Changes and FAQs for the sommer package

Giovanny Covarrubias-Pazaran

2023-05-10

The sommer package was developed to provide R users with a powerful and reliable multivariate mixed model solver. The package is focused on problems of the type $p > n$ (more effects to estimate than observations) and its core algorithm is coded in C++ using the Armadillo library. This package allows the user to specify the variance-covariance structure for the random effects, to specify heterogeneous variances, and to obtain other parameters such as BLUPs, BLUEs, residuals, fitted values, variances for fixed and random effects, etc.

Recently, I decided to code the main algorithm (Newton-Raphson & Average-Information) in C++ which encouraged me to refactor all the machinery including special functions and specification of the models. For a more in-depth explanation of how the machinery works please read the “Quick start for the sommer package” vignette by typing `vignette('sommer.start')`. Here I will focus on just making a translation of the old specification to the new specification.

The purpose of this vignette is to first show the changes in syntax for `sommer` and frequently asked question.

SECTION 1: The new syntax of sommer

1) The specification of multireponse model
2) The specification of multivariate unknown covariance structures
3) The specification of additional unknown covariance structures
4) The specification of unknown covariance structures in the residuals
5) Special models

SECTION 2: Frequently asked questions

1) I got an error similar to...
2) My model runs very slow.
3) Can I run both rrBLUP for markers and GBLUP for individuals in sommer?
4) I am missing BLUPs for individuals even when I provided them in the relationship matrix.
5) How can I use the AR1(), CS() and ARMA() functions?
6) Can I run GWAS in MET experiments with replicates?
7) How can I constrain the value of specific random effects?
8) How can I constrain two variance components to be equal?
9) I get an error when installing directly from github
10) I get an error when specifying an interaction of the form X:Z
11) I get the error “contrasts<-\texttt{(tmp, value = contr.funs[1 + isOF\[nn\]])}”
12) I get an error “Error: addition: incompatible matrix dimensions: n1xn1 and n2xn2”
13) My model only runs few iterations giving meaningless results

SECTION 1: The new syntax of sommer

1) The specification of multireponse model

In past versions (depending how old your version) there was an argument called `MVM` which had to be set to TRUE if the user wanted to run a true multi-trait model since the specification
fixed= cbind(y1,y2)-x

would by default fit 2 univariate models in parallel. That is no longer the case, the MVM argument doesn’t exist and if a model like the one above is specified it will run a true multi-trait model.

2) The specification of multivariate unknown covariance structures

In the previous versions when I introduced the multivariate solver I decided to follow the asreml syntax to specify the unknown covariance structure that needed to be estimated. For example, a diagonal model for the multitrait model, assuming a random effect called re looked something like this:

fixed= cbind(y1,y2)-x
random= - diag(trait):re

and an unstructured multitrait model was:

random= -usr(trait):re

Although this was easier for users familiar with asreml, it put a lot of limitations on the way constraints for variance components were specified. The same model in the new versions looks like this:

random= - var(re, Gtc=unsm(2))

where the Gtc argument helps us to indicate what type of structure this random effect represents. Here I specified an unstructured model with the function unsm() with a number 2 for 2 traits. The user can specify either diag() or unc() or any customized matrix with dimensions t x t (t being the number of traits) containing the number 0,1,2,3 that specify the constraint:

- 0: not to be estimated
- 1: estimated and constrained to be positive (i.e. variance component)
- 2: estimated and unconstrained (can be negative or positive, i.e. covariance component)
- 3: not to be estimated but fixed (value has to be provided in the Gti argument)

All these models fit a model with the following variance for re:

\[ \text{var}(u) = T \otimes A \]

where:

\[
\text{var}(u) = \begin{bmatrix}
\sigma^2_{g_{1,1},e_1} & \sigma_{g_{1,1},e_2} & \sigma_{g_{1,1},e_3} \\
\sigma_{g_{2,1},e_1} & \sigma^2_{g_{2,1},e_2} & \sigma_{g_{2,1},e_3} \\
\sigma_{g_{3,1},e_1} & \sigma_{g_{3,1},e_2} & \sigma^2_{g_{3,1},e_3}
\end{bmatrix} \otimes A
\]

By making this change now, the user has full control of the constraints applied to the estimation of variance components and can provide initial values easily using the Gti argument.

