nlmixr: an R package for fitting PK and PKPD models

Wenping Wang

2016-10-21

Introduction

'nlmixr' is an R package for fitting general dynamic models, pharmacokinetic (PK) models and pharmacokinetic-pharmacodynamic (PKPD) models in particular, with either individual data or population data. `nlmixr` has five main modules: 1) `dynmodel()` and its mcmc cousin `dynmodel.mcmc()` for nonlinear dynamic models of individual data; 2) `nlme_lin_cmpt()`for one to three linear compartment models of population data with first order absorption, or i.v. bolus, or i.v. infusion; 3) `nlme_ode()` for general dynamic models defined by ordinary differential equations (ODEs) of population data; 4) `saem_fit()` for general dynamic models defined by closed-form solutions or ordinary differential equations (ODEs) of population data by the Stochastic Approximation Expectation-Maximization (SAEM) algorithm; 5) `gnlmm()` for generalized non-linear mixed-models (optionally defined by ordinary differential equations) of population data by adaptive Gaussian quadrature algorithm.

A few utilities to facilitate population model building are also included in `nlmixr`.

```
library(nlmixr, quietly = TRUE)
```

```
##
## Attaching package: 'RcppArmadillo'

## The following objects are masked from 'package:RcppEigen':
##
##     fastLm, fastLmPure

##
## Attaching package: 'inline'

## The following object is masked from 'package:Rcpp':
##
##     registerPlugin
```

```
source("print.summary.lme.R")     #suppress data printout
```

Non-population dynamic model

The `dynmodel()` module fits general dynamic models, often expressed as a set of ODEs, of individual data with possible multiple endpoints. This module has similar functionality as the `ID` module of `ADAPT 5`.

We use two examples from the ADAPT 5 User's Guide to illustrate the usage of non-population dynamic model with `dynmodel()`.

Inverse Gaussian Absorption Model

This example illustrates the use of the inverse Gaussian (IG) function to model the oral absorption process of a delayed release compound. It is assumed that the plasma drug concentration following oral administration of the drug can be decomposed into an independent input process (representing dissolution, transit and absorption processes) followed by the disposition process. It is further assumed that the parameters of a linear two compartment model used to describe the disposition process have been estimated following intravenous drug administration to an individual. The model shown in Figure 1 will then be used to describe the plasma kinetics of an oral formulation of the drug delivered to the individual.

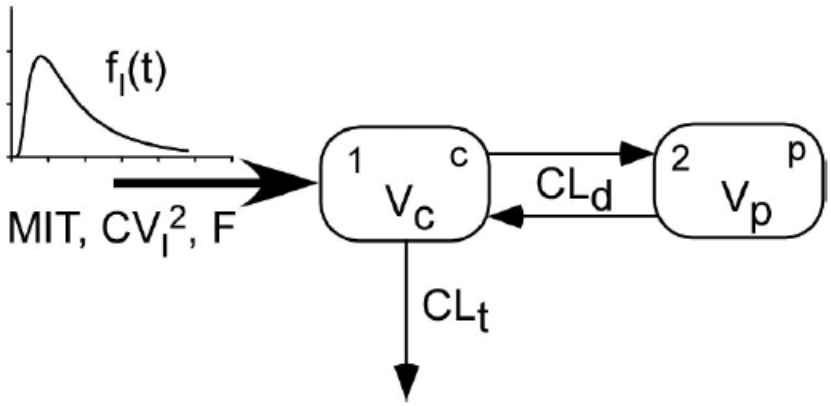


Figure 1. Two compartment disposition model with IG function input

This system of a two-compartment disposition model with IG absorption is defined in the following string:

```
ode <- "
   dose=200;
   pi = 3.1415926535897931;

   if (t<=0) {
      fI = 0;
   } else {
      fI = F*dose*sqrt(MIT/(2.0*pi*CVI2*t^3))*exp(-(t-MIT)^2/(2.0*CVI2*MIT*t));
   }

   C2 = centr/V2;
   C3 = peri/V3;
   d/dt(centr) = fI - CL*C2 - Q*C2 + Q*C3;
   d/dt(peri)  =              Q*C2 - Q*C3;
"
sys1 <- RxODE(model = ode)
```

In the model above the systemic drug input function, $f_i(t)$, is assumed to be a single inverse Gaussian function defined as:

$$f_i(t) = D \cdot F \sqrt{\frac{MIT}{2\pi CV_t^2 t^3}} \exp\left[-\frac{(t-MIT)^2}{2CV_t^2 MIT t}\right]$$

where `MIT` represents the mean input time and CV^2 is a normalized variance (is the standard deviation of the density function $f_i(t)/(D \cdot F)$ divided by `MIT`, i.e., the relative dispersion of input times). The factor `F` is the bioavailability of the orally administered dose .

In this example, disposition parameters are assumed known. The three parameters related to the delayed absorption, `MIT`, `CVI2` and `F`, are to be estimated.

`dynmodel()` takes the following arguments: an RxODE object (compiled ODE solver), a list of formulae that relates system defined quantities and measurement(s) with either or both additive error `add()` and proportional error `prop()`, an event table object that defines the inputs and observation schedule, a named vector with initial values of system parameters, a `data.frame` contains the data, optional known system parameters (`fixPars`) not to be estimated, and other optional control parameters for optimization routines.

```
dat <- read.table("invgaussian.txt", header=TRUE)
mod <- cp ~ C2 + prop(.1)
inits <- c(MIT=190, CVI2=.65, F=.92)
fixPars <- c(CL=.0793, V2=.64, Q=.292, V3=9.63)
ev <- eventTable()
ev$add.sampling(c(0, dat$time))
(fit <- dynmodel(sys1, mod, ev, inits, dat, fixPars))
```

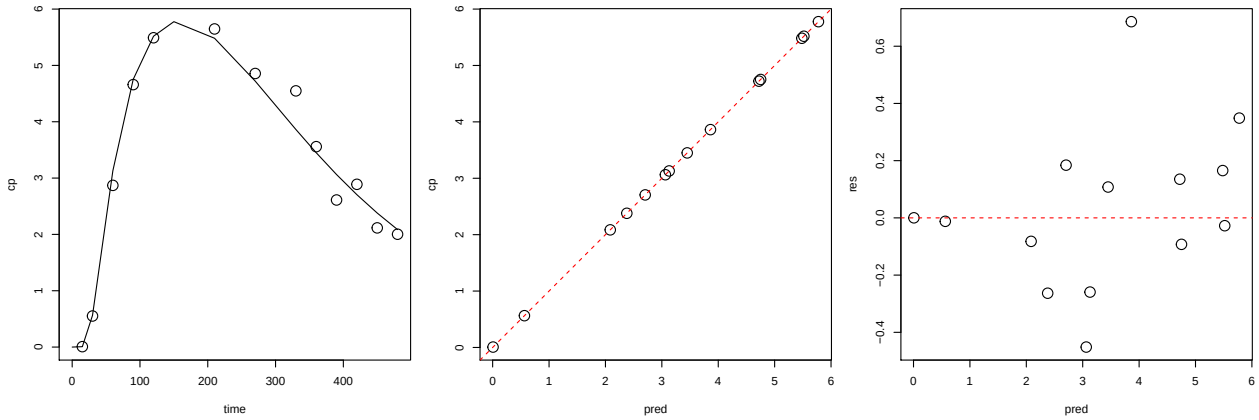
```
##              est        %cv
## MIT  191.9428628  0.1543739
## CVI2   0.6555526  0.9141670
## F      0.9060619  2.2309059
## err    0.0786291 18.9836400
##
##    -loglik        AIC        BIC
## -5.5422991 -3.0845982 -0.5283689
```

More information about the model (convergence information and number of function evulation of the optimization process) is displayed by calling the `summary()` function. Basic goodness-of-fit plots are generated by calling `plot()`.

```
summary(fit)
```

```
##              est          se        %cv
## MIT  191.9428628 0.296309668  0.1543739
## CVI2   0.6555526 0.005992846  0.9141670
## F      0.9060619 0.020213388  2.2309059
## err    0.0786291 0.014926666 18.9836400
##
##    -loglik        AIC        BIC
## -5.5422991 -3.0845982 -0.5283689
##
## iter: 165
## NELDER_FTOL_REACHED
```

```
par(mar=c(4,4,1,1), mfrow=c(1,3))
plot(fit, cex=2)
```



Parent/Metabolite (multiple endpoints)

Figure 2 shows the model used to describe the kinetics of a parent compound and its metabolite used in this example. The model relating dose of the parent compound to the plasma concentrations of parent drug and its metabolite can be rewritten in terms of the ratio Vm/fm along with the other model parameters Kp, Vp, K_{12}, K_{21} and fm.

