Additive Genetic Models in Mixed Populations

Facundo Muñoz

2017-04-14 breedR version: 0.12.1

Contents

Full diallel trial with founders from two different base populations E and J

```r
## Setup
library(breedR)
library(ggplot2)
set.seed(123)

## Simulation parameters
n.founders <- c(E = 9, J = 9)
sigma2 <- c(E = 3, J = 2, resid = 1)

founders <- data.frame(id = c(paste0('E', 1:n.founders['E']),
                              paste0('J', 1:n.founders['J'])),
                       pop.idx = c(rep(1, n.founders['E']),
                                   rep(2, n.founders['J'])),
                       BV = c(rnorm(n.founders['E'], sd = sqrt(sigma2['E'])),
                              rnorm(n.founders['J'], sd = sqrt(sigma2['J']))))


n.obs <- sum(n.founders)*150
obs.parents.idx <- matrix(sample(nrow(founders), 2*n.obs, replace = TRUE), ncol = 2)

## The 'family' is independent of the order of the parents
## While the population only takes into account the origin
## e.g. cross2fam(c('J3', 'E1')) gives 'E1:J3'
## while cross2pop(c('J3', 'E1')) gives 'EJ'
cross2fam <- function(x) paste(founders$id[sort(x)], collapse = ':')
cross2pop <- function(x) paste(names(n.founders)[founders$pop.idx[sort(x)]], collapse = '')

## Mendelian sampling term
msp <- function(x) {
  ss <- sigma2[founders$pop.idx[x]]
  s2 <- (ss[1] + ss[2])/4
  rnorm(1, sd = sqrt(s2))
}

dat <- data.frame(
  id  = sum(n.founders) + seq.int(n.obs),
  dad = obs.parents.idx[, 1],
  mum = obs.parents.idx[, 2],
  fam = apply(obs.parents.idx, 1, cross2fam),
  sp  = apply(obs.parents.idx, 1, cross2pop),
  bv  = apply(obs.parents.idx, 1,
              function(x) mean(founders$BV[x]) + msp(x)),
  resid = rnorm(n.obs, sd = sqrt(sigma2['resid'])))
```

```r
dat <- transform(dat,
                 y = bv + resid)

## Printing simulated setting
print(table(dat[, c('mum', 'dad')]), zero.print = "")
```

```
##     dad
## mum   1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
##   1    9  7  8 11 14  5 10 13  7  5  6 11  6 11 12  7 11 11
##   2    5  7 10  6  7  6  6  6  6 10  8 10  8  9  6  9  6  9
##   3   10  5 16 11 11  9  7 13 12  5  9 10  9  3 10  8  5 13
##   4    9  8  8  3  9  6  4  4  7  9 12 10  2  9 10 16  5  7
##   5    6  6  6  5  8  7 10 10 12  8  9  9  9  9  5 11  6  7
##   6    9 12  9  9 10  3 11 11 11 11  8 10  7 10  9  7 14  6
##   7   11  6  7 10  6 12  6  8  6  7 11  7  6  5 10  9  8  8
##   8    8 13 13  5  9 14 11 10  8  6 10  9  8  5 11  9  4  9
##   9    6  9  7  7  7  8 10  9 12  6  6  7  8 14  7  5  8  4
##   10 10  2  8  5  8 10 12  9 14  7 10  5 10  6 11  5  4  4
##   11 10  6  5  5 15  9  9 12  9 12  5  9  8 12  8  6  9  9
##   12 10  7  4  9  4  2  8 10  7  4  9 15 10  7 10 10  9  5
##   13 11 10  8  5 12  9  6 11 15 10 14 10  3  9  7  6  6 12
##   14  8 14  7 12 12 13  6  5 11  8  2  5  8 10  5 12  5  5
##   15 12  7 14  5  5 11  3  8 10  8  9 13  9  8  9  9  7 13
##   16 10 15 11  8  7  4  7 10  9  6  8  7  9 12  8  6  5 14
##   17  8  5 12 14  5 13  1  8  7  8  6  7  7  9 10  5  8 10
##   18  5  6 15  9 10  2  8  1  6  8  6 10  5  5 12 13  7  6
```

```r
str(dat)
```

```
## 'data.frame':    2700 obs. of  8 variables:
##  $ id   : num  19 20 21 22 23 24 25 26 27 28 ...
##  $ dad  : int  14 4 6 5 3 8 8 7 3 3 ...
##  $ mum  : int  13 14 13 16 6 4 2 1 17 15 ...
##  $ fam  : Factor w/ 171 levels "E1:E1","E1:E2",..: 152 62 88 78 39 56 25 7 50 48 ...
##  $ sp   : Factor w/ 3 levels "EE","EJ","JJ": 3 2 2 2 1 1 1 1 2 2 ...
##  $ bv   : num  1.837 -0.585 3.279 2.039 1.376 ...
##  $ resid: num  2.234 -0.672 -1.495 -0.133 1.17 ...
##  $ y    : num  4.07 -1.26 1.78 1.91 2.55 ...
```

