DenHaag

John Woolliams

5 February 2017

DenHaag is a package of functions for simulating populations using various options for managing inbreeding. The package provides a function for establising base population `new_pop()`, and the simplest option is then to add generations with random pairing of males and females using the function `add_gen()`. Both functions output a data frame containing data on pedigree (`"id"`, `"sire"`, `"dam"` and inbreeding coefficient `"f"`), a phenotype (`"ptype"`) with true breeding value (`"tbv"`), an estimate of breeding value (`"ebv"`). The populations are assumed to have two sexes so the data frame also contains data on sex (`"sex"`), and also a data column called `"noff"` which is set by the user to determine the number of offspring each parent has in the next generation. The EBV is calculated within `add_gen()` using a call to `blup()`, which in turn makes a call to function `a_inv()`.

The base is set up by calling `new_pop(nn,hh)` where `nn` is the number of individuals and `hh` is the heritability of the trait. The functions in the package assume an equal number of males and females in the population, and so `new_pop()` checks that the number in the base is an even positive integer and flags an error if not. It also flags an error if the heritability is outside the open interval (0,1).

```
library("DenHaag", lib.loc="~/R/win-library/3.3")
# new_pop() accepts numbers ...
my.df <- new_pop(7,0.25)

## [1] "Census size is expected to be even not odd! 7!"

my.df <- new_pop(8,1.25)

## [1] "Heritability out of bounds! 1.25!"

# ... or assigned variables
my.nn <- 8; my.hh <- 0.25
my.df <- new_pop(my.nn,my.hh)

## [1] "Creating 8 individuals for base generation"
##  id sex noff      ptype        ebv sire dam         tbv f
##   1   1    0 -0.4863416 -0.1215854    0   0  0.01921775 0
##   2   1    0 -0.9252726 -0.2313182    0   0 -0.19817195 0
##   3   1    0  0.6776721  0.1694180    0   0  0.36713140 0
##   4   1    0 -1.2143704 -0.3035926    0   0 -0.04250867 0
##   5   2    0 -1.4128958 -0.3532240    0   0 -0.58103219 0
##   6   2    0 -0.5000935 -0.1250234    0   0  0.63749181 0
##   7   2    0  1.6909138  0.4227285    0   0  1.20935392 0
##   8   2    0  1.0995517  0.2748879    0   0  0.66768668 0
```

Note "noff" is set to 0. The user selects which individuals become parents and the number of offspring for each parent by assigning "noff" values.

```
my.df$noff[2]=4; my.df$noff[4]=4; my.df$noff[5]=4; my.df$noff[7]=4
```

The next generation is then formed by calling add_gen(my.df,my.hh). The output from add_gen() is an extended data frame. The function add_gen() detects errors in setting "noff" such as total offspring not being a even positive integer, or numbers of offspring from males not being equal to the number of offsring from females.

```
my.df$noff[2]=4; my.df$noff[4]=3; my.df$noff[5]=4; my.df$noff[7]=3
my.df <- add_gen(my.df,my.hh)

## [1] "Offspring numbers are expected to be even not odd! 7!"

my.df$noff[2]=4; my.df$noff[4]=4; my.df$noff[5]=4; my.df$noff[7]=3
my.df <- add_gen(my.df,my.hh)

## [1] "Numbers of male and female parents unequal in matings!"

# the heritability can be entered as a number
my.df$noff[2]=4; my.df$noff[4]=4; my.df$noff[5]=4; my.df$noff[7]=4
my.df <- add_gen(my.df,0.25)

## [1] "Creating 8 offspring"
##  id sex noff       ptype         ebv sire dam         tbv f
##   1   1    0 -0.48634163 -0.12158541    0   0  0.01921775 0
##   2   1    0 -0.92527264 -0.33297671    0   0 -0.19817195 0
##   3   1    0  0.67767214  0.16941804    0   0  0.36713140 0
##   4   1    0 -1.21437038 -0.25167901    0   0 -0.04250867 0
##   5   2    0 -1.41289580 -0.45653442    0   0 -0.58103219 0
##   6   2    0 -0.50009350 -0.12502337    0   0  0.63749181 0
##   7   2    0  1.69091383  0.47629396    0   0  1.20935392 0
##   8   2    0  1.09955172  0.27488793    0   0  0.66768668 0
##   9   1    0 -1.01101384 -0.48279246    2   5 -0.42862704 0
##  10   1    0  0.59925116  0.14702899    2   7  0.29638354 0
##  11   1    0 -1.64944512 -0.53915506    4   5 -0.60485798 0
##  12   1    0 -0.57745275 -0.42085517    2   5 -0.40684953 0
##  13   2    0  0.89549226  0.22419102    4   7  0.48563948 0
##  14   2    0  0.77595609 -0.19266917    4   5  0.16043896 0
##  15   2    0 -0.60579163 -0.02511998    2   7  0.36494599 0
##  16   2    0 -0.02107487  0.09325286    4   7  0.22946008 0
```

Note that on output my.df has "noff" set to 0. Successive generations are then produced by repeated assigment of "noff" and calls to add_gen().

