cirad
AGRICULTURAL RESEARCH FOR DEVELOPMENT

Lab Excercises - breedR

Facundo Muñoz*

Orléans, September 18, 2018

0. Setup

Start by loading all the packages you need. We will surely need `breedR`. But you might want to use some other packages for plotting or data manipulation.

```
# library(···)
# ···
```

```
library(breedR)
library(tidyverse)
theme_set(theme_minimal())  # ggplot theme
```

1. Simple progeny test

Here is a template for the breedR function to fit a simple progeny test with response variable `y` and grouping variable `g` in dataset `D`. Use variables `phe_X` as response and `mum` as grouping variable from the `globulus` dataset (included with `breedR`)

```
# fm0 <- remlf90(
#   fixed = y ~ 1,
#   random = ~ g,
#   data = D)
```

```
fm0 <- remlf90(
  fixed = phe_X ~ 1,
  random = ~ mum,
  data = globulus)
```

*facundo.munoz@cirad.fr, @famuvie

2. Include further effects

1. Extend fm0 by including a fixed effect of the provenance (variable gg) and a random block effect (variable bl).
2. Specify sensible initial variances for the random component

```
# fm1 <- remlf90(
#   ···,
#   var.ini = list(···, resid = ···)
# )
```

```
## Initial variances:
## distribute the observed variance uniformly across all random
## components
var_response <- var(globulus$phe_X)
var_ini <- setNames(rep(var_response/3, 3), c("mum", "bl", "resid"))

fm1 <- remlf90(
  fixed = phe_X ~ gg,
  random = ~ mum + bl,
  var.ini = as.list(var_ini),
  data = globulus)
```

3. Extend fm1 by including an interaction between cross-classified variables

```
# fm2 <- remlf90(···)
```

```
## Verify which variables are nested and which are cross-classified
## (tip: check function table())
with(
  globulus,
  table(mum, gg)
)
```

```
##     gg
## mum   1  2  3  4  5  6  7  8  9 10 11 12 13 14
##   0  71 28  0  0  0  0  0  0  0 14  0  0  0  0
##   6   0  0  0  0  0  0  0  0  0  0 15  0  0  0
##   7   0  0  0  0  0  0  0  0  0  0 14  0  0  0
##   8   0  0  0  0  0  0  0  0  0  0 15  0  0  0
##   9   0  0  0  0  0  0  0  0  0  0 15  0  0  0
##   10  0  0  0  0 15  0  0  0  0  0  0  0  0  0
##   11  0  0  0  0 14  0  0  0  0  0  0  0  0  0
```

```
##    12  0  0  0  0 15  0  0  0  0  0  0  0  0  0
##    13  0  0  0  0 15  0  0  0  0  0  0  0  0  0
##    14  0  0  0  0 15  0  0  0  0  0  0  0  0  0
##    15  0  0  0  0 15  0  0  0  0  0  0  0  0  0
##    16  0  0  0  0  0  0  0  0 15  0  0  0  0  0
##    17  0  0  0  0  0  0  0  0 13  0  0  0  0  0
##    18  0  0  0  0  0  0  0  0 15  0  0  0  0  0
##    19  0  0  0  0  0  0  0  0 12  0  0  0  0  0
##    20  0  0  0  0  0  0  0  0 13  0  0  0  0  0
##    21  0  0  0  0  0  0  0  0 15  0  0  0  0  0
##    22  0  0  0  0  0  0  0 15  0  0  0  0  0  0
##    23  0  0  0  0  0  0  0 12  0  0  0  0  0  0
##    24  0  0  0  0  0  0  0 14  0  0  0  0  0  0
##    25  0  0  0  0  0  0  0 15  0  0  0  0  0  0
##    26  0  0  0  0  0  0  0 15  0  0  0  0  0  0
##    27  0  0  0  0  0  0  0 14  0  0  0  0  0  0
##    28  0  0  0  0  0  0  0 15  0  0  0  0  0  0
##    29  0  0  0  0  0  0  0 12  0  0  0  0  0  0
##    30  0  0  0  0  0  0  0 15  0  0  0  0  0  0
##    31  0  0 15  0  0  0  0  0  0  0  0  0  0  0
##    32  0  0 15  0  0  0  0  0  0  0  0  0  0  0
##    33  0  0 15  0  0  0  0  0  0  0  0  0  0  0
##    34  0  0 14  0  0  0  0  0  0  0  0  0  0  0
##    35  0  0 14  0  0  0  0  0  0  0  0  0  0  0
##    36  0  0 15  0  0  0  0  0  0  0  0  0  0  0
##    37  0  0 14  0  0  0  0  0  0  0  0  0  0  0
##    38  0  0 14  0  0  0  0  0  0  0  0  0  0  0
##    39  0  0  0  0  0  0  0  0  0  0  0  0 15  0
##    40  0  0  0  0  0  0 15  0  0  0  0  0  0  0
##    41  0  0  0 13  0  0  0  0  0  0  0  0  0  0
##    42  0  0  0 15  0  0  0  0  0  0  0  0  0  0
##    43  0  0  0 11  0  0  0  0  0  0  0  