There will be two independent additive-genetic variance models for the J and E populations. Each variance component will be estimated using the *pure* offspring only.

```r
## Build a pedigree for the whole mixed population
## and get the kinship matrix A
ped <- build_pedigree(1:3, data = dat)
A <- pedigreemm::getA(ped)

## Build the full incidence matrix
Z <- as(dat$id, 'indMatrix')

## Give the index vector of additive-genetic random effects
## that belong to one subpopulation;
## 'E', 'J' (founders) or 'EE', 'EJ' or 'JJ' (offspring).
idx_pop <- function(x) {
  if (nchar(x) == 1) grep(x, founders$id)
  else
    match(dat$id[dat$sp == x], as.data.frame(ped)$self)
}
```

0.1 Method 1: Hybrids as an independent population

This is the easiest way to go. It works as a first approximation, but has several shortcommings.

It needs to estimate one *virtual* variance for the hybrid population which is not linked to any genetic variance in the *real world*. Moreover, we don't use the hybrid observations to learn about the two varainces that really matter: $\sigma2_E$ and $\sigma2_J$

The advantage is that it predicts the Breeding Values of the hybrid offspring. However, the accuracy may be limited by the violation of the *single population* hypothesis of the model.

```
## Avoid estimating BLUPS for which we don't have information
## Otherwise, the run takes much longer (5 hs vs 6 min in this example)

## A[idx_pop('EE'), idx_pop('JJ')]  # This is null: populations are independent
Z_EE <- Z[, idx_pop('EE')]
Z_EJ <- Z[, idx_pop('EJ')]
Z_JJ <- Z[, idx_pop('JJ')]

A_EE <- A[idx_pop('EE'), idx_pop('EE')]
A_EJ <- A[idx_pop('EJ'), idx_pop('EJ')]
A_JJ <- A[idx_pop('JJ'), idx_pop('JJ')]

## Now fit a model with three additive-genetic compnents,
## by means of generic effects (as only one 'genetic' is allowed in breedR)

res1 <- remlf90(y ~ sp,
                generic = list(
                  E = list(incidence  = Z_EE,
                           covariance = A_EE),
                  J = list(incidence  = Z_JJ,
                           covariance = A_JJ),
                  H = list(incidence  = Z_EJ,
                           covariance = A_EJ)),
                data = dat
)
```