3) The specification of additional unknown covariance structures

If we focus for a moment on a univariate mixed model we can also have other unknown covariance structures specified.

\[ \text{var}(u) = E \otimes... \otimes F \otimes A \]

where:

\[
\text{var}(u) = \begin{bmatrix}
\sigma^2_{g_{1,1},e_1} & \sigma_{g_{1,1},e_2} & \sigma_{g_{1,1},e_3} \\
\sigma_{g_{2,1},e_1} & \sigma^2_{g_{2,1},e_2} & \sigma_{g_{2,1},e_3} \\
\sigma_{g_{3,1},e_1} & \sigma_{g_{3,1},e_2} & \sigma^2_{g_{3,1},e_3}
\end{bmatrix} \otimes... \otimes \begin{bmatrix}
\sigma^2_{g_{1,1},f_1} & \sigma_{g_{1,1},f_2} \\
\sigma_{g_{2,1},f_1} & \sigma^2_{g_{2,1},f_2}
\end{bmatrix} \otimes A
\]
If we think about the multi trait model, this is very similar but with an additional kroneker product for the multivariate version:

\[
var(u) = T \otimes E \otimes \cdots \otimes F \otimes A
\]

where:

\[
var(u) = \begin{bmatrix}
\sigma_{g_{1,1}}^2 & \sigma_{g_{1,1},g_{1,2}} \\
\sigma_{g_{2,1}} & \sigma_{g_{2,1},g_{2,2}}
\end{bmatrix} \otimes \begin{bmatrix}
\sigma_{g_{1,1}}^2 & \sigma_{g_{1,1},g_{1,2}} & \sigma_{g_{1,1},g_{1,3}} \\
\sigma_{g_{2,1}} & \sigma_{g_{2,1},g_{2,2}} & \sigma_{g_{2,1},g_{2,3}} \\
\sigma_{g_{3,1}} & \sigma_{g_{3,1},g_{3,2}} & \sigma_{g_{3,1},g_{3,3}}
\end{bmatrix} \otimes \cdots \otimes \begin{bmatrix}
\sigma_{g_{1,1}}^2 & \sigma_{g_{1,1},g_{1,2}} \\
\sigma_{g_{2,1}} & \sigma_{g_{2,1},g_{2,2}} \\
\sigma_{g_{3,1}} & \sigma_{g_{3,1},g_{3,2}}
\end{bmatrix} \otimes A
\]

Getting back to the point—the additional unknown covariance structures besides the multi-trait (T) before were specified with asreml syntax. For example a univariate diagonal and unstructured model, assumed a random effect called \textit{id} representing the treatments planted in different environments coded in a second random effect called \textit{env}. The model used to look like:

\[
\text{fixed} = y1-x \\
\text{random} = ~ \text{diag(env)}:\text{id} \\
\text{or random} = ~ \text{usr(env)}:\text{id}
\]

and now it would be specified as:

\[
\text{fixed} = y1-x \\
\text{random} = ~ \text{vsr(dsr(env),id)} \\
\text{or random} = ~ \text{vsr(usr(env),id)}
\]

where the \textit{dsr()} and \textit{usr()} functions specify diagonal and unstructured models respectively. Now \textit{csr()} for a customized structure is available. The main gain from having changed the formulation is that the new specification through the \textit{vsr()} function allows for constructing more complex models. For example, assume individuals specified in a column called \textit{id} tested in three environments in a column called \textit{env} measured at two different time points specified in a column called \textit{time}. We may want something more flexible than:

\[
\text{fixed} = y1-x \\
\text{random} = ~ \text{id}
\]

We could actually assume that individuals are correlated within the environments for the different time points but want to consider environments independent. The variance for such random effects is the following:

\[
var(u) = \begin{bmatrix}
\sigma_{g_{1,1}}^2 & \sigma_{g_{1,1},g_{1,2}} & \sigma_{g_{1,1},g_{1,3}} \\
\sigma_{g_{2,1}} & \sigma_{g_{2,1},g_{2,2}} & \sigma_{g_{2,1},g_{2,3}} \\
\sigma_{g_{3,1}} & \sigma_{g_{3,1},g_{3,2}} & \sigma_{g_{3,1},g_{3,3}}
\end{bmatrix} \otimes \begin{bmatrix}
\sigma_{g_{1,1}}^2 & \sigma_{g_{1,1},g_{1,2}} \\
\sigma_{g_{2,1}} & \sigma_{g_{2,1},g_{2,2}} \\
\sigma_{g_{3,1}} & \sigma_{g_{3,1},g_{3,2}}
\end{bmatrix} \otimes A
\]

which was not possible in previous versions of sommer and now can be specified as:

\[
\text{random} = ~ \text{vsr(usr(env),dsr(time),id)}
\]

and the same logic can be extended to as many interacting factors as desired.