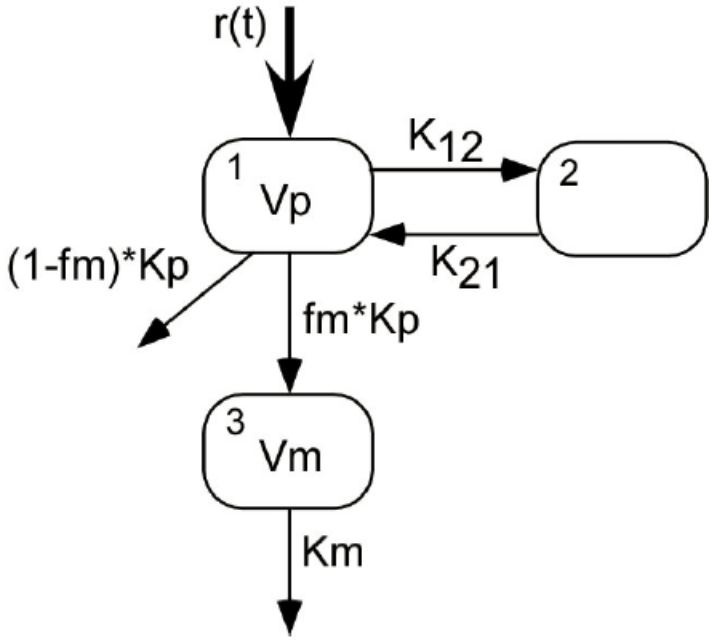


Figure 2. Model for example pmetab. Kp is the total elimination rate of the parent compound, while fm represent the fraction metabolized.

Note the list that specifies the statistical measurement models for concentrations of both parent and its metabite.

```
ode <- "
Cp = centr/Vp;
Cm = meta/Vm;
d/dt(centr) = -k12*centr + k21*peri -kp*centr;
d/dt(peri)  =  k12*centr - k21*peri;
```

```
d/dt(meta)  =                        kp*centr - km*meta;
"
sys2 <- RxODE(model = ode)

dat <- read.table("metabolite.txt", header=TRUE)
mod <- list(y1 ~ Cp+prop(.1), y2 ~ Cm+prop(.15))
inits <- c(kp=0.4, Vp=10., k12=0.2, k21=0.1, km=0.2, Vm=30.)
ev <- eventTable()
ev$add.dosing(100, rate=100)
ev$add.sampling(c(0, dat$time))
(fit <- dynmodel(sys2, mod, ev, inits, dat))
```

```
##             est       %cv
## kp   0.38362979  5.092584
## Vp  10.71097520  5.976756
## k12  0.17878774  8.415824
## k21  0.10020406 13.669766
## km   0.20962035  7.223518
## Vm  28.59650129  8.488434
## err  0.07706082 22.648616
## err  0.14501513 22.974311
##
##   -loglik       AIC       BIC
## -27.31560 -38.63120 -30.66534
```

Alternative error models can be tested and compared without re-compilation of the system. For instance, a combined error structure with both additive and proportional errors for the parent compound concentration is easily re-fitted with the following code:

```
mod <- list(y1 ~ Cp+add(.2)+prop(.1), y2 ~ Cm+prop(.15))
(fit <- dynmodel(sys2, mod, ev, inits, dat))
```

```
##              est        %cv
## kp   0.379652225   4.764781
## Vp  10.759771690   4.905665
## k12  0.175326233   7.816279
## k21  0.096208347  14.905049
## km   0.209364068   7.094367
## Vm  28.482949275   8.458955
## err  0.004999634 117.382974
## err  0.061440884  31.876548
## err  0.146165651  23.044552
##
##   -loglik       AIC       BIC
## -27.68870 -37.37741 -28.41582
```

Although the combined error produces a slightly higher likelihood, the previous proportional error model has smaller AIC and BIC, and is hence preferred.

Parent/Metabolite (continued - mcmc estimation)

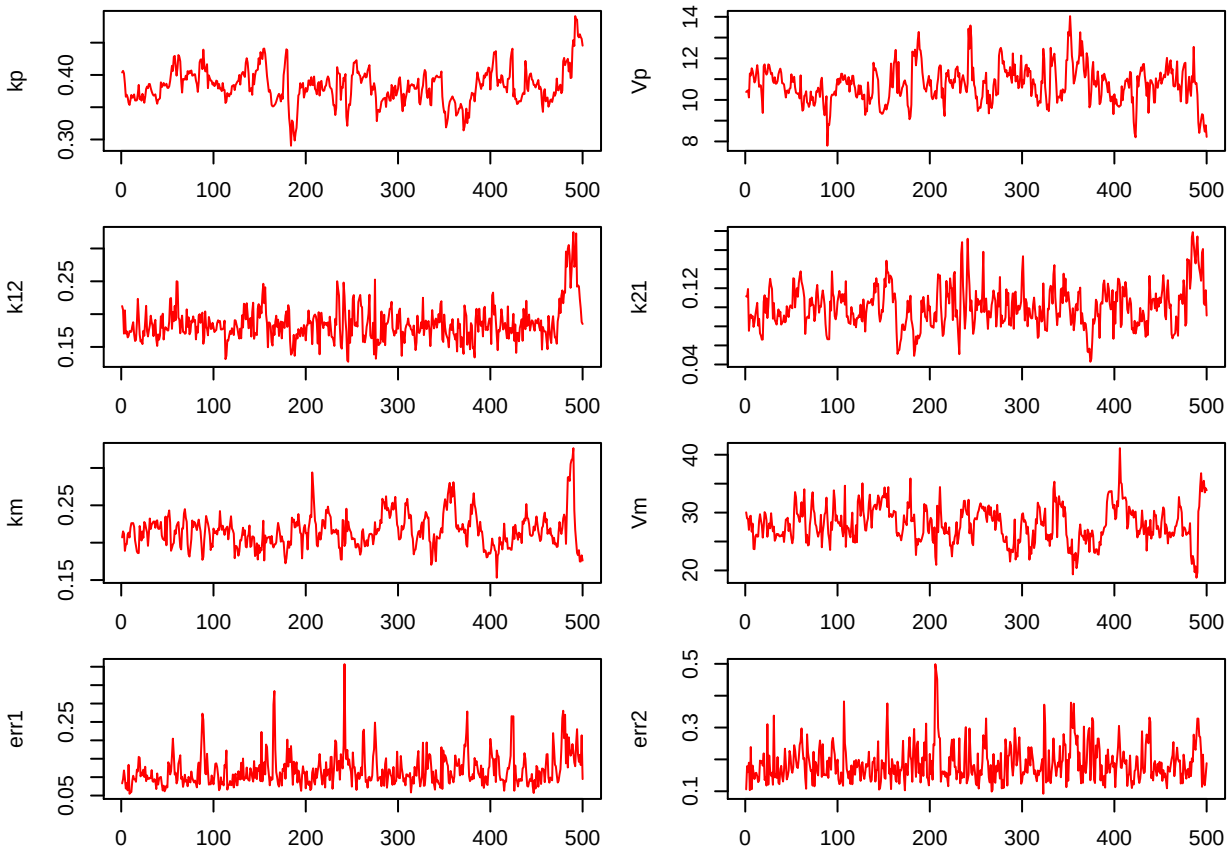
The `dynmodel.mcmc()` has a similar functionality and user interface as `dynmodel()` for general dynamic models, except it uses Bayesian Markov Chain Monte-Carlo (mcmc) for estimation. The underlying sampling algorithm is Neal's efficient slice sampling algorithm.

```
mod <- list(y1 ~ Cp+prop(.1), y2 ~ Cm+prop(.15))
(fit <- dynmodel.mcmc(sys2, mod, ev, inits, dat))
```

```
##            mean         sd       cv%
## kp    0.3831135 0.02754457  7.189662
## Vp   10.7457581 0.89146183  8.295942
## k12   0.1853772 0.02741360 14.788007
## k21   0.1007913 0.02206588 21.892647
## km    0.2157396 0.02269206 10.518263
## Vm   28.0270414 3.18531987 11.365166
## err1  0.1190089 0.04452630 37.414267
## err2  0.1881154 0.05883149 31.274152
##
## # samples: 500
```

`dynmodel.mcmc()` returns a matrix of raw mcmc samples. This matrix can be further manipulated for further plots and inferences. For instances, trace plots can be easily generated by the following:

```
par(mfrow=c(4,2), mar=c(2,4,1,1))
s <- lapply(1:dim(fit)[2], function(k)
     plot(fit[,k], type="l", col="red", ylab=dimnames(fit)[[2]][k]))
```



Linear compartment models

`nlme_lin_cmpt()` fits a linear compartment model with either first order absorption, or i.v. bolus, or i.v. infusion using the estimation algorithm implemented in the 'nlme' package. A user specifies the number of compartments (up to three), route of drug administrations, and the model parameterization. `nlmixr` supports the clearance/volume parameterization and the micro constant parameterization, with the former as the default. Specification of fixed effects, random effects and intial values follows the nlme notations.

