Simulating Breeding Programs

Matthias Frisch

1 Selection gain in one generation of phenotypic selection and monitoring genetic diversity in selection programs

We consider 264 tropical maize lines from CIMMYT's Drought Tolerance Maize for Africa project that were analyzed with 1135 SNP markers. Data from Crossa et al. (2010). The original marker data are available from the Genetics webpage. Thanks to Dr. Jose Crossa and Dr. Raman Babu for providing the map data and the permission to use this data set as an example for data analysis.

We consider this data as population of inbred lines in which we carry out selection for yield. Let's assume we want to generate a high yielding, drought tolerant synthetic variety.

In this first example, the inbred lines are tested in a field trial with $h^2 = 0.8$ and the best 30 lines are selected. The expected selection gain is estimated.

1.1 Data

We load SelectionTools and data files that come with the package.

```
> library ("SelectionTools")
> vignette.data <- new.env(parent = emptyenv())
> data("v-tropmaize-vcf", package="SelectionTools", envir=vignette.data)
> data("v-tropmaize-phe", package="SelectionTools", envir=vignette.data)
```

The marker data and the linkage map are provided in VCF like data structure (see Section 3 for details on data format).

```
> st.STvcf.to.dataframe ( vignette.data$v.tropmaize.vcf ) [ 1:10, 1:12 ]
```

	CHROM	POS	ID	REF	ALT	FORMAT	1	2	3	4	5	6
1	1	0.157104	PZB008591	1	2	GT	1/1	1/1	1/1	1/1	1/1	./.
2	1	2.941215	PZA036132	1	2	GT	1/1	0/0	0/0	1/1	./.	0/0
3	1	2.941320	PZA036131	1	2	GT	0/0	0/0	0/0	0/0	0/0	0/0
4	1	2.942227	PZA036142	1	2	GT	0/0	0/0	0/0	1/1	1/1	./.
5	1	2.942391	PZA036141	1	2	GT	1/1	1/1	0/0	1/1	1/1	1/1
6	1	4.193258	PZA003931	1	2	GT	0/0	0/0	0/0	0/0	0/0	0/0
7	1	4.193538	PZA003934	1	2	GT	0/0	1/1	1/1	1/1	1/1	1/1
8	1	4.608173	PZA028692	1	2	GT	0/0	0/0	0/0	0/0	0/0	0/0
9	1	9.058572	PZB019151	1	2	GT	0/0	1/1	1/1	0/0	./.	1/1
10	1	10.092630	PZA035181	1	2	GT	1/1	1/1	1/1	1/1	1/1	1/1

and the phenoypic data look like

```
> vignette.data$v.tropmaize.phe [1:10,]
```

	ii	yy
1	1	2.516
2	2	1.641
3	3	0.416
4	4	1.339
5	5	1.730
6	6	2.704
7	7	1.972
8	8	1.844
9	9	2.082
10	10	2.898

We load the data into SelectionTools

```
> st.load.vcf.data(vignette.data$v.tropmaize.vcf)
> st.load.performance.data(vignette.data$v.tropmaize.phe)
```

and look at the loaded data

```
> x <- st.marker.data.statistics ()      # Overview of marker data
```

```
> x$genotypes[1:10,1:5]
```

```
     Mar/Ind   1   2   3   4
1  PZB008591 2/2 2/2 2/2 2/2
2  PZA036132 2/2 1/1 1/1 2/2
3  PZA036131 1/1 1/1 1/1 1/1
4  PZA036142 1/1 1/1 1/1 2/2
5  PZA036141 2/2 2/2 1/1 2/2
6  PZA003931 1/1 1/1 1/1 1/1
7  PZA003934 1/1 2/2 2/2 2/2
8  PZA028692 1/1 1/1 1/1 1/1
9  PZB019151 1/1 2/2 2/2 1/1
10 PZA035181 2/2 2/2 2/2 2/2
```

```
> x$individual.list[1:5,]
```

```
  Name    InMis
1    1 0.056691
2    2 0.062268
3    3 0.047398
4    4 0.056691
5    5 0.146840
```

```
> x$marker.list[1:5,]
```

```
       Name NoAll MaMis ExHet  AM  A1  A2
1 PZB008591     2 0.364 0.253 192  50 286
2 PZA036132     2 0.076 0.358  40 374 114
3 PZA036131     2 0.019 0.129  10 482  36
4 PZA036142     2 0.049 0.469  26 314 188
5 PZA036141     2 0.087 0.491  46 274 208
```

The returned list consists of three elements. The element `$genotypes` lists the complete marker data. In `$indivdiual.list` all genotypes are listed, for each the frequency of missing marker data is given. `$marker.list` contains the number of alleles observed at the marker (`NoAll`), the frequency of missing values for each marker (`MaMis`), the expected heterozygosity (`ExHet`), and the count of the observed alleles. Here `AM` is the count of missing alleles, `A1` the count of allele 1, and so on. The expected heterozygosity is a measure for the allelic diversity at a locus.