0  0  0
##    44  0  0  0 13  0  0  0  0  0  0  0  0  0  0
##    45  0  0  0  0  0 10  0  0  0  0  0  0  0  0
##    46  0  0  0  0  0  0  0  0  0  0 15 15  0  0
##    47  0  0  0  0  0  0  0  0  0  0  0 13  0  0
##    48  0  0  0  0  0  0  0  0  0  0  0  0  0 13
##    49  0  0  0  0  0  0  0  0  0  0  0  0  0 15
##    50  0  0  0  0  0  0  0  0  0  0  0  0  0 13
##    51  0  0  0  0  0  0  0  0  0  0  0  0  0 14
##    52  0  0  0  0  0  0  0  0  0  0  0  0  0 15
##    53  0  0  0  0  0  0  0  0  0  0  0  0  0 14
##    54  0  0  0  0  0  0  0  0  0  0  0  0  0 14
##    55  0  0  0  0  0  0  0  0  0  0  0  0  0 15
```

```
##   56  0  0  0  0  0  0  0  0  0  0  0  0  0 14
##   57  0  0  0  0  0  0  0  0  0  0  0  0  0 15
##   58  0  0  0  0  0  0  0  0  0  0  0  0  0 14
##   59  0  0  0  0  0  0  0  0  0  0  0  0  0 15
##   60  0  0  0  0  0  0  0  0  0  0  0  0  0 14
##   61  0  0  0  0  0  0  0  0  0  0  0  0  0 15
##   62  0  0  0  0  0  0  0  0  0  0  0  0  0 15
##   63  0  0  0  0  0  0  0  0  0  0  0  0  0 14
##   64  0  0  0  0  0  0  0  0  0  0  0  0  0 15
##   65  0  0  0  0  0  0  0  0  0  0  0  0  0 14
##   66  0  0  0  0  0  0  0  0  0  0  0  0  0 13
##   67  0  0  0  0  0  0  0  0  0  0  0  0  0 14
##   68  0  0  0  0  0  0  0  0  0  0  0  0  0 14
```

```
## These look nested, although the big table is a little bit difficult
## to read. Furthermore, note that mum 0 is actually a code for
## unknown.

with(
  globulus,
  table(gg, bl)
)
```

```
##     bl
## gg    1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
##   1   5  4  5  5  4  5  5  4  5  5  5  5  5  4  5
##   2   2  2  2  2  2  2  2  2  2  2  2  2  2  1  1
##   3   7  8  8  8  8  8  8  7  7  8  8  8  8  8  7
##   4   3  4  4  3  3  2  3  4  4  4  4  4  4  3  3
##   5   6  6  6  6  6  5  6  6  6  6  6  6  6  6  6
##   6   1  0  1  1  1  1  1  1  1  0  0  1  1  0  0
##   7   1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
##   8   9  8  9  8  9  9  7  8  9  9  9  9  8  8  8
##   9   6  6  6  6  6  6  6  6  6  6  5  5  5  4  4
##   10  1  1  1  1  1  1  1  1  1  1  1  1  0  1  1
##   11  5  5  5  5  5  5  5  5  4  5  5  5  5  5  5
##   12  2  2  2  2  2  2  2  2  2  1  2  2  2  1  2
##   13  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
##   14 20 20 20 20 20 19 21 20 21 19 21 20 20 18 20
```

```
## These are clearly crossed

## Another approach is to build the interaction and check the number
## of realised levels. If equal to the largest nubmer of levels of the
## variables, then they are nested (assuming there are no unobserved
```

```
## levels for any of the variables).
are_nested <- function(...) {
  x <- lapply(list(...), factor)
  max_nl <- max(vapply(x, nlevels, 1L))
  int_nl <- nlevels(do.call("interaction", c(x, list(drop = TRUE))))
  return(identical(int_nl, max_nl))
}
with(
  globulus,
  are_nested(
    replace(globulus$mum, which(globulus$mum == 0), NA),
    gg
  )
)
```

```
## [1] FALSE
```

```
## Oops, it turns out they are not strictly nested!! The following
## approach allows both to check nestedness and to figure out which
## levels are not nested.
globulus %>%
  select(mum, gg) %>%
  unique() %>%
  count(mum) %>%
  filter(n > 1)
```

```
## # A tibble: 2 x 2
##     mum     n
##   <int> <int>
## 1     0     3
## 2    46     2
```

```
## mum 0 is actually a code for unknown, thus it is ok. mum 46 should
## not come from two different provenances. This can be a mistake in
## the dataset.
```

```
## Note that I modify the dataset only for the sake of this model.
## In this way I keep my original dataset impoluted, and my workspace
## clean of unnecessary variables.
fm2 <- remlf90(
  fixed = phe_X ~ gg,
  random = ~ mum + bl + gg_bl,
  data = transform(
    globulus,
    gg_bl = factor(gg:bl)
```

```
  )
)
```