We use an extended version of the Theophiline PK data [1] accompanied with the NONMEM distribution (also an example in the nlme documentation) as an illustration of `nlme_lin_cmpt`. We model the Theophiline PK by a one-compartment model with first order absorption and with default clearance/volume parameterization. All model parameters are log-transformed; random effects are added to `KA` and `CL`.

```
dat <- read.table(system.file("examples/theo_md.txt", package = "nlmixr"), head=TRUE)
specs <- list(fixed=lKA+lCL+lV~1, random = pdDiag(lKA+lCL~1), start=c(lKA=0.5, lCL=-3.2, lV=-1))
fit <- nlme_lin_cmpt(dat, par_model=specs, ncmt=1)
```

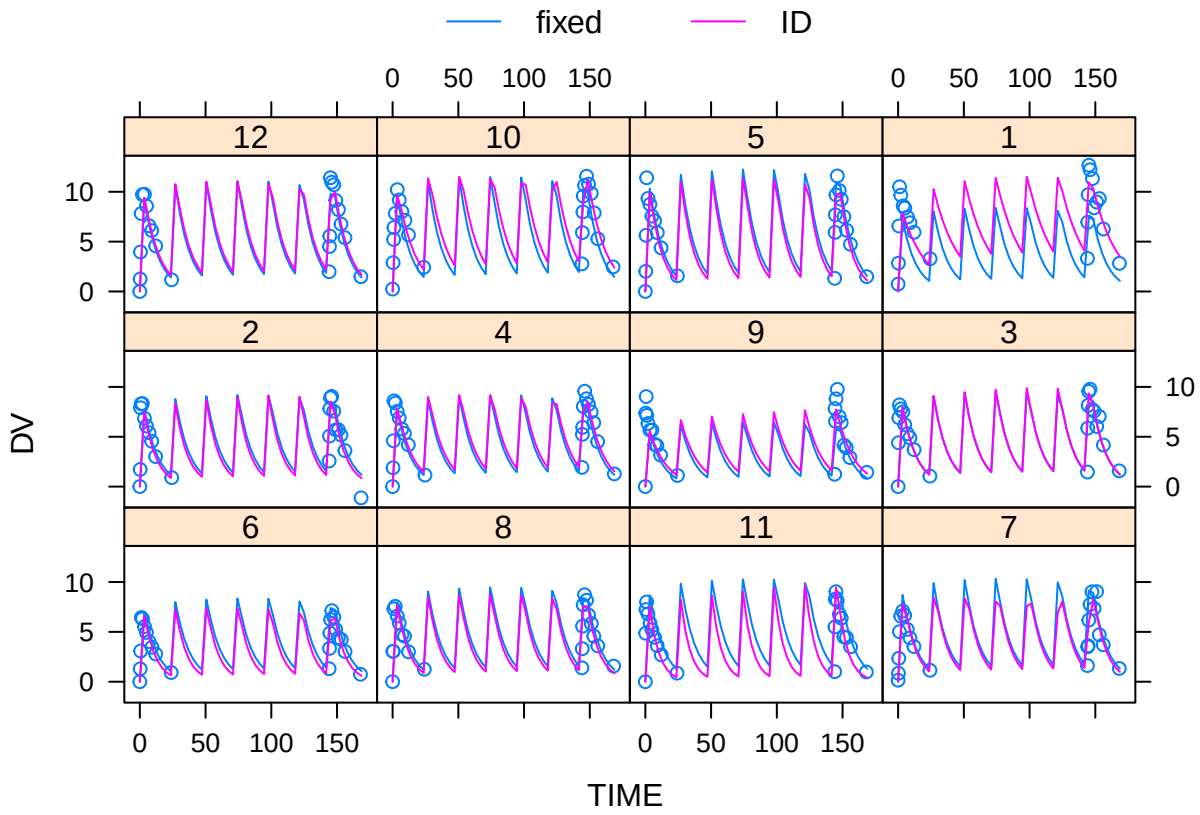
```
## Loading required package: parallel
```

```
summary(fit)
```

```
## Nonlinear mixed-effects model fit by maximum likelihood
##   Model: DV ~ ..ModList$user_fn(lCL, lV, lKA, TIME, ID)
##        AIC      BIC    logLik
##   859.5037 880.9594 -423.7518
##
## Random effects:
##  Formula: list(lKA ~ 1, lCL ~ 1)
##  Level: ID
##  Structure: Diagonal
##               lKA       lCL Residual
## StdDev: 0.4761767 0.2484193 1.065936
##
## Fixed effects: lKA + lCL + lV ~ 1
##         Value  Std.Error  DF   t-value p-value
## lKA  0.261483 0.14793530 250   1.76755  0.0784
## lCL -3.183560 0.07521599 250 -42.32557  0.0000
## lV  -0.825241 0.02591327 250 -31.84626  0.0000
##  Correlation:
##     lKA    lCL
## lCL -0.010
## lV   0.230 -0.106
##
## Standardized Within-Group Residuals:
##        Min         Q1        Med         Q3        Max
## -4.9720710 -0.3943890  0.0629013  0.4122894  2.8119726
##
## Number of Observations: 264
## Number of Groups: 12
```

[1] To demonstrate/test the capability of handling multiple doses by `nlme_lin_cmpt`, we simulated the Day 7 concentrations with a once daily (*q.d.*) regimen for 7 days, in addition to the Day 1 concentrations of the original Theophiline data.

```
plot(augPred(fit,level=0:1))
```



I.v. bolus can be specified by setting `oral=FALSE`, i.v. infusion by `oral=FALSE` and `infusion=TRUE`. To use micro-constant parameterization, one simply sets `parameterization=2`. Covariate analyses can be performed with the nlme() notations. In the following sample code, `WT` is a covariate to the log-transformed `CL` and `V`.

```
specs <- list(
    fixed=list(lKA~1, lCL+lV~WT),
    random = pdDiag(lKA+lCL~1),
    start=c(0.5, -3.2, 0, -1, 0))
fit <- nlme_lin_cmpt(dat, par_model=specs, ncmt=1)
#plot(augPred(fit,level=0:1))
#fit
```

Additional arguments/options to nlme() can be passed along via calls to `nlme_lin_cmpt`. For instance, if information on the iteration processs of optimization is of interest, one may pass `verbose=TRUE` to nlme() when calling `nlme_lin_cmpt`.

```
fit <- nlme_lin_cmpt(dat, par_model=specs, ncmt=1, verbose=TRUE)
```

Typically, nlme-defined models do not require starting values for inter-individual variance components. If you do want to specify these, the initial 'random' statement would need to be replaced with:

```
random = pdDiag(value=diag(c(2,2)), form =lKA+lCL~1)
```

where the 'value' statement specifies the starting values for the diagonal random-effects matrix in this case. The values are the square of the CV of the IIV divided by the residual error SD: with an IIV of 30% and a residual error of 20%, starting values would be (0.3/0.2)^2=2.25.

Parameterization in `nlme_lin_cmpt`

Depending on the model selection and parameterization selection, for internal calculations, `nlme_lin_cmpt` uses a particular set of parameterizations from the following list, the first three being the clearance/volume parameterizations for one-three compartments, and the last three the corresponding micro constant parameterizations. `TLAG` is excluded when `tlag=FALSE`, `KA` and `TLAG` are excluded when `oral=FALSE`.

```
pm <- list(
    c("CL", "V", "KA", "TLAG"),
    c("CL", "V", "CLD", "VT", "KA", "TLAG"),
    c("CL", "V", "CLD", "VT", "CLD2", "VT2", "KA", "TLAG"),
    c("KE", "V", "KA", "TLAG"),
    c("KE", "V", "K12", "K21", "KA", "TLAG"),
    c("KE", "V", "K12", "K21", "K13", "K31", "KA", "TLAG")
)
dim(pm)<-c(3,2)
```

Model parameters in the `par_model` argument and the parameters used for internal calculations are bridged by a function supplied to the `par_trans` argument. A user can do any parameter transformation deemed necessary within such a function, however, symbols defined in the environment of the `par_trans` function (including the formal arguments and the derived variables) have to be a superset of parameters required by a particular model with the chosen route of administration, parameterization and `tlag` flag. For instance, with `ncmt=1`, `oral=TRUE`, and `parameterization=1`, the environment of the `par_trans` function has to contain `CL`, `V` and `KA`; whereas with `ncmt=1`, `oral=TRUE`, `parameterization=2`, and `tlag=TRUE`, the environment of the `par_trans` function has to have `KE`, `V`, `KA`, and `TLAG`.

To facilitate models with the clearance/volume parameterization and the micro parameterization, `nlmixr` provides a set of predefined `par_trans` functions with log-transformed parameters of linear compartment models with different routes of administration and parameterizations. Arguments `ncmt`, `oral`, `parameterization`, and `tlag` to function `nlme_lin_cmpt` uniquely determine a proper `par_trans` function via an internal utility. Below is such a function for `ncmt=1`, `oral=TRUE`, `parameterization=1`, and `tlag=TRUE`.

```
par.1cmt.CL.oral.tlag <- function(lCL, lV, lKA, lTLAG)
{
  CL <- exp(lCL)
  V <- exp(lV)
  KA <- exp(lKA)
  TLAG <- exp(lTLAG)
}
```

With this model, a user needs to specify the fixed-effects, random-effects and initial values of the fixed effects for parameters `lCL`, `lV`, `lKA`, and `lTLAG`.