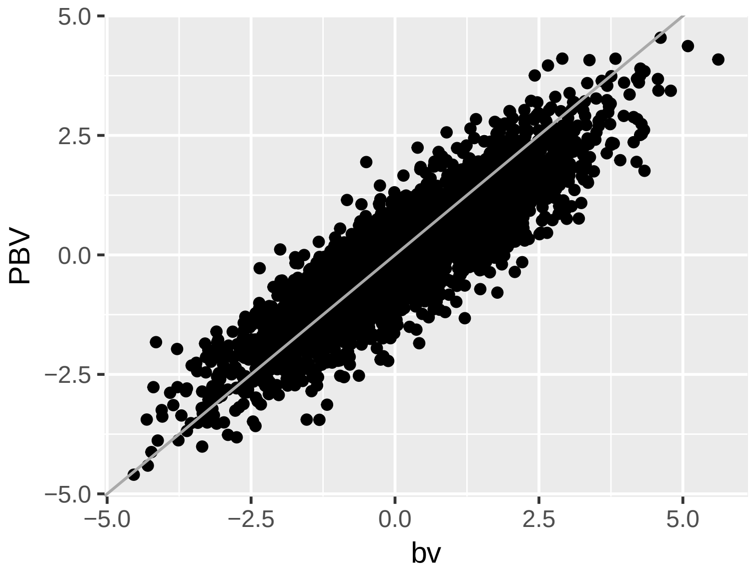
```
## Using default initial variances given by default_initial_variance()
## See ?breedR.getOption.
```

```
summary(res1)
```

```
## Formula: y ~ 0 + sp
##    Data: dat
##    AIC   BIC logLik
##  10067 10091  -5029
##
## Parameters of special components:
##
##
## Variance components:
```

```
##           Estimated variances   S.E.
## E                       3.414 0.7079
## J                       2.213 0.6757
## H                       2.466 0.6606
## Residual                1.004 0.3316
##
## Fixed effects:
##          value   s.e.
## sp.EE 0.21488 0.6192
## sp.EJ 0.24748 0.3724
## sp.JJ 0.15989 0.4992
```

```
PBV <- as.matrix(cbind(Z_EE, Z_JJ, Z_EJ)) %*%
  do.call('rbind', lapply(ranef(res1), function(x) cbind(PBV = x, se = attr(x, 'se'))))

ggplot(cbind(dat, PBV), aes(bv, PBV)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, col = 'darkgray')
```



0.2 Method 2: GCA/SCA model for hybrids

Another approach is to model hybrids with GCAs and SCAs, with additional variance parameters. The additive component of the genetic variances for the E and J populations is half the population variance. But these parameters will also gather dominance and epistasis effects.

However, we don't take advantage of the known relationship between the additive components by treating them as independent parameters. Furthermore, we don't use the relationship between hybrids and pures to learn about the original genetic variances. Nor even within hybrids, as we treat SCA as an unstructured effect, while there are both half and full siblings.

```
## We only want to apply 'dad', 'mum' and 'sca' effects to hybrids,
## and make it zero for non-hybrids. We do so by pre-multiplying by a
## diagonal indicator matrix
Ind <- diag(dat$sp == 'EJ')

## Build a specific incidence matrices for generic random effects
```

```r
Z_dad <- Ind %*% as(dat$dad, 'indMatrix')
```

```
## Note: method with signature 'matrix#sparseMatrix' chosen for function '%*%',
##  target signature 'matrix#lgTMatrix'.
##  "ANY#TsparseMatrix" would also be valid
```