3. Extract results

1. Use functions summary(), fixef() and ranef() to extract estimates
2. Variance estimates are stored in fm$var
3. Plot observed phenotype and residual()s vs fitted() values assess the model predictive ability and to diagnose residuals

```
summary(fm2)
```

```
## Formula: phe_X ~ 0 + gg + mum + bl + gg_bl
##    Data: transform(globulus, gg_bl = factor(gg:bl))
##   AIC  BIC logLik
##  5679 5699  -2836
##
## Parameters of special components:
##
##
## Variance components:
##          Estimated variances   S.E.
## mum                   1.1095 0.4193
## bl                    2.5946 1.0688
## gg_bl                 0.1279 0.2783
## Residual             14.2820 0.6962
##
## Fixed effects:
##        value   s.e.
## gg.1  13.532 1.2216
## gg.2  14.028 1.3423
## gg.3  16.118 0.6659
## gg.4  11.864 0.8579
## gg.5  15.886 0.7259
## gg.6  10.211 1.6524
## gg.7  13.995 1.4977
## gg.8  15.693 0.6464
## gg.9  16.476 0.7346
## gg.10 12.845 1.5208
## gg.11 16.723 0.7638
## gg.12 16.920 1.0510
## gg.13 16.297 1.4977
```

```
## gg.14 14.425 0.5312
```

```
fixef(fm2)  # this is _printed_ as a table, but is actually a vector
```

```
## $gg
##       value      s.e.
## 1  13.53213 1.2216221
## 2  14.02847 1.3423309
## 3  16.11834 0.6659234
## 4  11.86400 0.8578513
## 5  15.88563 0.7259361
## 6  10.21074 1.6524497
## 7  13.99507 1.4977110
## 8  15.69261 0.6463732
## 9  16.47588 0.7345685
## 10 12.84542 1.5207746
## 11 16.72296 0.7637987
## 12 16.91993 1.0510469
## 13 16.29733 1.4977110
## 14 14.42464 0.5312102
```

```
attr(fixef(fm2)$gg, "se")  # The SE are stored as attributes
```

```
##         1         2         3         4         5         6         7
## 1.2216221 1.3423309 0.6659234 0.8578513 0.7259361 1.6524497 1.4977110
##         8         9        10        11        12        13        14
## 0.6463732 0.7345685 1.5207746 0.7637987 1.0510469 1.4977110 0.5312102
```

```
str(ranef(fm2))  # list of point estimates with SEs as attributes
```

```
## List of 3
##  $ mum  : atomic [1:64] 0 0.673 -0.55 -0.64 0.379 ...
##   ..- attr(*, "se")= Named num [1:64] 1.053 0.793 0.801 0.793 0.793 ...
##   .. ..- attr(*, "names")= chr [1:64] "15" "16" "17" "18" ...
##  $ bl   : atomic [1:15] -2.454 -2.503 -0.499 1.061 -1.957 ...
##   ..- attr(*, "se")= Named num [1:15] 0.605 0.607 0.601 0.605 0.605 ...
##   .. ..- attr(*, "names")= chr [1:15] "1" "2" "3" "4" ...
##  $ gg_bl: atomic [1:204] 0.00133 0.0603 -0.02113 -0.02519 -0.08931 ...
##   ..- attr(*, "se")= Named num [1:204] 0.351 0.352 0.351 0.351 0.352 ...
##   .. ..- attr(*, "names")= chr [1:204] "1:1" "1:2" "1:3" "1:4" ...
##  - attr(*, "class")= chr [1:2] "ranef.breedR" "breedR_estimates"
```

```
fm2$var  # We don't have a fancy function for this one (gets a matrix)
```

```
##          Estimated variances    S.E.
## mum                  1.10950 0.41928
## bl                   2.59460 1.06880
```

```
## gg_bl                 0.12792 0.27834
## Residual             14.28200 0.69624
```

```
globulus %>%
  mutate(
    predicted = fitted(fm2),
    residuals = residuals(fm2)
  ) %>%
  gather(
    outcome,
    value,
    phe_X, residuals
  ) %>%
  ggplot(aes(predicted, value)) +
  geom_abline(
    data = data.frame(
      int = c(0, 0),
      slp = c(1, 0),
      outcome = c("phe_X", "residuals")
    ),
    aes(intercept = int, slope = slp),
    color = "darkgray") +
  geom_point() +
  facet_grid(outcome ~ ., scales = "free_y", switch = "y") +
  labs(x = "Predicted Values", y = NULL)
```

```
## Warning: attributes are not identical across measure variables;
## they will be dropped
```

4. Pedigrees

1. Replace the genetic variable `mum` at *family* level, by an *animal model* with a genetic effect at *individual* level.

Help: `RShowDoc("Handling-pedigrees", package = "breedR")`

```
# fm3 <- remlf90(
#   fixed = ···,
#   random = ···,
#   genetic = list(
#     model = 'add_animal',  # model name for the term
#     pedigree = ···,        # pedigree (see Details in ?remlf90)
#     id = ···),             # variable name of the individual
#   data = globulus
# )
```

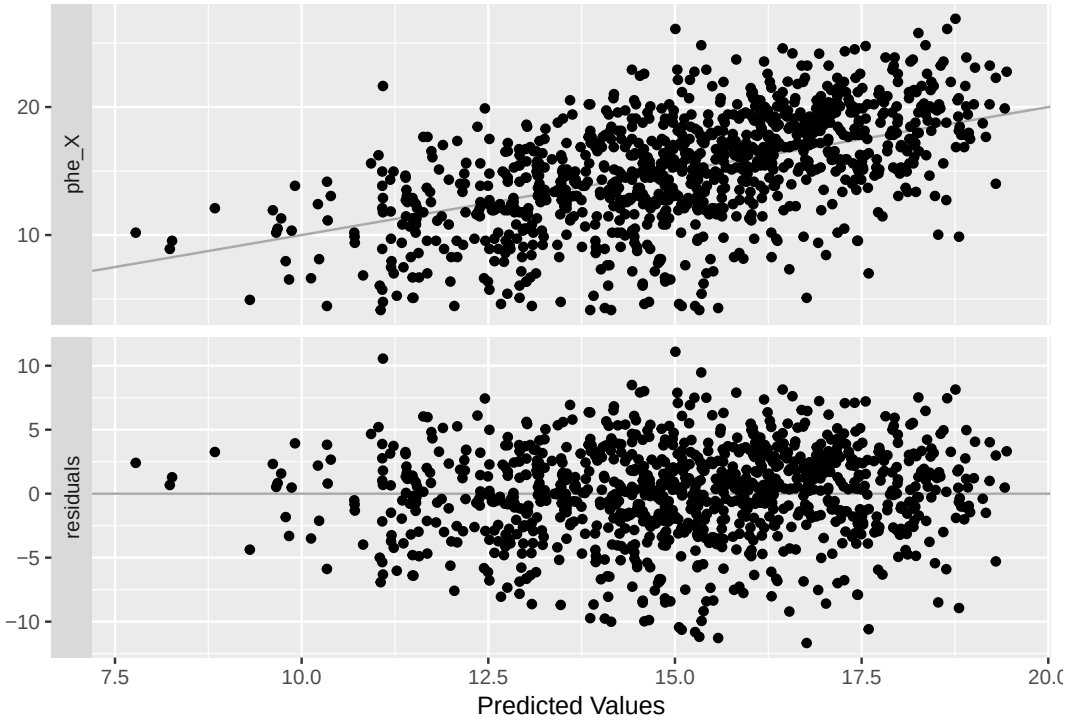


Figure 1: Observed and residual vs fitted values.

```
fm3 <- remlf90(
  fixed = phe_X ~ gg,
  random = ~ bl,
  genetic = list(
    model = 'add_animal',
    pedigree = globulus[, 1:3],
    id = 'self'),
  data = globulus
)
```

2. Retrieve the predicted individual breeding values. Make sure they are in the same order as observed in the globulus dataset.

```
# ranef(fm3)$genetic
## Here are the breeding values of observed and unobserved indivuduals
## Use model.matrix(fm3)$genetic to get the incidence matrix
## and combine appropriately
```

```
PBV.full <- ranef(fm3)$genetic
PBV <- model.matrix(fm3)$genetic %*% PBV.full
```

5. Heritability

1. Compute the heritability of the globulus dataset using the three available methods. See: `RShowDoc("Heritability", package = "breedR")`.