If a user perfers to parameterize a linear compartment model other than the supported parameterizations, he/she needs to write a cutomized parameterization function and supply the `par_trans` argument when calling `nlme_lin_cmpt`. Note that in the following example, the customized `par_trans` function defines `KE`, `V`, and `KA` – parameters needed for `ncmt=1`, `parameterization=2` with the default options `oral=TRUE` and `tlag=FALSE`.

```
mypar <- function(lKA, lKE, lCL)
{
    KA <- exp(lKA)
    KE <- exp(lKE)
    CL <- exp(lCL)
```

```
    V  <- CL/KE
}
specs <- list(
    fixed=lKA+lCL+lKE~1,
    random = pdDiag(lKA+lCL~1),
    start=c(0.5, -2.5, -3.2)
)
fit <- nlme_lin_cmpt(
    dat, par_model=specs,
    ncmt=1, parameterization=2, par_trans=mypar)
#plot(augPred(fit,level=0:1))
fit
```

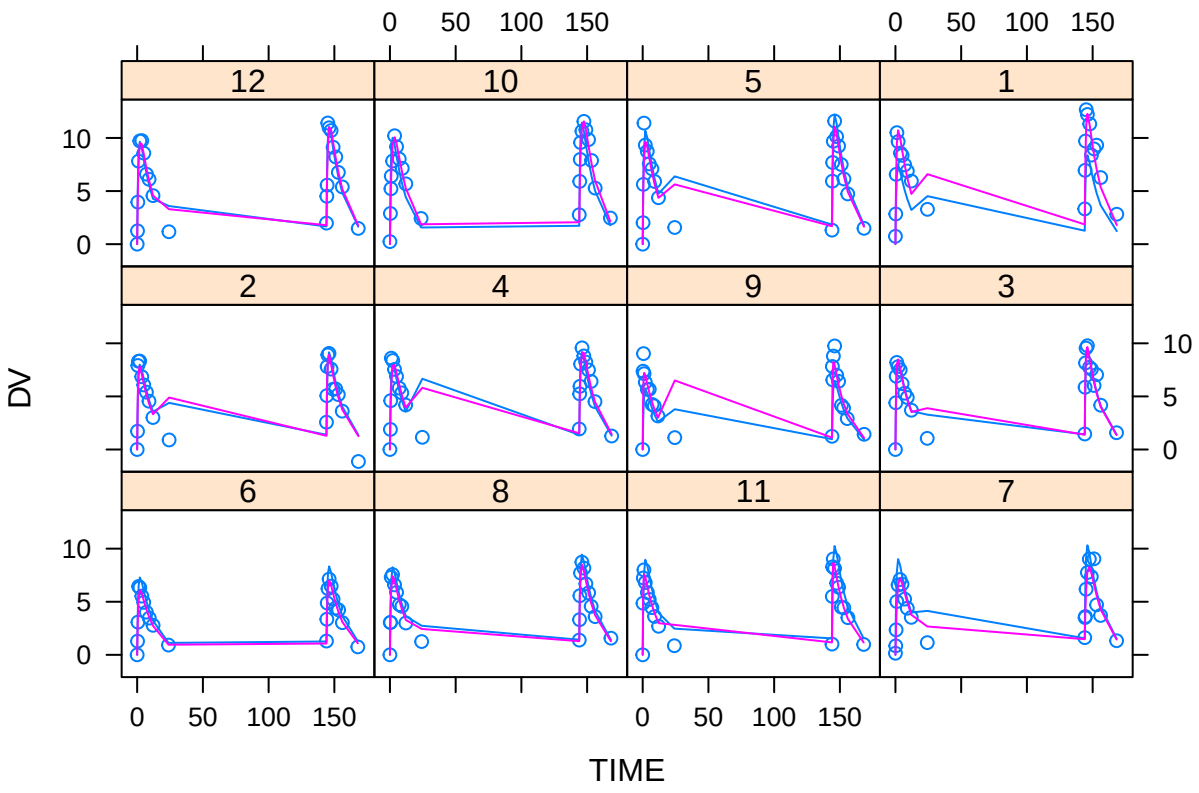
```
## Nonlinear mixed-effects model fit by maximum likelihood
##   Model: DV ~ ..ModList$user_fn(lKA, lKE, lCL, TIME, ID)
##   Log-likelihood: -415.5953
##   Fixed: lKA + lCL + lKE ~ 1
##        lKA        lCL        lKE
##  0.3049943 -3.2084427 -2.4163560
##
## Random effects:
##  Formula: list(lKA ~ 1, lCL ~ 1)
##  Level: ID
##  Structure: Diagonal
##               lKA      lCL Residual
## StdDev: 0.4681407 0.168249 1.031414
##
## Number of Observations: 264
## Number of Groups: 12
```

Models defined by ordinary differential equations

`nlme_ode()` fits a general population PKPD model defined by a set of ODEs. The user-defined dynamic system is defined in a string and provided to the `model` argument. The syntax of this mini-modeling language is detailed in the appendix. In addition to the `par_model` and `par_trans` arguments as before, a user specifies the response variable. A response variable can be any of the state variables or the derived variables in the system definition. Occasionally, the response variable may need to be scaled to match the observations. In the following example, we model the afore-mentioned Theophiline PK example by a set of ODEs. In this system, the two state variables `depot` and `centr` denote the drug amount in the absorption site and the central circulation, respetively. Observations are the measured drug concentrations in the central circulation (not the drug amount) at times. Hence, the response variable is the volume-scaled drug amount in the central circulation.

```
ode <- "
d/dt(depot) =-KA*depot;
d/dt(centr) = KA*depot - KE*centr;
"
dat <- read.table("theo_md.txt", head=TRUE)
dat$WG <- dat$WT>70
mypar <- function(lKA, lKE, lCL)
{
    KA <- exp(lKA)
```

```
    KE <- exp(lKE)
    CL <- exp(lCL)
    V  <- CL/KE
}
specs <- list(fixed=lKA+lKE+lCL~1, random = pdDiag(lKA+lCL~1), start=c(lKA=0.5, lKE=-2.5, lCL=-3.2))
fit <- nlme_ode(dat, model=ode, par_model=specs, par_trans=mypar, response="centr", response.scaler="V"
nlme_gof(fit)
```




```
fit
```

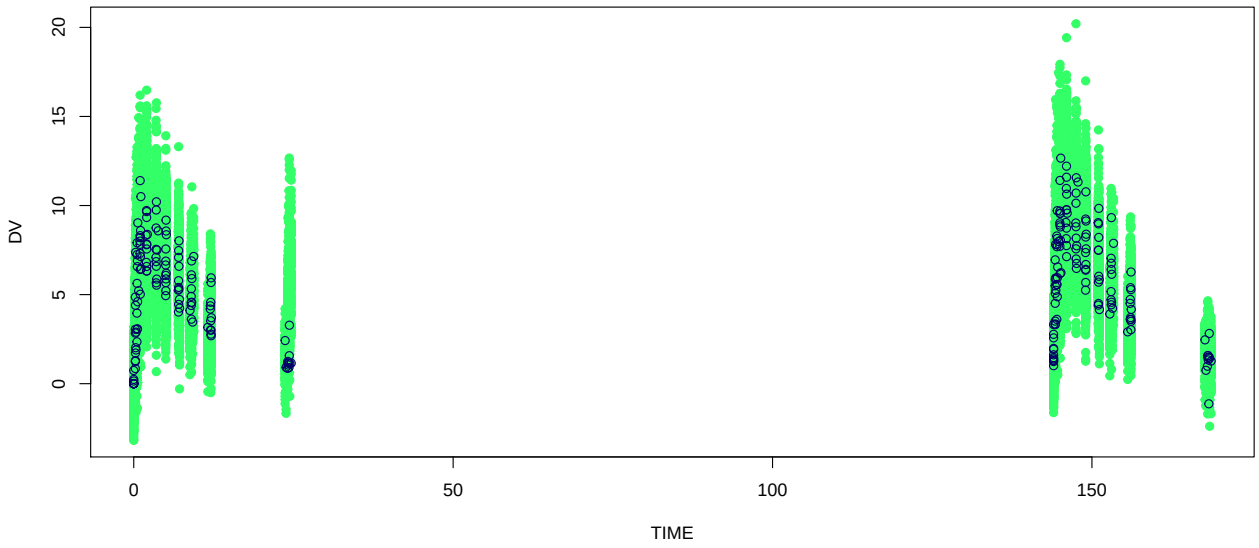
```
## Nonlinear mixed-effects model fit by maximum likelihood
##   Model: DV ~ ..ModList$user_fn(lKA, lKE, lCL, TIME, ID)
##   Log-likelihood: -415.595
##   Fixed: lKA + lKE + lCL ~ 1
##        lKA        lKE        lCL
##  0.3050693 -2.4163748 -3.2084408
##
## Random effects:
##  Formula: list(lKA ~ 1, lCL ~ 1)
##  Level: ID
##  Structure: Diagonal
##                lKA      lCL Residual
## StdDev: 0.4681084 0.168246 1.031418
##
## Number of Observations: 264
## Number of Groups: 12
```

Population modeling utilities

Visual Predictive Checks (VPC)