Recommended contributions can be obtained for achieving a target group coancestry by setting a group coancestry, say my.gc, followed by a call to oc_sel(my.df,my.tp,my.tc,my.gc). The parameter my.tp is a number which gives the number of eligible candidates for selection, which are assumed to be the **most recent** individuals. The parameter my.tc is the total number of offspring required in the next

generation, which is only used to scale the optimum contributions to a projected number of offspring - although these projections are not integer! The function `oc_sel()` calls `a_mat()` to produce the numerator relationships among the candidates. The function prints a table listing selected parents and contributions but the function only returns `"TRUE"` or `"FALSE"` indicating the success of the algorithm. The user can then set `"noff"` guided by the recommendations.

```
my.gc <- 0.2
oc_sel(my.df,8,8,my.gc)

## [1] "Recommendations for 8 offspring from the most recent 8 parents"
##  id sex         ebv           c      noff
##   9   1 -0.48279246 0.037827315 0.6052370
##  10   1  0.14702899 0.367190032 5.8750405
##  12   1 -0.42085517 0.094982653 1.5197225
##  13   2  0.22419102 0.248201034 3.9712165
##  14   2 -0.19266917 0.115357592 1.8457215
##  15   2 -0.02511998 0.009069228 0.1451076
##  16   2  0.09325286 0.127372147 2.0379544

## [1] TRUE

# NOTE due to authors inexperience of 'SWEAVE", the following may not agree
# with the recommendations!
my.df$noff[9]=1; my.df$noff[11]=4; my.df$noff[12]=3
my.df$noff[14]=3; my.df$noff[15]=2; my.df$noff[16]=3
my.df <- add_gen(my.df,0.25)

## [1] "Creating 8 offspring"
##  id sex noff       ptype         ebv sire dam         tbv     f
##   1   1    0 -0.48634163 -0.12158541    0   0  0.01921775 0.000
##   2   1    0 -0.92527264 -0.28187479    0   0 -0.19817195 0.000
##   3   1    0  0.67767214  0.16941804    0   0  0.36713140 0.000
#
# … lines removed for brevity! …
#
##   9   1    0 -1.01101384 -0.45511122    2   5 -0.42862704 0.000
##  10   1    0  0.59925116  0.18652601    2   7  0.29638354 0.000
##  11   1    0 -1.64944512 -0.45704097    4   5 -0.60485798 0.000
##  12   1    0 -0.57745275 -0.46508414    2   5 -0.40684953 0.000
##  13   2    0  0.89549226  0.23538795    4   7  0.48563948 0.000
##  14   2    0  0.77595609 -0.23835689    4   5  0.16043896 0.000
##  15   2    0 -0.60579163  0.15844077    2   7  0.36494599 0.000
##  16   2    0 -0.02107487  0.03202776    4   7  0.22946008 0.000
##  17   1    0 -0.21810803 -0.32835891    9  14 -0.01518335 0.125
##  18   1    0 -0.56378628 -0.38201562   12  14 -0.29780081 0.125
##  19   1    0 -0.87175803 -0.42256451   11  14 -0.20802897 0.250
##  20   1    0 -0.56155780 -0.26237106   11  16 -0.37546726 0.125
##  21   2    0  1.90116479  0.14362345   11  15  0.03598308 0.000
##  22   2    0 -1.02596614 -0.33216218   12  16 -0.13913900 0.000
```

```
##  23   2    0  0.15327593 -0.10607495   11  15  0.01231825 0.000
##  24   2    0 -0.24093209 -0.22001446   12  16 -0.49866434 0.000
```