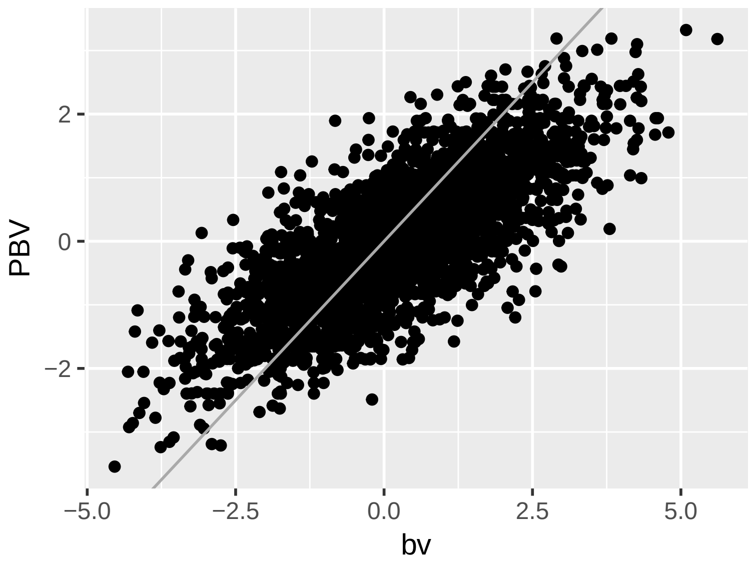
```r
Z_mum <- Ind %*% as(dat$mum, 'indMatrix')
Z_sca <- Ind %*% as(as.numeric(dat$fam), 'indMatrix')

## The structure variances are diagonal
D <- diag(sum(n.founders))

res2 <- remlf90(y ~ sp,
                generic = list(
                  E = list(incidence  = Z_EE,
                           covariance = A_EE),
                  J = list(incidence  = Z_JJ,
                           covariance = A_JJ),
                  dad = list(incidence = Z_dad,
                             covariance = D),
                  mum = list(incidence = Z_mum,
                             covariance = D),
                  sca = list(incidence = Z_sca,
                             covariance = diag(nlevels(dat$fam)))),
                data = transform(dat)
)
```

```
## Using default initial variances given by default_initial_variance()
## See ?breedR.getOption.
```

```r
summary(res2)
```

```
## Formula: y ~ 0 + sp
##    Data: transform(dat)
##    AIC   BIC logLik
##  10137 10172  -5062
##
## Parameters of special components:
##
##
## Variance components:
##          Estimated variances    S.E.
## E                    1.52350 0.29447
## J                    0.73816 0.19334
## dad                  0.66033 0.24008
## mum                  0.55015 0.20263
## sca                  0.01581 0.02514
## Residual             2.07940 0.07179
##
## Fixed effects:
##         value   s.e.
## sp.EE 0.21521 0.4165
## sp.EJ 0.25605 0.2627
## sp.JJ 0.15932 0.2929
```

```r
PBV <- as.matrix(cbind(Z_EE, Z_JJ, Z_dad, Z_mum, Z_sca)) %*%
  do.call('rbind', lapply(ranef(res2), function(x) cbind(PBV = x, se = attr(x, 'se'))))

ggplot(cbind(dat, PBV), aes(bv, PBV)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, col = 'darkgray')
```



This approach does not work very well. Much of the variation has not been adequately accounted for, and ended up in the residuals. May be there is some misspecification in the model.

0.3 Method 3: grid search

The difficulty of the problem resides that we cannot estimate two variance parameters at the same time. If we split the matrix A in blocks corresponding to each subpopulation, we can write the covariance matrix for the mixture as

$$\begin{aligned}\Sigma &= \begin{bmatrix} \sigma_E^2 A_E & \mathbf{0} & \sigma_E^2 A_{E:EE} & \frac{3\sigma_E^2+\sigma_J^2}{4} A_{E:H} & \mathbf{0} \\ \mathbf{0} & \sigma_J^2 A_J & \mathbf{0} & \frac{\sigma_E^2+3\sigma_J^2}{4} A_{J:H} & \sigma_J^2 A_{J:JJ} \\ \sigma_E^2 A_{E:EE} & \mathbf{0} & \sigma_E^2 A_{EE} & \frac{3\sigma_E^2+\sigma_J^2}{4} A_{EH} & \mathbf{0} \\ \frac{3\sigma_E^2+\sigma_J^2}{4} A_{E:H} & \frac{\sigma_E^2+3\sigma_J^2}{4} A_{J:H} & \frac{3\sigma_E^2+\sigma_J^2}{4} A_{HE} & \frac{\sigma_E^2+\sigma_J^2}{2} A_{HH} & \frac{\sigma_E^2+3\sigma_J^2}{4} A_{HJ} \\ \mathbf{0} & \sigma_J^2 A_{J:JJ} & \mathbf{0} & \frac{\sigma_E^2+3\sigma_J^2}{4} A_{JH} & \sigma_J^2 A_{JJ} \end{bmatrix} \\ &= \sigma_E^2 \begin{bmatrix} A_E & \mathbf{0} & A_{E:EE} & \frac{3+\lambda}{4} A_{E:H} & \mathbf{0} \\ \mathbf{0} & \lambda A_J & \mathbf{0} & \frac{1+3\lambda}{4} A_{J:H} & \lambda A_{J:JJ} \\ A_{E:EE} & \mathbf{0} & A_{EE} & \frac{3+\lambda}{4} A_{EH} & \mathbf{0} \\ \frac{3+\lambda}{4} A_{E:H} & \frac{1+3\lambda}{4} A_{J:H} & \frac{3+\lambda}{4} A_{HE} & \frac{1+\lambda}{2} A_{HH} & \frac{1+3\lambda}{4} A_{HJ} \\ \mathbf{0} & \lambda A_{J:JJ} & \mathbf{0} & \frac{1+3\lambda}{4} A_{JH} & \lambda A_{JJ} \end{bmatrix},\end{aligned} \tag{1}$$

where $\lambda = \frac{\sigma_J^2}{\sigma_E^2}$.