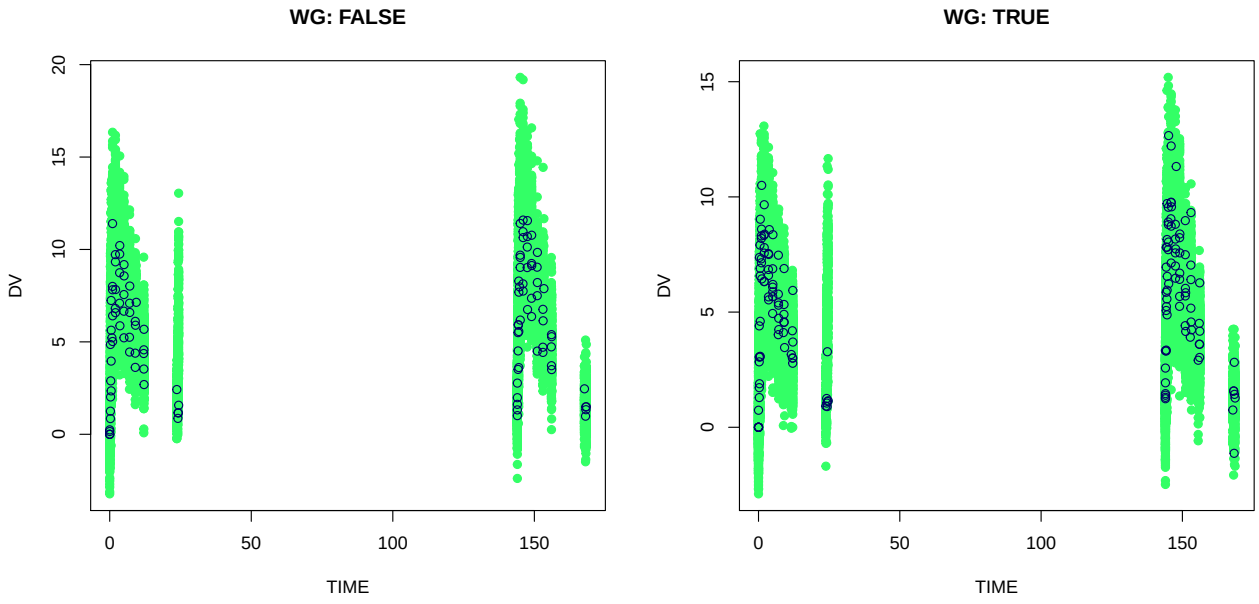
VPC plots can be produced by calling `vpc()`:

```
vpc(fit, 100)
```



Conditional VPCs can be easily generated:

```
par(mfrow=c(1,2))
vpc(fit, 100, condition="WG")
```



Bootstrap

```
dat <- read.table("theo_md.txt", head=TRUE)
specs <- list(fixed=lKA+lCL+lV~1, random = pdDiag(lKA+lCL~1), start=c(lKA=0.5, lCL=-3.2, lV=-1))
set.seed(99); nboot = 20;

cat("generating", nboot, "bootstrap samples...\n")
```

```
## generating 20 bootstrap samples...
```

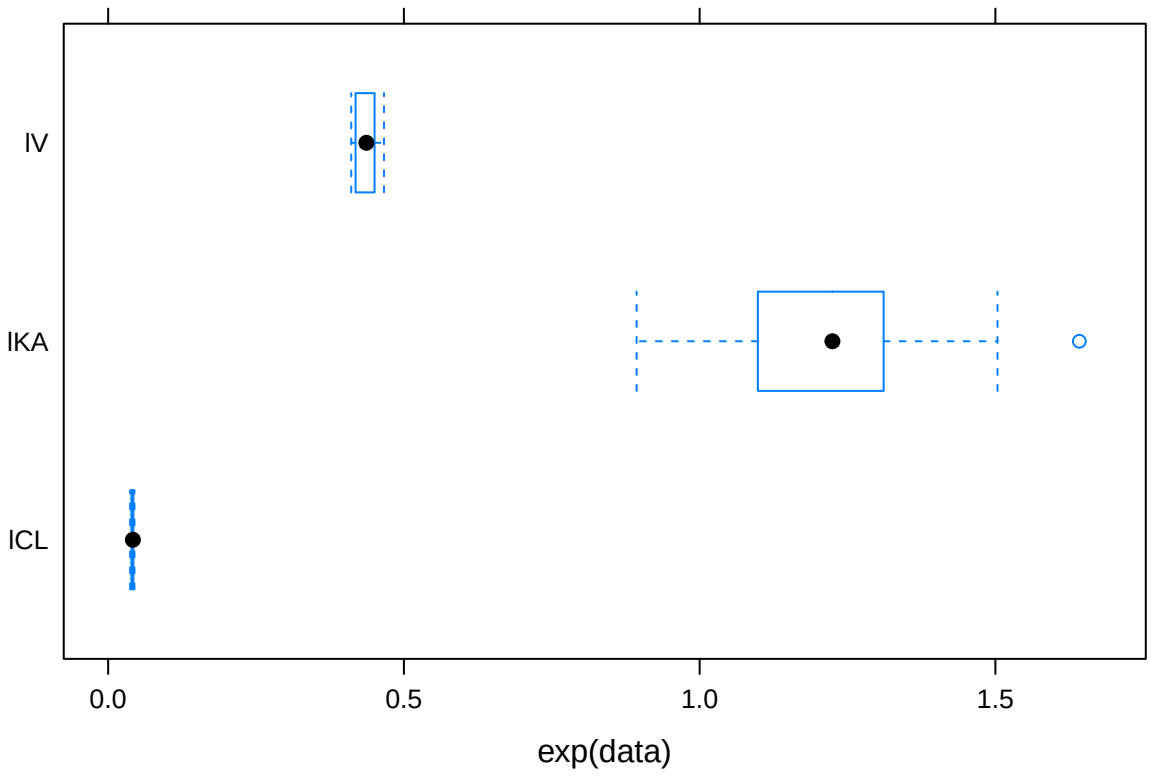
```
cmat <- matrix(NA, nboot, 3)
for (i in 1:nboot)
{
    #print(i)
    bd <- bootdata(dat)
    fit <- nlme_lin_cmpt(bd, par_model=specs, ncmt=1)
    cmat[i,] = fit$coefficients$fixed
}
dimnames(cmat)[[2]] <- names(fit$coefficients$fixed)
print(head(cmat))
```

```
##             lKA       lCL         lV
## [1,] 0.17007860 -3.159300 -0.8454881
## [2,] 0.40783953 -3.203716 -0.8235642
## [3,] 0.01032742 -3.291500 -0.8786744
## [4,] 0.19120988 -3.146474 -0.7917586
## [5,] 0.19590298 -3.169079 -0.8151188
## [6,] 0.49588394 -3.131458 -0.8080344
```

```
require(lattice)
```

```
## Loading required package: lattice
```

```
df <- do.call("make.groups", split(cmat, col(cmat)))
df$grp <- dimnames(cmat)[[2]][df$which]
print(bwplot(grp~exp(data), df))
```



Covariate selection

```
dat <- read.table("theo_md.txt", head=TRUE)
dat$LOGWT <- log(dat$WT)
dat$TG <- (dat$ID < 6) + 0    #dummy covariate

specs <- list(
    fixed=list(lKA=lKA~1, lCL=lCL~1, lV=lV~1),
    random = pdDiag(lKA+lCL~1),
    start=c(0.5, -3.2, -1))
fit0 <- nlme_lin_cmpt(dat, par_model=specs, ncmt=1)
cv <- list(lCL=c("WT", "TG", "LOGWT"), lV=c("WT", "TG", "LOGWT"))
fit <- frwd_selection(fit0, cv, dat)
```

```
## covariate selection process:
##
## adding WT to lCL : p-val = 0.2434705
## adding TG to lCL : p-val = 0.2783598
## adding LOGWT to lCL : p-val = 0.2761787
## adding WT to lV : p-val = 0.001122798
## adding TG to lV : p-val = 0.0495326
## adding LOGWT to lV : p-val = 0.002160107
##  WT added to lV
##
## adding WT to lCL : p-val = 0.4261532
## adding TG to lCL : p-val = 0.2445977
## adding LOGWT to lCL : p-val = 0.4696535
## adding TG to lV : p-val = 0.1322862
```

```
## adding LOGWT to lV : p-val = 0.007955083
##  LOGWT added to lV
##
## adding WT to lCL : p-val = 0.4593683
## adding TG to lCL : p-val = 0.2183384
## adding LOGWT to lCL : p-val = 0.4892345
## adding TG to lV : p-val = 0.09987725
##
## covariate selection finished.
```

```
print(summary(fit))
```

```
## Nonlinear mixed-effects model fit by maximum likelihood
##   Model: DV ~ ..ModList$user_fn(lCL, lV, lKA, TIME, ID)
##        AIC      BIC    logLik
##   845.8469 874.4545 -414.9234
##
## Random effects:
##  Formula: list(lKA ~ 1, lCL ~ 1)
##  Level: ID
##  Structure: Diagonal
##               lKA       lCL Residual
## StdDev: 0.4048949 0.2347384 1.037429
##
## Fixed effects: structure(list(lKA = lKA ~ 1, lCL = lCL ~ 1, lV = lV ~ 1 + WT + LOGWT), .Names = c("l
##                     Value Std.Error  DF   t-value p-value
## lKA              0.250343  0.128089 248   1.95445  0.0518
## lCL             -3.179805  0.071520 248 -44.46009  0.0000
## lV.(Intercept) -22.666028  8.481362 248  -2.67245  0.0080
## lV.WT           -0.110324  0.038267 248  -2.88298  0.0043
## lV.LOGWT         6.970555  2.630665 248   2.64973  0.0086
##  Correlation:
##                lKA    lCL    lV.(I) lV.WT
## lCL            -0.012
## lV.(Intercept) -0.010  0.004
## lV.WT          -0.008  0.003  0.996
## lV.LOGWT        0.010 -0.004 -1.000 -0.998
##
## Standardized Within-Group Residuals:
##         Min          Q1         Med          Q3         Max
## -4.95120077 -0.34388312  0.08881133  0.44882478  2.85560405
##
## Number of Observations: 264
## Number of Groups: 12
```