Method 1: using a `genetic` term automatically yields a heritability

```
## It's done!! check
# summary(fm3)

## Can you discover how to extract the numeric value (and SE) from the
## object?
```

```
summary(fm3)  # Heritability is automatically included in the summary
```

```
## Formula: phe_X ~ 0 + gg + bl + pedigree
##    Data: globulus
##   AIC  BIC logLik
##  5675 5690  -2835
##
## Parameters of special components:
##
##
## Variance components:
##          Estimated variances  S.E.
## bl                     2.656 1.083
## genetic                4.994 1.773
## Residual              10.495 1.560
##
##              Estimate    S.E.
## Heritability   0.2746 0.09577
##
## Fixed effects:
##        value   s.e.
## gg.1  13.533 0.6287
## gg.2  14.027 0.8548
## gg.3  16.117 0.6753
## gg.4  11.863 0.8748
## gg.5  15.885 0.7384
## gg.6  10.208 1.6900
## gg.7  13.995 1.5411
## gg.8  15.691 0.6546
## gg.9  16.474 0.7468
```

```
## gg.10 12.845 1.1333
## gg.11 16.717 0.9149
## gg.12 16.945 1.0974
## gg.13 16.297 1.5411
## gg.14 14.424 0.5331
```

```
fm3$funvars   # Its value is stored here
```

```
##             Heritability
## mean            0.275240
## sample mean     0.274600
## sample sd       0.095766
```

Method 2: using custom formulae

```
## Work out the formula for heritability from model terms
##
## For single-trait models,
## every term is G_i_i_1_1 where i is the index of the term
## except for the residual term which is R_1_1
##

## Translate the formula into a character string with the format above
# h2fml <- "..."

## Ask breedR to compute estimates for that quantity using
## `progsf90.options`
# fm3.1 <- remlf90(
#   fixed = ...,
#   random = ...,
#   genetic = ...,
#   progsf90.options = paste('se_covar_function h2', h2fml),
#   data = globulus
# )
```

```
## Counting model effects:
## 1: gg, 2: bl, 3: genetic
## h2 = genetic / (bl + genetic + redidual) = 3 / (2 + 3 + R)
h2fml <- 'G_3_3_1_1/(G_2_2_1_1+G_3_3_1_1+R_1_1)'

fm3.1 <- remlf90(
  fixed = phe_X ~ gg,
  random = ~ bl,
  genetic = list(
```

```
    model = 'add_animal',
    pedigree = globulus[, 1:3],
    id = 'self'),
  progsf90.options = paste('se_covar_function h2', h2fml),
  data = globulus
)

summary(fm3.1)  # Heritability is automatically included in the summary
```

```
## Formula: phe_X ~ 0 + gg + bl + pedigree
##    Data: globulus
##   AIC  BIC logLik
##  5675 5690  -2835
##
## Parameters of special components:
##
##
## Variance components:
##          Estimated variances  S.E.
## bl                     2.656 1.083
## genetic                4.994 1.773
## Residual              10.495 1.560
##
##              Estimate    S.E.
## h2             0.2746 0.09577
## Heritability   0.2746 0.09577
##
## Fixed effects:
##        value   s.e.
## gg.1  13.533 0.6287
## gg.2  14.027 0.8548
## gg.3  16.117 0.6753
## gg.4  11.863 0.8748
## gg.5  15.885 0.7384
## gg.6  10.208 1.6900
## gg.7  13.995 1.5411
## gg.8  15.691 0.6546
## gg.9  16.474 0.7468
## gg.10 12.845 1.1333
## gg.11 16.717 0.9149
## gg.12 16.945 1.0974
## gg.13 16.297 1.5411
## gg.14 14.424 0.5331
```

```
fm3.1$funvars   # Its value is stored here
```

```
##                    h2 Heritability
## mean          0.275240     0.275240
## sample mean 0.274600     0.274600
## sample sd    0.095766     0.095766
```