4) The specification of unknown covariance structures in the residuals

Previously, sommer was limited to only diagonal models in the residuals (unstructured available only for multi-trait before). Now all the same applications discussed for the random term also apply for the residual term. Just keep in mind that the residual term is always called \textit{units}.

Previous versions:

\[
\text{random} = ~ \text{diag(trait):diag(env):units} \\
\text{random} = ~ \text{usr(trait):diag(env):units} # \text{limit}
\]
New versions (>3.7):

```r
random = ~ vsr(dsr(env), units, Gtc=mm) ## can be extended to more interacting factors
random = ~ vsr(usr(env), units, Gtc=mm) ## can be extended to more interacting factors
random = ~ vsr(at(env), units, Gtc=mm) ## can be extended to more interacting factors
random = ~ vsr(csr(env), units, Gtc=mm) ## can be extended to more interacting factors
```

where `mm` can be any matrix specifying the type of multi-trait model (constraints). For example we could use `unsm()` `diag()`, `uncm()` or any other customized matrix.

5) Special models

In previous versions the use of asreml formulation really limited the expansion of sommer to more sophisticated models. Now there are many more possible models.

Previous versions:

### Overlay models

**Previous version:** limited to 2 columns and only random and no covariance structures.

```r
random = ~ x + and(y)
```

New versions (>3.7): in theory there are no limits. Can be extended to more interacting factors in the unknown covariance structures and can overlay as many columns as needed. Plus is fully functional with the multivariate models.

```r
random = vsr(..., overlay(x1,...,xn), Gtc=mm)
```

### Random regression models

**Previous version:** Not available before

New versions (>3.7): in theory no limits. Can be extended to more interacting factors in the unknown covariance structures and only requires the use of the `leg()` function. Plus is fully functional with the multivariate models.

```r
random = vsr(usr(leg(v,1)),x)
random = vsr(dsr(leg(v,1)),x)
random = vsr(leg(v,1),x)
```

### GWAS models

**Previous version:** Only univariate models available

New versions (>3.7): all the power of the `mmer()` function is available plus you can fit multivariate GWAS models. See details in the `sommer.start` vignettes.

### Spatial models

**Previous version:** It was called directly in the formula

```r
random = spl2D(Row,Col,at=?)
```

New versions (>3.7): It has to be called within the `vsr()` function but now it can be combined with all the unknown covariance structures available.

```r
random = vsr(...,spl2D(Row,Col,at=?), Gtc=mm) # being mm any multi-trait constraint-structure.
```
Customized random effects  Previous version: It was provided in the grouping argument

\[
\text{random} = \text{grp}(x), \\
\text{grouping} = \text{list}(x=Z)
\]

New versions (>3.7): It has to be called within the \texttt{vsr()} function but now it can be combined with all the unknown covariance structures available.

\[
\text{random} = \text{vsr}(\ldots, Z, \text{Gtc=mm}) \# \text{mm is any multi-trait constraint-structure.}
\]

SECTION 2: Frequently asked questions

1) I got an error similar to:

```
# iteration LogLik wall cpu(sec) restrained 
# 1 -224.676 18:11:23 3 0 
# Sistem is singular. Aborting the job. You could try a bigger tolparinv value.
```

This error indicates that your model is singular (phenotypic variance V matrix is not invertible) and therefore the model is stopped, throwing the error message and returning an empty list. You can try a simpler model or just modify the argument \texttt{tolparinv} in the \texttt{mmmer()} function. The default is 1e-3, which means that it will try to invert V and if it fails it will try to add a small value to the diagonal of V of 1e-3 to make it invertible and try bigger and bigger numbers. If this fails then the program will return the empty list.

Sometimes the model becomes singular when you use variance covariance matrices (i.e. genomic relationship matrices) that are not full-rank. You can try to make it full-rank and try again.