Stochastic Approximation Expectation-Maximization (SAEM)

`saem_fit()` fits a nonlinear mixed-effect model by the SAEM algorithm. `saem_fit()` is a compiled function that changes when the structure model changes. Before running this function, a user needs to generate a configuration list by calling the function `configsaem()`. Standard inputs to this function are: 1) a compiled saem model 2) a data.frame; 3) a list of covariates (`covar`) and residual model with additive (`res.mod=1`), proportional (`res.mod=2`) and combination of additive and proportional (`res.mod=3`); 4) initial values for the fixed effect and residual error.

```
#ode <- "d/dt(depot) =-KA*depot;
#        d/dt(centr) = KA*depot - KE*centr;"
#m1 = RxODE(ode, modName="m1")
#ode <- "C2 = centr/V;
#        d/dt(depot) =-KA*depot;
#        d/dt(centr) = KA*depot - KE*centr;"
#m2 = RxODE(ode, modName="m2")

PKpars = function()
{
  CL = exp(lCL)
  V = exp(lV)
  KA = exp(lKA)
  KE = CL / V
  xxx = 0;
  #initCondition = c(0,xxx)
}
PRED = function() centr / V
PRED2 = function() C2

saem_fit <- gen_saem_user_fn(model=lincmt(ncmt=1, oral=T))
#saem_fit <- gen_saem_user_fn(model=m1, PKpars, pred=PRED)
#saem_fit <- gen_saem_user_fn(model=m2, PKpars, pred=PRED2)


#--- saem cfg
nmdat = read.table("theo_sd.dat",  head=T)
model = list(saem_mod=saem_fit, covars="WT")
inits = list(theta=c(.05, .5, 2))
cfg   = configsaem(model, nmdat, inits)
cfg$print = 50

#cfg$Rfn = nlmixr:::Ruser_function_cmt
#dyn.load("m1.d/m1.so");cfg$Rfn = nlmixr:::Ruser_function_ode
fit = saem_fit(cfg)
```

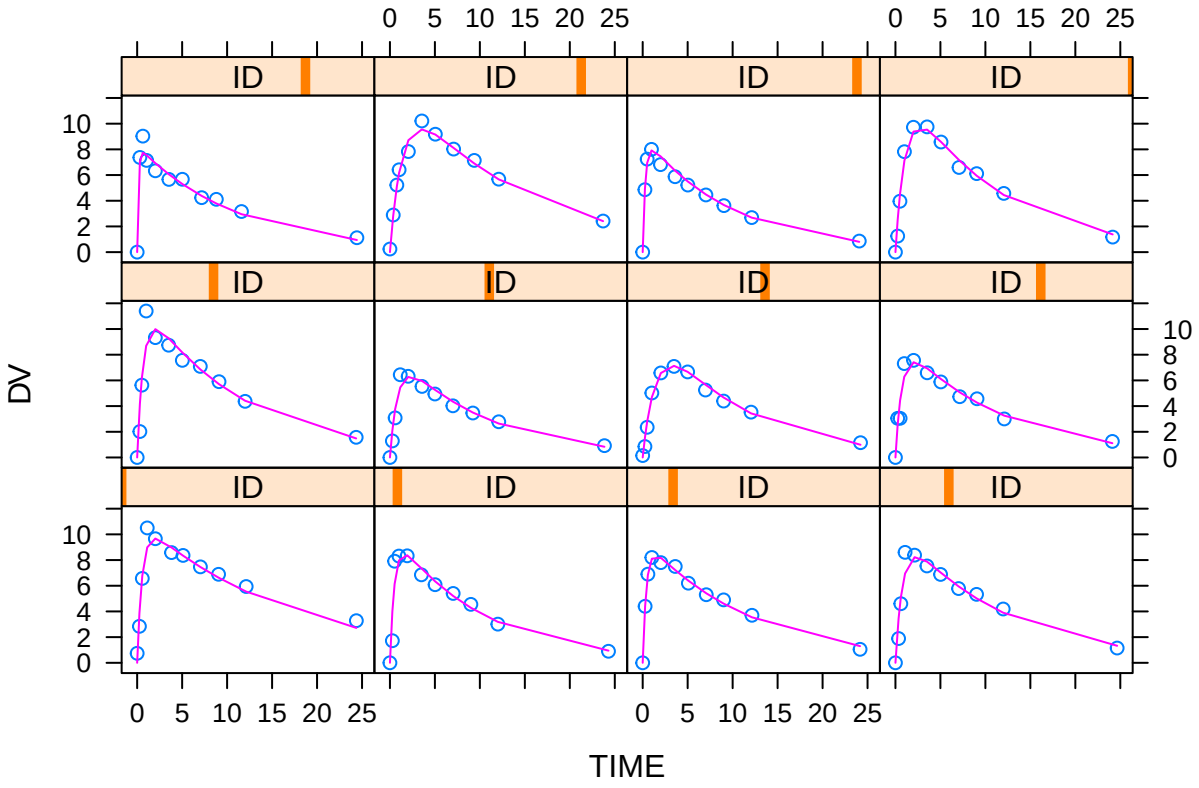
```
## 1:    -3.3578   -0.4721    0.3392    6.1542    3.8000    3.8000   14.6177
## 50:   -3.1984  -0.9246   0.3069   0.4985   0.5607   0.7941   0.5127
## 100:   -3.2131  -0.8776   0.3614   0.0628   0.4145   0.5660   0.5004
## 150:   -3.2135  -0.8746   0.3782   0.0668   0.3598   0.5493   0.5110
## 200:   -3.2224  -0.7873   0.4067   0.0677   0.0185   0.3082   0.4966
## 250:   -3.2259  -0.7767   0.4617   0.0775   0.0170   0.4208   0.4870
## 300:   -3.2201  -0.7794   0.4585   0.0742   0.0171   0.4264   0.4836
## 350:   -3.2206  -0.7806   0.4579   0.0729   0.0175   0.4268   0.4831
## 400:   -3.2178  -0.7816   0.4570   0.0722   0.0178   0.4314   0.4814
## 450:   -3.2178  -0.7810   0.4569   0.0710   0.0181   0.4285   0.4808
## 500:   -3.2176  -0.7817   0.4562   0.0709   0.0179   0.4296   0.4801
```