An alternative to setting a target group coancestry is to set a cost of inbreeding and use penalised contributions. Analagous to `oc_sel()`, recommended contributions for penalised contributions can be obtained by using the function `cost_f(my.df,my.tp,my.tc,my.cost)` where the first three arguments are as with `oc_sel()` and the final argument is set by the user.

```
my.cost <- 1000
cost_f(my.df,8,8,my.cost)

## [1] "Recommendations for 8 offspring from the most recent 8 parents"
## [1] "Recommendations have group coancestry 0.21342007500113"
##  id sex        ebv          c      noff
##  17   1 -0.3283589 0.17565547 2.8104875
##  18   1 -0.3820156 0.12742125 2.0387399
##  19   1 -0.4225645 0.05625261 0.9000418
##  20   1 -0.2623711 0.14067068 2.2507308
##  21   2  0.1436235 0.12964660 2.0743457
##  22   2 -0.3321622 0.12042217 1.9267547
##  23   2 -0.1060750 0.12939691 2.0703505
##  24   2 -0.2200145 0.12053432 1.9285491

## [1] TRUE

# NOTE due to authors inexperience of 'SWEAVE", the following may not agree
# with the recommendations!
my.df$noff[17]=2; my.df$noff[18]=2; my.df$noff[19]=2; my.df$noff[20]=2
my.df$noff[21]=4; my.df$noff[22]=1; my.df$noff[23]=1; my.df$noff[24]=2
my.df <- add_gen(my.df,0.25)

## [1] "Creating 8 offspring"
##  id sex noff       ptype         ebv sire dam         tbv      f
##   1   1    0 -0.48634163 -0.12158541    0   0  0.01921775 0.0000
##   2   1    0 -0.92527264 -0.31992473    0   0 -0.19817195 0.0000
##   3   1    0  0.67767214  0.16941804    0   0  0.36713140 0.0000
#
# … lines removed for brevity! …
#
##  17   1    0 -0.21810803 -0.32692168    9  14 -0.01518335 0.1250
##  18   1    0 -0.56378628 -0.61048020   12  14 -0.29780081 0.1250
##  19   1    0 -0.87175803 -0.55042847   11  14 -0.20802897 0.2500
##  20   1    0 -0.56155780 -0.37321789   11  16 -0.37546726 0.1250
##  21   2    0  1.90116479 -0.06562991   11  15  0.03598308 0.0000
##  22   2    0 -1.02596614 -0.41164911   12  16 -0.13913900 0.0000
##  23   2    0  0.15327593 -0.17215490   11  15  0.01231825 0.0000
##  24   2    0 -0.24093209 -0.32803097   12  16 -0.49866434 0.0000
##  25   1    0 -0.45184319 -0.36430264   20  24 -1.25278621 0.1875
##  26   1    0 -0.83714238 -0.41637010   20  24 -0.12226313 0.1875
##  27   1    0  0.44573223 -0.21209592   19  21 -0.23847198 0.1875
```

```
##  28   1    0 -0.63697876 -0.52808007   18  22 -0.28186179 0.1875
##  29   2    0 -2.48381847 -0.62802309   18  21 -0.05379796 0.1250
##  30   2    0  0.43524871 -0.11093465   17  21  0.07676607 0.1250
##  31   2    0 -0.04354662 -0.22170158   17  23 -0.30238204 0.1250
##  32   2    0 -1.65379789 -0.47930884   19  21  0.04915578 0.1875
```

The function `add_ma_gen()` is a modification of add-gen to produce the next generation using maximum avoidance (minimum coancestry) mating. The function call to `add_ma_gen()` is identical to `add_gen()`. Function `add_ma_gen()` calls `a_mat()` to obtain the numerator relationships for the parents. Prior to printing the updated population data frame it prints out the group coancestry, the average inbreeding coefficient achieved and the estimate of alpha - which is expected to be negative!

```
my.df$noff[25]=2; my.df$noff[26]=2; my.df$noff[27]=2; my.df$noff[28]=2
my.df$noff[29]=2; my.df$noff[30]=2; my.df$noff[31]=2; my.df$noff[32]=2
my.df <- add_ma_gen(my.df,0.25)

## [1] "Creating 8 offspring with maximum avoidance"
## [1] "Group coancestry 0.25146484375 and average offspring F 0.1640625 with
alpha -0.116764514024788"
##  id sex noff       ptype          ebv sire dam         tbv        f
##   1   1    0 -0.48634163 -0.121585408    0   0  0.01921775 0.000000
##   2   1    0 -0.92527264 -0.297319702    0   0 -0.19817195 0.000000
##   3   1    0  0.67767214  0.169418035    0   0  0.36713140 0.000000
#
# … lines removed for brevity! …
#
##  25   1    0 -0.45184319 -0.408269160   20  24 -1.25278621 0.187500
##  26   1    0 -0.83714238 -0.279268862   20  24 -0.12226313 0.187500
##  27   1    0  0.44573223 -0.086154192   19  21 -0.23847198 0.187500
##  28   1    0 -0.63697876 -0.392182349   18  22 -0.28186179 0.187500
##  29   2    0 -2.48381847 -0.628918105   18  21 -0.05379796 0.125000
##  30   2    0  0.43524871  0.051318296   17  21  0.07676607 0.125000
##  31   2    0 -0.04354662 -0.112246981   17  23 -0.30238204 0.125000
##  32   2    0 -1.65379789 -0.312776652   19  21  0.04915578 0.187500
##  33   1    0 -0.68555644 -0.539178088   25  29 -0.07449120 0.156250
##  34   1    0  0.30680194 -0.049145481   27  31 -0.36646881 0.203125
##  35   1    0  0.27761236 -0.277330930   28  32  0.29748722 0.156250
##  36   1    0 -1.32911972 -0.561973430   26  29 -0.05688834 0.156250
##  37   2    0 -1.37200265 -0.325622623   25  30 -1.25544462 0.140625
##  38   2    0  2.45721099  0.203020285   26  30  0.21463430 0.140625
##  39   2    0  0.45111666 -0.256637757   28  32  0.03136374 0.156250
##  40   2    0  0.20751364 -0.061386504   27  31  0.01325146 0.203125
```

At any time the population data frame `my.df` can be saved and reloaded.

```
save(my.df,file="my_df.Rda")
# ... and later ...
load("my_df.Rda")
```