2) My model runs very slow

Keep in mind that sommer uses direct inversion (DI) algorithm which can be very slow for large datasets. The package is focused on problems of the type \( p > n \) (more random effect levels than observations) and models with dense covariance structures. For example, for an experiment with dense covariance structures with low-replication (i.e. 2000 records from 1000 individuals replicated twice with a covariance structure of 1000x1000) sommer will be faster than MME-based software. Also for genomic problems with large number of random effect levels, i.e. 300 individuals (n) with 100,000 genetic markers (p). For highly replicated trials with small covariance structures or \( n > p \) (i.e. 2000 records from 200 individuals replicated 10 times with covariance structure of 200x200) asreml or other MME-based algorithms will be much faster and we recommend you to opt for those software.

3) Can I run both; rrBLUP for markers and GBLUP for individuals in sommer?

Both types of models can be fitted in sommer. Please see the vignette #1.

4) I am missing BLUPs for individuals even when I provided them in the relationship matrix

I got this good question in the past: “when I want to fit an animal model with the sommer package using an additive relationship matrix(A), this A matrix would contain parents. But the random effects only contains animals in the random effect but not including parents in the A matrix. How can I get the random effects for parents?”

Answer: The easy way to do it is to make sure that even if the parents don’t show up in the dataset, you need to make sure that they are present in the levels of the column that contains the individuals (i.e. animal
IDs), in addition they have to be provided in the relationship matrix and that’s it. They should be returned in the blups.

library(sommer)

data(DT_cpdata)

DT <- DT_cpdata
GT <- GT_cpdata
MP <- MP_cpdata

### create the variance-covariance matrix
A <- A.mat(GT) # additive relationship matrix

### look at the data and fit the model
set.seed(12)

DT2 <- droplevels(DT[sample(1:nrow(DT),100),]) # we simulate a dataset with only 100 animals
nrow(DT2); length(levels(DT2$id))

## [1] 100
## [1] 100

# we fit a model with the reduced dataset where only 100 blups will be returned since only # 100 levels exist in the "id" column
mix1 <- mmer(Yield~1,
random=-vsr(id,Gu=A)
+ Rowf + Colf,
rcov=-units,
data=DT2, verbose = FALSE)

## Adding additional levels of Gu in the model matrix of 'id'

summary(mix1)

# Multivariate Linear Mixed Model fit by REML
# ********************** sommer 4.2 **********************
# logLik AIC BIC Method Converge
# Value -47.00674 96.01348 98.61865 NR TRUE

# Variance-Covariance components:
# VarComp VarCompSE Zratio Constraint
# u:id.Yield-Yield 1531.7 1000.9 1.530 Positive
# Rowf.Yield-Yield 157.1 297.4 0.528 Positive
# Colf.Yield-Yield 0.0 396.4 0.000 Positive
# units.Yield-Yield 3358.4 883.6 3.801 Positive

# Fixed effects:
# Trait Effect Estimate Std.Error t.value
# 1 Yield (Intercept) 127.4 7.214 17.66

# Groups and observations:
# Yield
# u:id 363
# Rowf 13
# Colf 35
# Use the '$' sign to access results and parameters
As of 4.1.2 this shouldn’t be a problem since internally the `mmer()` solver adds the missing levels, but we leave this for reference for people using older versions of sommer.

5) **How can I use the AR1(), CS() and ARMA() functions**

Sommer doesn’t support the estimation of additional correlation components like AR1 in the way asreml does. Still, if the user knows the correlation value or can do an iterative approach to find the best value then these functions can be used to specify the variance covariance structure for a given random effect.

For example, in the DT_rpdata dataset we have a field with row and column coordinates. This allows fitting row and column as random effects:
library(sommer)
data(DT_cpdata)
DT <- DT_cpdata
mix1 <- mmer(Yield ~ 1,
             random = Rowf + Colf,
             rcov = units,
             data = DT, verbose = FALSE)
summary(mix1)$varcomp

## VarComp VarCompSE Zratio Constraint
## Rowf.Yield-Yield 832.2879 393.8951 2.112968 Positive
## Colf.Yield-Yield 153.9201 126.7582 1.214281 Positive
## units.Yield-Yield 3647.3486 290.4910 12.555804 Positive

