found in the data set
   - Simulation times from `mod`
   - Parameters from base list or from `data`
3. `mod %>% data_set(...)  %>% ...` [includes $>$ 0 `evid=0`]
   - As many `IDs` as in the data set
   - Only simulation times that are coded into the data set
   - But see `obsaug` argument to `mrgsim`
   - Parameters from base list or from `data`

How to set up your simulations (2)

1. `mod %>% idata_set(...) %>% ev(...) %>% ...`
   - As many `IDs` as in the `idata` set
   - Simulation times from `mod`
   - Parameters from base list or `idata`
2. `mod %>% idata_set(...) %>% data_set(...) %>% ...`
   - As many `IDs` as in the `data` set
   - Simulation times from `mod` or data set
     - data set if it includes `evid=0`
     - `mod` if data set has no `evid=0`
   - Individual parameters looked up in `idata`

Function summary

Before mrgsim

- update
- param
- init
- omat
- smat
- ev
- data_set
- idata_set
- Req, req
- carry.out
- obsonly, obsaug
- drop.re, zero.re

After mrgsim (via mrgsolve)

- plot
- as.data.frame
- as.matrix

After mrgsim (via dplyr)

- as.tbl
- filter
- group_by
- mutate
- select
- summarise
- summarise.each

Review

- Read / load a model with `mread`
- Various functions to look at the model
- `param`, `init`, `print`, `see`
- Use pipes in your work flow
- Simulate with `mrgsim`
- Various ways to update the model object
- Introduce events with data sets or events objects