Method 3: using *Bootstrap* estimation

1. Fit the model to your data (DONE: fm3).
2. Write a function to simulate observations from the fitted model.

   You can use `breedR.sample.phenotype()` if the model is simple enough

3: Write a function to fit a simulated dataset and extract the target values

```
# resample_globulus <- function(fit) {
#   ## Use the estimated values in fit to produce a new data.frame
#   ## of the same size and with the same variables as globulus.
#   return(dat)
# }
```

```
# sim_target() <- function(dat) {
#   ## Fit the same model as fm3 to this fake dataset dat
#   ## Return the point estimates of all variances and heritability
#   return(estimates)
# }
```

```
## 2: Function to simulate from fitted model fm3
resample_globulus <- function(fit) {
  dat <- breedR.sample.phenotype(
    # fixed = 0,   # limitation: can't use categorical fixed covariate
    random = list(
      bl = list(
        nlevels = length(ranef(fit)$bl),
        sigma2 = fit$var['bl', 1]
      )
    ),
    genetic = list(
      ## Note that this will re-sample a full pedigree each time,
      ## which is a further source of variation not considered in
      ## previous estimates
      model    = 'add_animal',
      Nparents = c(63, 5),
      sigma2_a = fit$var['genetic', 1],
```

```
      check.factorial = FALSE
    ),
    residual.variance = fit$var['Residual', 1],
    N = nrow(model.frame(fit))
  )

  ## Remove founders from simulated data
  dat <- dat[!with(dat, is.na(sire) & is.na(dam)),]

  ## Fix the simulation to account for the fixed provenance effects
  dat$gg <- model.frame(fit)$gg  # include the variable in the dataset
  Xgg <- model.matrix(~ 0 + gg, dat)  # compute the incidence matrix
  gg_eff <- as.vector(Xgg %*% fixef(fit)$gg)  # fixed effects values
  dat$phenotype <- dat$phenotype + gg_eff

  # rename variables to keep the original names
  names(dat) <- gsub("rnd\\.", "", names(dat))
  return(dat)
}
```

```
## 2: Function to fit a simulated dataset and extract the target
## values
sim_target <- function(dat) {
  ## Fit the original model
  ## avoiding messages about initial variances spec
  res <- suppressMessages(
    remlf90(fixed = phenotype ~ gg,
            random = ~ bl,
            genetic = list(
              model = 'add_animal',
              pedigree = dat[, 1:3],
              id = 'self'),
            data = dat)
  )
  ## Extract point estimates of all variances
  ## and point estimate of heritability
  ans <- res$var[, 'Estimated variances']
  ans <- c(ans, h2 = unname(ans['genetic']/sum(ans)))
  return(ans)
}
```

```
## Replicate the data-generation+estimation process many times
## Warning: this can take a while, depending on the model and the
## dataset.
```

```
boot_estimates <- function(N, fit) {
  ans <- replicate(N, sim_target(dat = resample_globulus(fit)))
  return(as.data.frame(t(ans)))
}

empirical_dist <- boot_estimates(N = 100, fit = fm3)
```

```
empirical_dist %>%
  summarise_all(funs(estimate = mean, SE = sd)) %>%
  gather(var, value) %>%
  separate(var, into = c("quantity", "estimate")) %>%
  mutate(
    quantity = fct_inorder(quantity)
  ) %>%
  spread(estimate, value)
```

```
##   quantity   estimate         SE
## 1       bl  2.7245307 1.14745278
## 2  genetic  5.1510700 1.39296857
## 3 Residual 10.4231260 0.91502110
## 4       h2  0.2800827 0.06503522
```

```
## Compare SEs with
rbind(
  fm3$var,
  h2 = fm3$funvars[-1, ]
)
```

```
##          Estimated variances     S.E.
## bl                    2.6557 1.083000
## genetic               4.9940 1.773300
## Residual             10.4950 1.560300
## h2                    0.2746 0.095766
```

```
empirical_dist %>%
  select(-h2) %>%
  gather("variable", "value") %>%
  ggplot(aes(value)) +
  geom_histogram(bins = 21) +
  facet_wrap("variable")
```

```
empirical_dist %>%
  select(h2) %>%
  ggplot(aes(h2)) +
  # geom_histogram(bins = 21) +
```

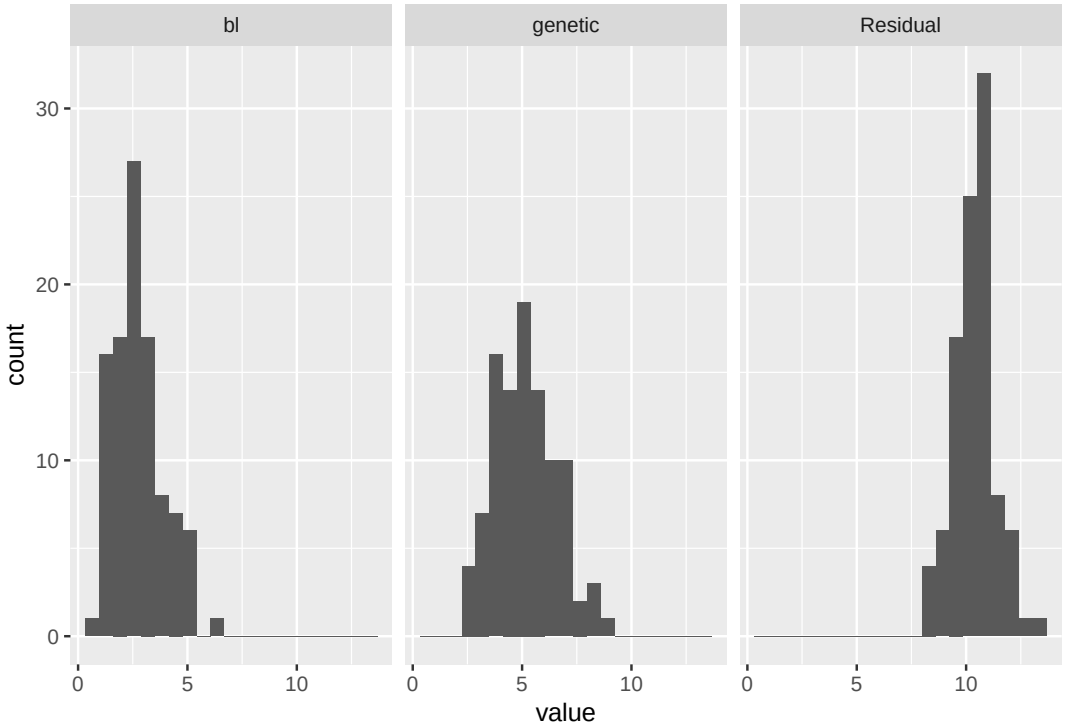


Figure 2: Resampling distribution of variances

```
  geom_density() +
  lims(x = c(0, 1))
```