If the user wants to relax the independence between rows and define an AR1 covariance structure among
rows then the model could be fitted as:

library(sommer)
data(DT_cpdata)
DT <- DT_cpdata
mixAR1row <- mmer(Yield ~ 1,
                   random = vsr(Rowf, Gu = AR1(Rowf, rho = 0.3)) + Colf,
                   rcov = units,
                   data = DT, verbose = FALSE)
summary(mixAR1row)$varcomp

## VarComp VarCompSE Zratio Constraint
## u:Rowf.Yield-Yield 791.8219 387.8695 2.041465 Positive
## Colf.Yield-Yield 154.5660 134.2703 1.188855 Positive
## units.Yield-Yield 3643.6027 290.1689 12.556834 Positive

The same could be done for the column random effect:

library(sommer)
data(DT_cpdata)
DT <- DT_cpdata
mixAR1col <- mmer(Yield ~ 1,
                   random = Rowf + vsr(Colf, Gu = AR1(Colf, rho = 0.3)),
                   rcov = units,
                   data = DT, verbose = FALSE)
summary(mixAR1col)$varcomp

## VarComp VarCompSE Zratio Constraint
## Rowf.Yield-Yield 830.3623 392.8264 2.113815 Positive
## u:Colf.Yield-Yield 178.7490 134.2703 1.331262 Positive
## units.Yield-Yield 3624.6074 287.6072 12.602629 Positive

If on the other hand, you would like to model the presence of correlation in row and columns at the same
time the model would look like this:

library(sommer)
data(DT_cpdata)
DT <- DT_cpdata
mixAR1rowcol <- mmer(Yield ~ 1,
                      random = vsr(Rowf:Colf,
                                   Gu = kronecker(AR1(Rowf, rho = 0.3), AR1(Colf, rho = 0.3), make.dimnames = TRUE))
                      ,
                      rcov = units,
                      data = DT, verbose = FALSE)
summary(mixAR1rowcol)$varcomp
rcov=units, data=DT, verbose = FALSE)

summary(mixAR1rowcol)$varcomp

## VarComp VarCompSE Zratio Constraint
## units.Yield-Yield 2025.584 622.1023 3.256030 Positive

Notice that if you specify a random effect that is the interaction between 2 random effects the covariance structure to be specified in the Gu argument has to be built using the \texttt{kron}ecker() function. The same applies to the \texttt{ARMA()} and \texttt{CS()} functions. Please keep in mind that the correlation values (rho argument) is a fixed value not estimated by REML like asreml does but you can always follow an iterative approach.

6) Can I run GWAS in MET experiments with replicates?

Yes, please see the vignette for QG using sommer and the method of GWAS by GBLUP.

7) How can I constrain the value of specific random effects?

When using the \texttt{vsr()} function three additional arguments help to control the following:

- \texttt{Gu:} matrix for covariances among levels for the \texttt{u.th} random effect
- \texttt{Gti:} matrix of initial values for the variance-covariance components
- \texttt{Gtc:} matrix of constraints for the variance-covariance components

Since each random effect can be seen as a multi-trait variance covariance structure, the univariate models are just an extension where the multi-trait variance covariance structure is a 1 x 1 matrix. When inspecting the results for the mixed models fitted by the \texttt{mmer()} function corresponding to the variance components stored in the sigma element, you will notice that each random effect contains a t x t matrix which corresponds to the multi-trait structure we referred to. For example:

```r
data(DT_cpdata)
DT <- DT_cpdata
GT <- GT_cpdata
MP <- MP_cpdata

### create the variance-covariance matrix
A <- A.mat(GT) # additive relationship matrix

### look at the data and fit the model
mix1 <- mmer(Yield~1, 
  random=-vsr(id,Gu=A), 
  rcov=-units, 
  data=DT, verbose = FALSE)

mix1$sigma`
```