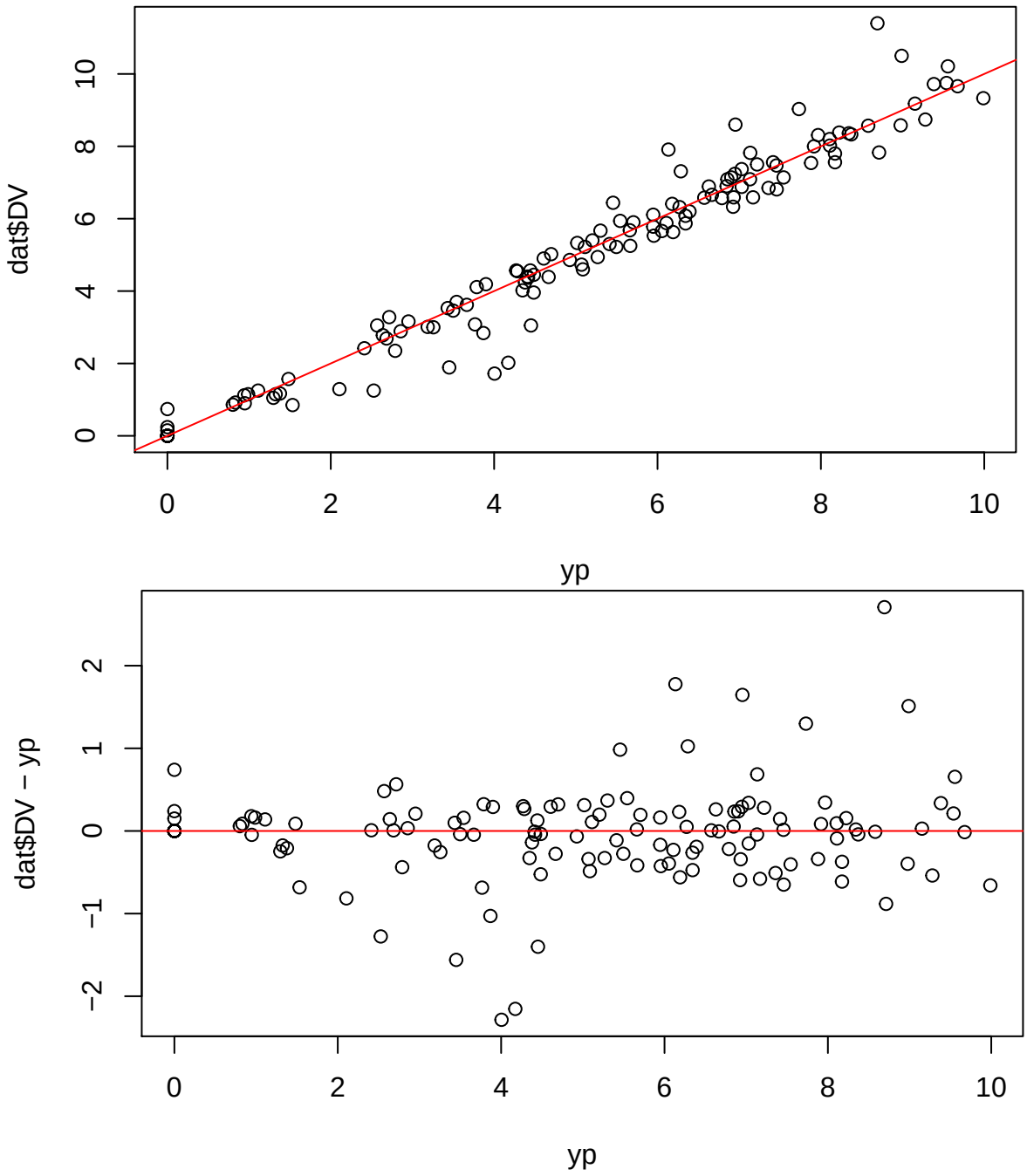
```
fit
```

```
## THETA:
##            th    log(th) se(log_th)
## CL 0.04005116 -3.2175976 0.08185220
```

```
## V  0.45760821 -0.7817419 0.04195414
## KA 1.57806680  0.4562006 0.19308887
##
## OMEGA:
##             [,1]       [,2]      [,3]
## [1,] 0.07088993 0.00000000 0.0000000
## [2,] 0.00000000 0.01785336 0.0000000
## [3,] 0.00000000 0.00000000 0.4295701
##
## SIGMA:
## [1] 0.4801045
```

```
df = plot(fit)
```





Generalized non-linear mixed-models (gnlmm)

Generalized non-linear mixed-models (gnlmm) find many useful applications in different fields, pharmacokinetics and pharmacodynamics in particular.

`gnlmm()` calculates the marginal likehood by adaptive Gaussian quadrature. For a description of this method, please find an excellent discussion in the documentation of SAS `PROC NLMIXED`.

At minimum, `gnlmm()` takes three arguments: the user-defined log-likehood function, the data frame and initial values. Initial values take the form of a named list: `THTA` for fixed effects, `OMGA` for random effect. The latter is a list of formulae; the lhs of a formula specifies the block of correlated random effects (ETAs), the

rhs of the formula gives the initial values of the lower half of the variance matrix.

Read in demo data:

```
load("/home/annie/nlmixr/vignettes/.RData")
```

Count data

This example uses the pump failure data of Gaver and O'Muircheartaigh (1987). The number of failures and the time of operation are recorded for 10 pumps. Each of the pumps is classified into one of two groups corresponding to either continuous or intermittent operation.

```
llik <- function()
{
    if (group==1) lp = THETA[1]+THETA[2]*logtstd+ETA[1]
    else          lp = THETA[3]+THETA[4]*logtstd+ETA[1]
    lam = exp(lp)
    dpois(y, lam, log=TRUE)
}
inits = list(THTA=c(1,1,1,1), OMGA=list(ETA[1]~1))

fit = gnlmm(llik, pump, inits,
    control=list(
        reltol.outer=1e-4,
        optim.outer="nmsimplex",
        nAQD=5
    )
)
```

Covariance matrix of fixed-effect parameters can be calculated with `calcCov()` after a fit.

```
cv = calcCov(fit)
cbind(fit$par[fit$nsplt==1], sqrt(diag(cv)))
```

```
##            [,1]      [,2]
## [1,]  2.9879358 1.3859686
## [2,] -0.4385056 0.7398137
## [3,]  1.8105187 0.4175574
## [4,]  0.6139161 0.5816672
```

`gnlmm()` fit matches well of `PROC NLMIXED`.

Binary data

For this example, consider the data from Weil (1970), also studied by Williams (1975), Ochi and Prentice (1984), and McCulloch (1994). In this experiment 16 pregnant rats receive a control diet and 16 receive a chemically treated diet, and the litter size for each rat is recorded after 4 and 21 days.

```
llik <- function()
{
    lp = THETA[1]*x1+THETA[2]*x2+(x1+x2*THETA[3])*ETA[1]
    p = pnorm(lp)
```

```
    dbinom(x, m, p, log=TRUE)
}
inits = list(THTA=c(1,1,1), OMGA=list(ETA[1]~1))

gnlmm(llik, rats, inits, control=list(nAQD=7))
```

```
## $par
## [1] 1.2920759 0.9574611 3.7752148 1.9313475
##
## $value
## [1] 105.296
##
## $counts
## function gradient
##       83       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

The `gnlmm()` fit closely matches the fit of `PROC NLMIXED`.

gnlmm with ODEs

```
ode <- "
d/dt(depot) =-KA*depot;
d/dt(centr) = KA*depot - KE*centr;
"
sys1 = RxODE(ode)

pars <- function()
{
    CL = exp(THETA[1] + ETA[1])#; if (CL>100) CL=100
    KA = exp(THETA[2] + ETA[2])#; if (KA>20) KA=20
    KE = exp(THETA[3])
    V  = CL/KE
    sig2 = exp(THETA[4])
}
llik <- function() {
    pred = centr/V
    dnorm(DV, pred, sd=sqrt(sig2), log=TRUE)
}
inits = list(THTA=c(-3.22, 0.47, -2.45, 0))
inits$OMGA=list(ETA[1]~.027, ETA[2]~.37)
#inits$OMGA=list(ETA[1]+ETA[2]~c(.027, .01, .37))
theo <- read.table("theo_md.txt", head=TRUE)

fit = gnlmm(llik, theo, inits, pars, sys1,
    control=list(trace=TRUE, nAQD=5))
```

```
##   Nelder-Mead direct search function minimizer
## function value for initial parameters = 834.991461
##   Scaled convergence tolerance is 0.834992
## Stepsize computed as 0.322000
## BUILD              7 902.879208 834.991461
## HI-REDUCTION       9 877.923419 834.991461
## HI-REDUCTION      11 848.107039 834.991461
## HI-REDUCTION      13 844.888781 834.991461
## HI-REDUCTION      15 841.843876 834.991461
## HI-REDUCTION      17 840.846718 834.991461
## REFLECTION        19 836.922869 833.750459
## LO-REDUCTION      21 836.830579 833.443759
## LO-REDUCTION      23 836.582343 832.689793
## HI-REDUCTION      25 835.610385 832.689793
## LO-REDUCTION      27 835.554942 832.689793
## LO-REDUCTION      29 834.991461 832.689793
## LO-REDUCTION      31 833.750459 832.689793
## HI-REDUCTION      33 833.720267 832.534662
## LO-REDUCTION      35 833.443759 832.346177
## LO-REDUCTION      37 833.370339 832.346177
## Exiting from Nelder Mead minimizer
##     39 function evaluations used
```

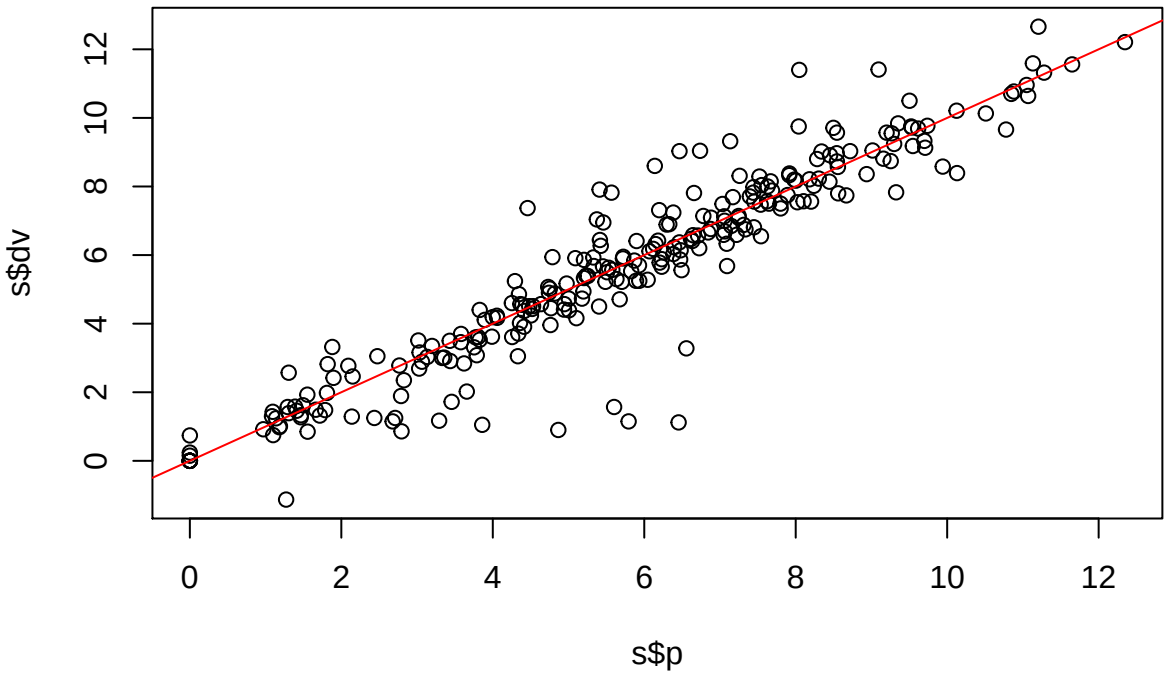
```
cv = calcCov(fit)
cbind(fit$par[fit$nsplt==1], sqrt(diag(cv)))
```

```
##              [,1]       [,2]
## [1,] -3.21535598 0.03904621
## [2,]  0.28169434 0.18194757
## [3,] -2.41704403 0.02980278
## [4,]  0.05843364 0.04566005
```

After convergence, `prediction()` can be used to calculate the prediction.