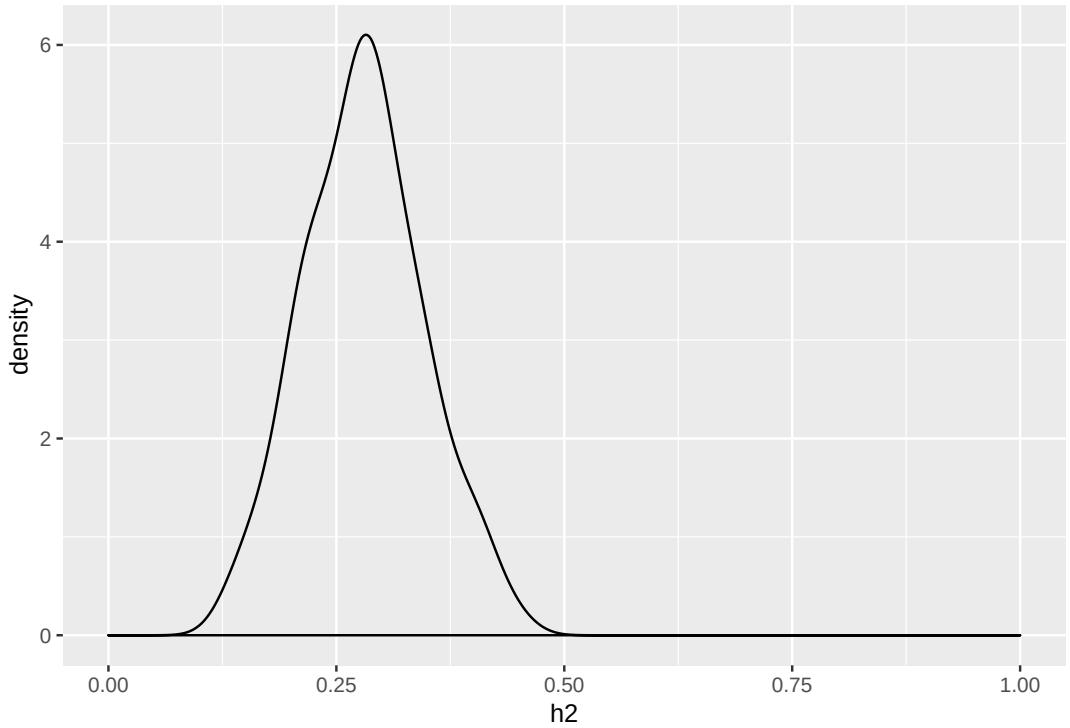


Figure 3: Resampling distribution of heritability

6. Spatial effects

1. Fit an animal model without the `blocks` effect

```
# fm4 <- remlf90(···)
```

```
# I will use the genetic component a few times (DRY)
gen.globulus <- list(
  model    = 'add_animal',
  pedigree = globulus[, 1:3],
  id       = 'self'
)

fm4 <- remlf90(
  fixed = phe_X ~ gg,
  genetic = gen.globulus,
  data = globulus
```

```
)
```

2. Asess the spatial independence of residuals by
    - plotting their spatial distribution
    - interpreting the variogram of residuals

```
## You can simply use the plot() function, specifying type =
## "residuals" See ?plot.remlf90
##
## If you get an error, please read the super-informative error
## message and fix it!
```

```
## For the variogram of residuals, I'll let you guess the function
## name.
```

```
## Since coordinates have not
## been passed before they
## must be provided explicitly.
coordinates(fm4) <- globulus[, c('x', 'y')]
plot(fm4, 'resid')
```

```
## Of course, you can do it manually, but it takes a bit more work.
# globulus %>%
#   add_column(residuals = resid(fm4)) %>%
#   ggplot(aes(x, y, fill = residuals)) +
#   geom_tile() +
#   coord_fixed() +
#   scale_fill_gradient2()
```

```
variogram(fm4)
```

3. Extend `fm4` with each of the three possible spatial models in `breedR`

```
# fm5 <- list(
#   bl = remlf90(
#     ...
#     spatial = list(
#       model = 'blocks',  # spatial model name
#       coord = ···,       # matrix or data.frame with coordinates
#       id = 'bl'),        # name of the variable
#     data    = globulus
#   ),
#   sp = remlf90(
#     ...
#     spatial = list(
```

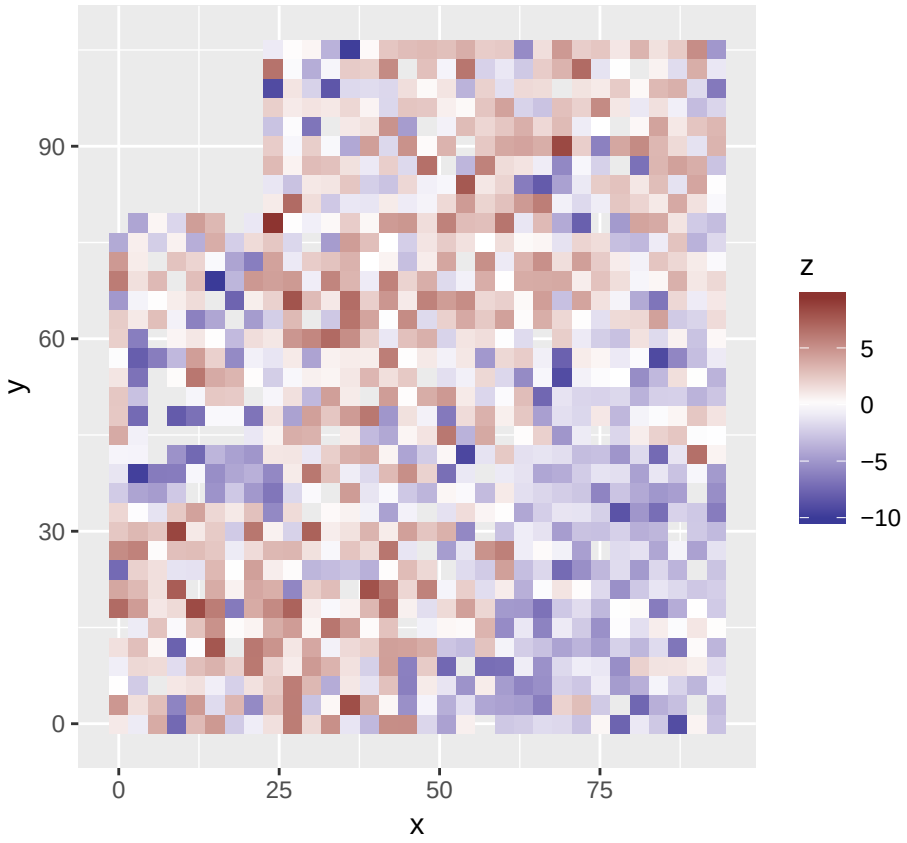


Figure 4: Spatial distribution of residuals from an animal model without spatial effect.