## Yield
## Yield 650.4145

Here the sigma element contains the random effects for \texttt{id} and \texttt{units} (error). Each of these contains a matrix of 1 by 1, but in a multi trait model would look like this:

```r
data(DT_cpdata)
DT <- DT_cpdata
GT <- GT_cpdata
MP <- MP_cpdata

### create the variance-covariance matrix
```
A <- A.mat(GT)  # additive relationship matrix
### look at the data and fit the model

mix2 <- mmer(cbind(Yield,color)~1,
  random=vsr(id,Gu=A, Gtc=unsm(2)),
  rcov=vsr(units,Gtc=diag(2)),
  data=DT, verbose = FALSE)

mix2$sigma`u:id`

## Yield color
## Yield 634.6295932 0.471715518
## color 0.4717155 0.005126228

Notice that for 2 traits this becomes a 2 by 2 matrix. In order to put constraints in some of these random
effect matrices you can use the Gtc argument as show above. For the id random effect we have specified
that variance components should be estimated and be positive (diagonals with a 1), whereas covariance
components should be estimated and unconstrained to be positive or negative (off-diagonals with a 2).

unsm(2)

## [,1] [,2]
## [1,] 1 2
## [2,] 2 1

mix2$sigma`u:id`

## Yield color
## Yield 634.6295932 0.471715518
## color 0.4717155 0.005126228

On the other hand, for the units (error) random effect we have specified in the Gtc argument that variance
components should be estimated and be positive (diagonals), whereas covariance components should not be
estimated (off-diagonals with a 0)

diag(2)

## [,1] [,2]
## [1,] 1 0
## [2,] 0 1

mix2$sigma`u:units`

## Yield color
## Yield 4009.336 0.000000000
## color 0.000 0.002563711

If the user would like to constrain a value to be fixed and not change through the estimation process of other
random effects the user needs to provide the initial value (scaled with respect to the error variance) of those
variance-covariance components in the Gti argument and use a matrix with the value 3 in the Gtc constraint
matrix.

mm <- matrix(3,1,1)  # matrix to fix the var comp
initialVal <- mix1$sigma_scaled`u:id`/2  # we want to fix the vc to be half of the previous univariate

mix3 <- mmer(Yield~1,
  random=vsr(id, Gu=A, Gti=initialVal, Gtc=mm),  # constrained
  rcov=var(units), # unconstrained
  data=DT, verbose = FALSE)
# analyze variance components
summary(mix1)$varcomp

## VarComp VarCompSE Zratio Constraint
## u:id.Yield-Yield 650.4145 325.5562 1.997856 Positive
## units.Yield-Yield 4031.0153 344.6051 11.697493 Positive

summary(mix3)$varcomp

## VarComp VarCompSE Zratio Constraint
## u:id.Yield-Yield 325.2072 259.0889 1.255196 Fixed
## u:units.Yield-Yield 4051.3786 352.4804 11.493912 Positive

For the vsc() function used in mmec() for the c x c problem the theta and thetaC arguments can be used directly in the covariance structures. For example to fix the residuals equal to 1:

```r
mm <- matrix(3,1,1) ## matrix to fix the var comp
vei <- var(DT$Yield, na.rm = TRUE) # we want to fix the vc to be half of the previous univariate model

mix3 <- mmec(Yield ~ 1, random=~Rowf, # unconstrained
              rcov= ~ vsc(isc(units, thetaC=mm, theta=matrix(1/vei,1,1))), # constrained
data=DT, verbose = FALSE)
```

# analyze variance components
summary(mix3)$varcomp

## VarComp VarCompSE Zratio Constraint
## Rowf:isc:isc 976.0743 8.916297e-16 1.094708e+18 Positive
## units:mm:vei: 1.0000 1.227629e-03 8.145785e+02 Fixed

8) How can I constrain two variance components to be equal?

Sometimes the built-in capacities of sommer are not flexible enough to do all what users want. One of those situations is to fix variance components to be equal. Let’s simulate some multitrait data:

```r
library("MASS") ## needed for mvrnorm
n <- 100
mu <- c(1,2)
Sigma <- matrix(c(10,5,5,10),2,2)
Y <- mvrnorm(n,mu,Sigma); colnames(Y) <- c("y1","y2")
## this simulates multivariate normal rvs
y <- as.vector(t(Y))
df1 <- data.frame(Y)
df2 <- data.frame(y)
```