```
pred = function() {
    pred = centr/V
}

s = prediction(fit, pred)
plot(s$p, s$dv); abline(0,1,col="red")
```



References

- Lindstrom, M.J. and Bates, D.M. (1990) "Nonlinear Mixed Effects Models for Repeated Measures Data", Biometrics, 46, 673-687.
- Pinheiro, J.C. and Bates, D.M. (1996) "Unconstrained Parametrizations for Variance-Covariance Matrices", Statistics and Computing, 6, 289-296.
- Pinheiro, J.C., and Bates, D.M. (2000) "Mixed-Effects Models in S and S-PLUS", Springer.
- Sheiner and Beal (19??) "NONMEM users guide".
- Wang, W, Hallow, KM, and James, DA (2015) "A tutorial on RxODE: simulating differential equation pharmacometrics models in R", CPT: Pharmacometrics & Systems Pharmacology, to appear.
- D'Argenio DZ, Schumitzky A, and Wang X (2009). "ADAPT 5 User's Guide: Pharmacokinetic/Pharmacodynamic Systems Analysis Software".
- Neal, RM (2003) "Slice sampling" (with discussion), Annals of Statistics, vol. 31, no. 3, pp. 705-767.
- Karlsson, M. O. and Savic, R. M. (2007), Diagnosing Model Diagnostics. Clinical Pharmacology & Therapeutics, 82: 17-20.
- Gaver, D. P. and O'Muircheartaigh, I. G. (1987), "Robust Empirical Bayes Analysis of Event Rates," Technometrics, 29, 1–15.
- McCulloch, C. E. (1994), "Maximum Likelihood Variance Components Estimation for Binary Data," Journal of the American Statistical Association, 89, 330–335.
- Ochi, Y. and Prentice, R. L. (1984), "Likelihood Inference in a Correlated Probit Regression Model," Biometrika, 71, 531–543.
- Weil, C. S. (1970), "Selection of the Valid Number of Sampling Units and Consideration of Their Combination in Toxicological Studies Involving Reproduction, Teratogenesis, or Carcinogenesis," Food and Cosmetic Toxicology, 8, 177–182.
- Williams, D. A. (1975), "The Analysis of Binary Responses from Toxicological Experiments Involving Reproduction and Teratogenicity," Biometrics, 31, 949–952.
- Kuhn, E., and Lavielle, M. Coupling a stochastic approximation version of EM with a MCMC procedure. ESAIM P&S 8 (2004), 115–131.
- Kuhn, E., and Lavielle, M. Maximum likelihood estimation in nonlinear mixed effects models. Computational Statistics and Data Analysis 49 (2005), 1020–1038.

Appendix: Technical notes

RxODE Syntax

An `RxODE` model specification consists of one or more statements terminated by semi-colons, `;`, and optional comments (comments are delimited by `#` and an end-of-line marker). `NB:` Comments are not allowed inside statements.

A block of statements is a set of statements delimited by curly braces, `{ ... }`. Statements can be either assignments or conditional `if` statements. Assignment statements can be either `simple` assignments, where the left hand is an identifier (i.e., variable), or special `time-derivative` assignments, where the left hand specifies the change of that variable with respect to time, e.g., `d/dt(depot)`.

Expressions in assignment and `if` statements can be numeric or logical (no character expressions are currently supported). Numeric expressions can include the following numeric operators (`+`, `-`, `*`, `/`, `^`), and those mathematical functions defined in the C or the R math libraries (e.g., `fabs`, `exp`, `log`, `sin`). (Note that the modulo operator `%` is currently not supported.)

Identifiers in an `RxODE` model specification can refer to:

- state variables in the dynamic system (e.g., compartments in a pharmacokinetics/pharamcodynamics model);
- implied input variable, `t` (time), `podo` (oral dose, for absorption models), and `tlast` (last time point);
- model parameters, (`ka` rate of absorption, `CL` clearance, etc.);
- others, as created by assignments as part of the model specification.

Identifiers consists of case-sensitive alphanumeric characters, plus the underscore `_` character. `NB:` the dot `.` character is `not` a valid character identifier.

The values of these variables at pre-specified time points are saved as part of the fitted/integrated/solved model (see `eventTable`, in particular its member function `add.sampling` that defines a set of time points at which to capture a snapshot of the syste via the values of these variables).

The ODE specification mini-language is parsed with the help of the open source tool `DParser`, Plevyak (2015).

Dosing events

A unique feature of general PKPD modeling is the ubiquity of dosing events. When a PKPD model is defined by ODEs, the ODE solver needs to recognize these discrete events and re-start the integration process if necessary. Sheiner and Beal implemented the concept of using an integer variable `EVID` to inform the ODE solver of the nature of the current event. In addition to `EVID`, NONMEM includes several other auxiliary variables to completely and uniquely define a general dosing event: CMT, AMT, RATE, ADDL, II.

RxODE borrows the core ideas from the NONMEM implementation but uses a more compact yet somewhat convoluted format to represent the discrete dosing events.

- `EVID=0` denotes an observation event
- `EVID>0` denotes an dosing event. In general, an `EVID` in nlmixr has five digits:
  a. The right-most two digits are reserved for defining different events;
  b. The next two digits point to which state variable (or compartment) this event is applied to;
  c. The fifth digit from the right takes a value 1 if the dosing is an infusion and 0 if the dosing is a bolus.
- `AMT` is the drug amount with a bolus dosing. In case of an infusion, a positive number denotes the start of an infusion at a particular time with such an infusion rate; a negative number denotes the end of a previously started infusion.

`nlmixr` & NONMEM comparison

NONMEM functionality not supported by `nlmixr`:

- mixture model
- steady-state (SS) dosing

`nlmixr` functionality not supported by NONMEM:

- General nested random effects
- ARMA residual model
- anova() for model selection
- integrated GoF & VPC functionality
- integrated simulation with uncertainty

Calculating covariance matrix of fixed-effect parameter estimates

```
llik <- function()
{
    if (group==1) lp = THETA[1]+THETA[2]*logtstd+ETA[1]
    else          lp = THETA[3]+THETA[4]*logtstd+ETA[1]
    lam = exp(lp)
    dpois(y, lam, log=TRUE)
}
inits = list(THTA=c(1,1,1,1), OMGA=list(ETA[1]~1))

fit = gnlmm(llik, pump, inits,
    control=list(
        reltol.outer=1e-4,
        optim.outer="nmsimplex",
        nAQD=5
    )
)
```

```
cv = calcCov(fit)
Rinv = attr(cv,"RinvS")$Rinv
S    = attr(cv,"RinvS")$S
Rinv*2                  #inverse hessian matrix
```

```
##             [,1]         [,2]        [,3]         [,4]
## [1,]  1.92090906 -0.982043600  0.02346553 -0.074210340
## [2,] -0.98204360  0.547324257 -0.02948253  0.008201758
## [3,]  0.02346553 -0.029482526  0.17435419  0.077309248
## [4,] -0.07421034  0.008201758  0.07730925  0.338336769
```

```
solve(S)*4              #inverse of score function product sum
```

```
##           [,1]      [,2]       [,3]       [,4]
## [1,]  5.759594 -2.818007 0.00000000 0.00000000
## [2,] -2.818007  1.457283 0.00000000 0.00000000
## [3,]  0.000000  0.000000 0.10134670 0.02897866
## [4,]  0.000000  0.000000 0.02897866 0.31489911
```

```r
Rinv %*% S %*% Rinv      #sandwich estimate
```

```
##              [,1]        [,2]        [,3]        [,4]
## [1,]  0.69011982 -0.37466452  0.05062485 -0.07520208
## [2,] -0.37466452  0.23382290 -0.06855757 -0.01661099
## [3,]  0.05062485 -0.06855757  0.30663581  0.16745935
## [4,] -0.07520208 -0.01661099  0.16745935  0.39614278
```