However, if we are not interested in evaluating the parents, we can disregard the first two block rows and columns from the matrix.

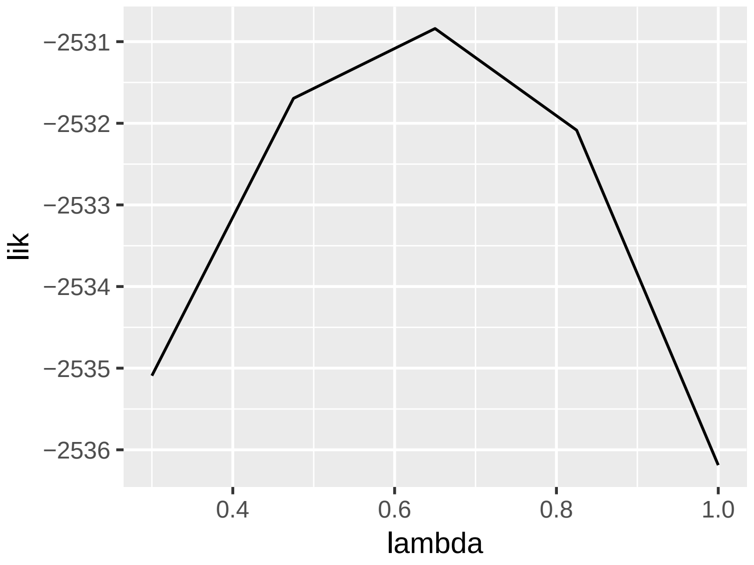
Now, fit the model for several values of λ and maximize the likelihood.

```r
## Setup parallel computing
# library(doParallel)
# cl <- makeCluster(2)
# registerDoParallel()
# on.exit(stopCluster(cl))

## Introduce the corresponding scaling factors
## in the relationship matrix
scale_A <- function(x) {

    ## The pure E subpopulations remains the same
  S <- A
  E.idx <- c(idx_pop('E'), idx_pop('EE'))

  ## The pure J subpopulations get multiplied by lambda
  J.idx <- c(idx_pop('J'), idx_pop('JJ'))
  S[J.idx, J.idx] <- A[J.idx, J.idx] * x

  ## The hybrids related wuth pure E get a factor of (3+lambda)/4
  S[idx_pop('EJ'), E.idx] <- A[idx_pop('EJ'), E.idx] * (3+x)/4
  S[E.idx, idx_pop('EJ')] <- A[E.idx, idx_pop('EJ')] * (3+x)/4

  ## The hybrids related wuth pure J get a factor of (1+3*lambda)/4
  S[idx_pop('EJ'), J.idx] <- A[idx_pop('EJ'), J.idx] * (1+3*x)/4
  S[J.idx, idx_pop('EJ')] <- A[J.idx, idx_pop('EJ')] * (1+3*x)/4

  ## Finally, the hybrids related with other hybrids get a factor of (1+lambda)/2
  S[idx_pop('EJ'), idx_pop('EJ')] <- A[idx_pop('EJ'), idx_pop('EJ')] * (1+x)/2

  return(S)
}

## Condicional likelihood given lambda
cond_lik <- function(x) {
  require(breedR)
  ## Conditional structure matrix
  S <- scale_A(x)

  ## Temporarily, let's use only the pure pops
  idx <- c(idx_pop('EE'), idx_pop('JJ'))

  suppressWarnings(
    res <- remlf90(y ~ sp,
                   generic = list(
                     E = list(incidence  = Z[dat$sp != 'EJ', idx],
                              covariance = S[idx, idx])),
                   data = dat[dat$sp != 'EJ', ]
    )
  )
  logLik(res)
}

lambda <- seq(.3, 1, length.out = 5)
```

```
lik <- sapply(lambda, cond_lik)  # (sequential)
```

```
## Using default initial variances given by default_initial_variance()
## See ?breedR.getOption.
##
## Using default initial variances given by default_initial_variance()
## See ?breedR.getOption.
##
## Using default initial variances given by default_initial_variance()
## See ?breedR.getOption.
##
## Using default initial variances given by default_initial_variance()
## See ?breedR.getOption.
##
## Using default initial variances given by default_initial_variance()
## See ?breedR.getOption.
```

```
# lik <- foreach(x = seq.int(lambda), .combine = c) %dopar% cond_lik(lambda[x])

ggplot(data.frame(lambda, lik), aes(lambda, lik)) +
  geom_line()
```