Now lets assume that we want to fit a multitrait model with an unstructured error variance structure with the built-in capacity. That would be as easy as this:

```r
mix1 <- mmer(cbind(y1,y2)-1, rcov=~vsr(units, Gtc=unsm(2)), data=df1, verbose = FALSE)
mix1$sigma
```

```r
## $u:units`
## y1 y2
## y1 9.902536 5.543521
## y2 5.543521 10.125877
```
But now assume that you would like to constrain the variance of \(y_1\) and \(y_2\) to be equal. This requires the user to take a different approach. The user can build externally the multitrait matrices and fit them directly. Let’s do this to recreate the exact same result as using the built-in capabilities:

\[
X \leftarrow \text{kronecker}(\text{rep}(1,n), \text{diag}(1,2))
\]

\[
V_1 \leftarrow \text{matrix}(\text{c}(1,0,0,0), 2, 2)
\]

\[
V_2 \leftarrow \text{matrix}(\text{c}(0,0,0,1), 2, 2)
\]

\[
V_3 \leftarrow \text{matrix}(\text{c}(0,1,1,0), 2, 2)
\]

\[
sig_1 \leftarrow \text{kronecker}(\text{diag}(1,n), V_1) \quad \# \text{ variance component 1}
\]

\[
sig_2 \leftarrow \text{kronecker}(\text{diag}(1,n), V_2) \quad \# \text{ variance component 2}
\]

\[
gam \leftarrow \text{kronecker}(\text{diag}(1,n), V_3) \quad \# \text{ covariance component}
\]

# now fit the model

\[
mix_2 \leftarrow \text{mer}(y \sim X - 1, \text{rcov} = \text{~vsr}(\text{sig}_1) + \text{vsr}(\text{sig}_2) + \text{vsr}(\text{gam}, G_\text{ti} = \text{matrix}(.15)), \text{data}=\text{df2}, \text{verbose} = \text{FALSE})
\]

\[
mix_2$sigmaVector
\]

## u:sig1.y-y u:sig2.y-y u:gam.y-y
## 9.902350 10.125683 5.543179

Notice that we fitted a univariate model but we built the kernels and fitted those kernels one by one.

Now we will constrain the two variance components to be equal. This is done the following way:

\[
sig \leftarrow \text{sig}_1 + \text{sig}_2
\]

\[
mix_3 \leftarrow \text{mer}(y \sim X - 1, \text{rcov} = -\text{vsr}(\text{sig}) + \text{vsr}(\text{gam}, G_\text{ti} = \text{matrix}(.15)), \text{data}=\text{df2}, \text{nIters}=30, \text{verbose} = \text{FALSE})
\]

\[
mix_3$sigmaVector
\]

## u:sig.y-y u:gam.y-y
## 10.014017 5.543179

9) I get an error when installing directly from github

Some people experience issues when trying to install the newest version of sommer from GitHub in their mac computer using the following command line:

```
devtools::install_github('covaruber/sommer')
```

one of the error messages that have been identified by users is related to the installation of the gfortran compiler. If you have that issue, the information of the following website may solve your instalation problem:

[https://www.cynkra.com/blog/2021-03-16-gfortran-macos/](https://www.cynkra.com/blog/2021-03-16-gfortran-macos/)

10) I get an error when specifying an interaction of the form X:Z

If you get an error message similar to the following when specifying an interaction in your model:

```
Error in X:Z : NA/NaN argument
```

Please check that your variables in the interaction are all of class factor. Making all factors using the `as.factor()` function should fix your error.

11) I get the error: `contrasts<-(*tmp*, value = contr.funs[1 + isOF[nn]])`

If you get an error message similar to the following when fitting your model:
12) I get an error “Error: addition: incompatible matrix dimensions: n1xn1 and n2xn2”

This is very likely because you’re fitting a different residual variance for different levels of a random variable. Please make sure you sort your dataset by the variables you’re fitting the residuals at. For example:

DT=DT[with(DT, order(Env)),]

sorts a data set named DT by levels of a variable named Env. That way a model with a different residual variance for each level of Env can be fitted as:

rcov ~ vsc(dsc(Env), isc(units))

13) My mmec model only runs few iterations giving meaningless results

This can happen when the units of your trait being modeled are either too small or too big. Try scaling the units of your trait so the differences between the levels is not in either too big (var>10,000) or too small units (var < 0.01). Alternatively, modify the argument tolParInv to a smaller value like 1e-8 or higher. Last and probably the best since is guaranteed to work is, scale your trait using the scale() function.

Literature


Covarrubias-Pazaran G. 2018. Software update: Moving the R package sommer to multivariate mixed models for genome-assisted prediction. doi: https://doi.org/10.1101/354639