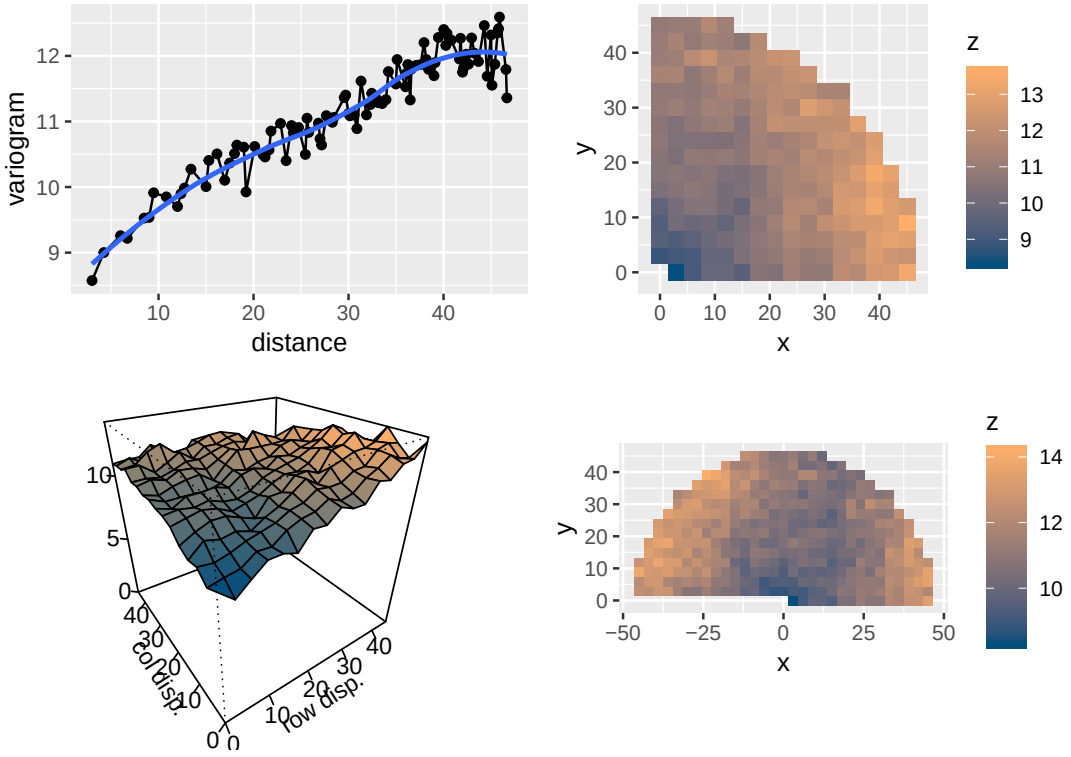


Figure 5: Variogram of residuals from an animal model without spatial effect.

```
#       model   = 'splines',  # spatial model name
#       coord   = ...         # matrix or data.frame with coordinates
#       n.knots = ...         # number of internal knots in each dim
#     ),
#     data    = globulus,
#     method  = 'em'          # `ai` does not like splines
#   ),
#   ar = remlf90(
#     fixed   = phe_X ~ gg,
#     genetic = gen.globulus,
#     spatial = list(
#       model = 'AR',       # spatial model name
#       coord = ...,        # matrix or data.frame with coordinates
#       rho   = ...         # autocorrelation coefficient in each dim
#     ),
#     data    = globulus
#   )
# )
```

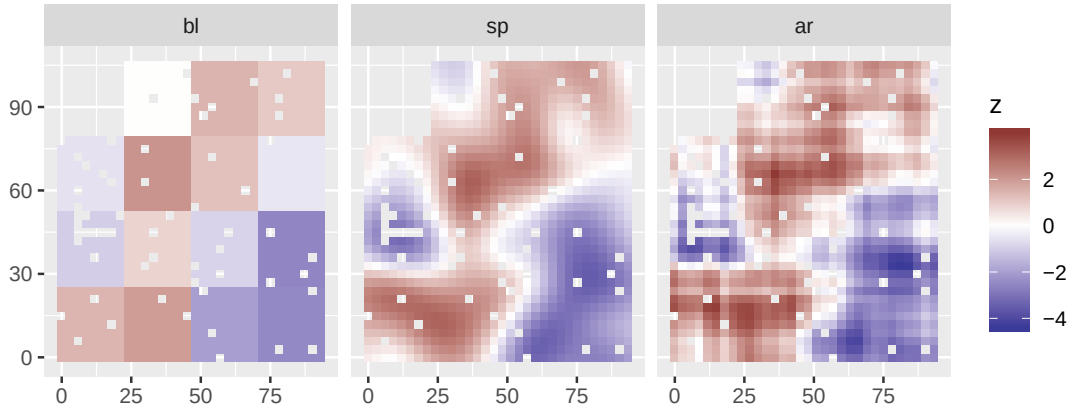
```
fm5 <- list(
  bl = remlf90(
    fixed   = phe_X ~ gg,
    genetic = gen.globulus,
    spatial = list(
      model = 'blocks',
      coord = globulus[, c('x', 'y')],
      id = 'bl'
    ),
    data    = globulus
  ),
  sp = remlf90(
    fixed   = phe_X ~ gg,
    genetic = gen.globulus,
    spatial = list(
      model   = 'splines',
      coord   = globulus[, c('x','y')],
      n.knots = c(9, 9)
    ),
    data    = globulus,
    method  = 'em'  # `ai` does not like splines
  ),
  ar = remlf90(
    fixed   = phe_X ~ gg,
    genetic = gen.globulus,
```

```
    spatial = list(
      model = 'AR',
      coord = globulus[, c('x','y')],
      rho = c(.9, .9)
    ),
    data    = globulus
  )
)
```