This one works well. But for some reason, when I include the hybrids, for lambdas below about 0.58 and all the variance is accounted for residual variance.

There must be something wrong somewhere. May be the simulation of the Breeding Values is not correct?

Analogously, if I include only the hybrid subpopulation, it also works quite well, identifying the base variance.

```
## Take lambda maximizing the likelihood
lambda0 <- lambda[which.max(lik)]

S <- scale_A(lambda0)

## Temporarily, let's use only the pure pops
idx <- c(idx_pop('EE'), idx_pop('JJ'))

# ## Remove the founders, which I don't want to evaluate
```

```r
# idx <- -(1:sum(n.founders))

res3 <- remlf90(y ~ sp,
                generic = list(
                  E = list(incidence  = Z[dat$sp != 'EJ', idx],
                           covariance = S[idx, idx])),
                data = dat[dat$sp != 'EJ', ])
```

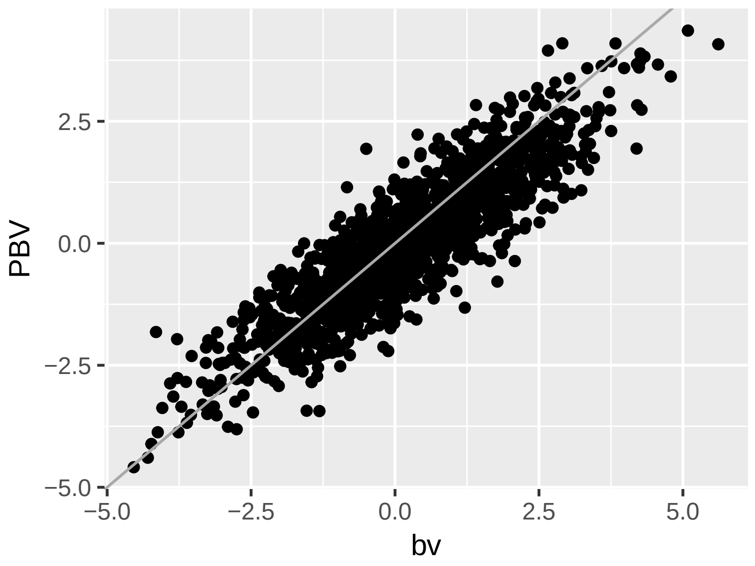
```
## Using default initial variances given by default_initial_variance()
## See ?breedR.getOption.
```

```r
summary(res3)
```

```
## Formula: y ~ 0 + sp
##    Data: dat[dat$sp != "EJ", ]
##   AIC  BIC logLik
##  5066 5076  -2531
##
## Parameters of special components:
##
##
## Variance components:
##          Estimated variances   S.E.
## E                      3.391 0.8126
## Residual               1.013 0.3327
##
## Fixed effects:
##         value   s.e.
## sp.EE 0.21489 0.6171
## sp.EJ 0.00000 0.0000
## sp.JJ 0.15989 0.4981
```

Lambda was maximized at 0.65, giving an estimated additive-genetic variance for the J population of 2.2.

```r
PBV <- as.matrix(Z[dat$sp != 'EJ', idx]) %*%
  do.call('rbind', lapply(ranef(res3), function(x) cbind(PBV = x, se = attr(x, 'se'))))

ggplot(cbind(dat[dat$sp != 'EJ', ], PBV), aes(bv, PBV)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, col = 'darkgray')
```


PBV
bv
2.5
0.0
−2.5
−5.0
−5.0
−2.5
0.0
2.5
5.0