4. Plot and compare the predicted spatial effect from each model

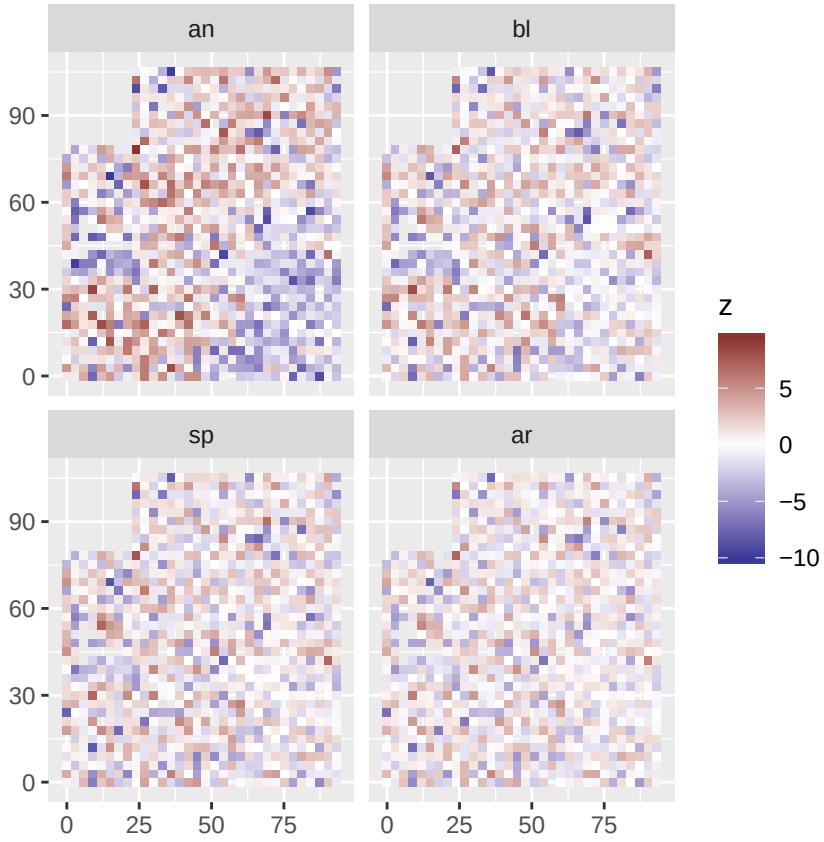
```
## You can simply use the plot() function with type = "spatial"
## (also "fullspatial", check the difference).
##
## In order to compare the three plots under the same scale, use
## compare.plots(list(p1, p2, p3)). See `?compare.plots`.
##
```

```
compare.plots(lapply(fm5, plot, type = "spatial")) +
  labs(x = NULL, y = NULL)  # remove useless coordinate labels
```



5. Plot and compare the residuals from the animal model without spatial effect (`fm4`) and from the three spatial models (`fm5`).

```
compare.plots(
  c(an = list(plot(fm4, type = "residuals")),
    lapply(fm5, plot, type = "residuals")
  )
) +
  facet_wrap(".id", ncol = 2) +  # Override default arrangement
  labs(x = NULL, y = NULL)
```



7. Genotype-Environment interaction in multi-site trials

Here we are using the `douglas` dataset, included with `breedR`, which is a 3-site trial with shared genetic material.

For comparison, let's first fit a *reference* model without interaction.

1. Fit a **reference** model for the variable `C13` with fixed effects of provenance (`orig`) and site (`site`) and a random family (`mum`) effect

```
# fm7.0 <- remlf90(
#   ...
# )
```

```
fm7.0 <- remlf90(
  fixed  = C13 ~ site + orig,
  random = ~ mum,
  data = douglas
)
```

2. Fit a simple interaction model with constant variance accross sites.

```
# fm7.1 <- remlf90(
#   ...
# )
```

```
fm7.1 <-
  douglas %>%
  mutate(
    famXsite = factor(factor(mum):site)
  ) %>%
  remlf90(
    fixed  = C13 ~ site + orig,
    random = ~ mum + famXsite,
    data = .
  )
```

3. Extend the previous model to account for site-varying interaction variances.

```
## 1. Create dummy variables for the families on each site,
## that are 0 or NA in the other sites.
##
## 2. Fit the model using a main family effect and three site-specific
## interaction effects
```

```
## 1. Create dummy interaction family variables for each site
## You can do it site by site like this:
# douglas %>%
#   mutate(
#     fam = factor(mum),
#     fam_s1 = replace(fam, site != "s1", NA),
#     fam_s2 = replace(fam, site != "s2", NA),
#     fam_s3 = replace(fam, site != "s3", NA)
#   )

## But the above has a lot of duplicate code, and is not practical
## with many sites. Instead, you can use an indicator matrix for site
## and build from there.
fm7.2 <-
  douglas %>%
  bind_cols(
    data.frame(.$mum * model.matrix(~ 0 + site, data = .)) %>%
      setNames(paste0("fam_s", 1:3))
  ) %>%
  remlf90(
    fixed  = C13 ~ site + orig,
    random = ~ mum + fam_s1 + fam_s3,
```

```
    # method = "em",
    data = .
  )

## In the above code, the model with all three interaction variables
## did not converge with method = "ai", and it returned a very small
## variance for fam_s2 with method = "em".
##
## This is symptomatic of no or very little variance. Thus I removed
## the effect and refitted the model with AI.
```

4. Examine and compare the site-specific breeding values from each of the three models above

```
## Each of the terms of the differents models will possibly have a
## different number of levels. Be very careful in keeping the right
## order to make values comparable.
##
## A safe practice is to pre-multiply the random effects by the
## incidence matrix (`model.matrix()`) to retrieve the values in the
## ordering of the original dataset.
##
## Do that for each of the models, and then filter out duplicate lines
## using `unique()`.
```

```
## Extract the predicted values for any effect in a given model
fitted_effect <- function(fm, eff) {
  as.numeric(model.matrix(fm)[[eff]] %*% ranef(fm)[[eff]])
}

douglas %>%
  select(mum, site) %>%
  mutate(
    fam = factor(mum),
    M0 = fitted_effect(fm7.0, "mum"),
    M1 = fitted_effect(fm7.1, "mum") +
      fitted_effect(fm7.1, "famXsite"),
    M2 = fitted_effect(fm7.2, "mum") +
      fitted_effect(fm7.2, "fam_s1") +  # initial zeros are still 0
      fitted_effect(fm7.2, "fam_s3")
  ) %>%
  unique() %>%   # remove redundant values
  gather(model, BV, M0:M2) %>%
  ggplot(aes(as.numeric(site), BV, group = fam)) +
```

```
  geom_line(color = 'darkgray') +
  scale_x_continuous(breaks = 1:3,
                     labels = paste0('s_', 1:3)) +
  theme(axis.ticks.x = element_blank()) +
  facet_wrap("model") +
  xlab("Site") +
  ylab("Interaction")
```