1.2 Preprocessing the marker data

Then the marker data are preprocessed. We discard markers that have (a) more than two alleles, (b) more that 10 percent missing values, (c) an expected heterozygosity below 0.1. And we discard individuals that have more than 10% missing values:

```
> st.restrict.marker.data (NoAll.MAX=2  ) # Max. no. of alleles
> st.restrict.marker.data (MaMis.MAX=0.1) # Max. missing at a marker
> st.restrict.marker.data (ExHet.MIN=0.1) # Minimum gene diversity
> st.restrict.marker.data (InMis.MAX=0.1) # Max. missing per individual
```

If we now check the marker data again we see that the very first marker `PZB008591` is gone now as he had 36.4% missing data.

```
> x <- st.marker.data.statistics ()
```

```
> x$marker.list[1:5,]
```

```
       Name NoAll MaMis ExHet AM  A1  A2
1 PZA036132     2 0.057 0.360 26 332 102
2 PZA036131     2 0.013 0.131  6 422  32
3 PZA036142     2 0.009 0.461  4 292 164
4 PZA036141     2 0.061 0.477 28 262 170
5 PZA003931     2 0.022 0.191 10 402  48
```

1.3 Estimation of marker effects

We start by making a copy of the data set that we call `C0` for cycle 0, the base population of our selection experiment.

```
> st.copy.marker.data ( "C0", source.data.set="default" )
```

We estimate the marker effects with ridge regression. We can either use the standard ridge regression BLUP (Whittaker et al. 2000, Meuwissen et al. 2001) with estimating variance components, which is equivalent to an ‘anmimal model’ based G-BLUP (Habier et al. 2007)

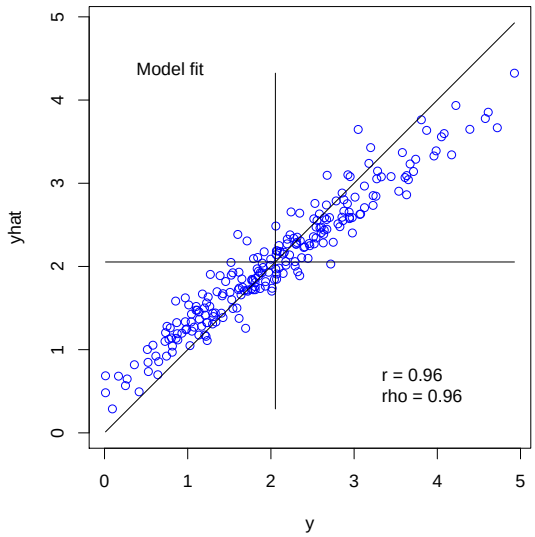
```
> gs.esteff.rr ( method="BLUP", data.set="C0")
```

or the fast ridge regression method by Hofheinz and Frisch (2014) that omits the variance component estimation,

```
> gs.esteff.rr ( method="BLUP", hsq=0.85, maxiter=0, data.set="C0")
```

We check the model fit.

```
> gs.plot.model.fit ( training.set="C0" )
```



In the model fit plot, the yhat values are the estimated genotypic values of the base population. This genotypic values are used for the simulations.

After having carried out the effect estimation, the marker data set hold genetic effects. Mathematically ridge regression in an 'animal model' is carried out and the effects were obtained from the genotypic values and the marker matrix with the transfomation given in Eq. (8) of Shen et al. (2013). The core computations are carried out with SelectionTool's mixed model engine `st.mxd()`.

1.4 Initialization of the simulation routines

The simulation routines of SelectionTools are based on the software described in Maurer et al. (2008). The simulation routines use their own data structures, which are different from the data structures used for marker matrices and genome wide prediction. Therefore the data set of the C0 cycle is used to initialize simulation routines. We first reset the simulation routines and then chose a intermediate level of messages

```
> reset.all()
> st.set.info.level(0)
```

Then we create a simulation population from the marker matrix of cycle 0

```
> st.set.simpop ( pop.name="C0", data.set="C0" )
```

```
M: 10 chromosomes defined
M: 823 loci defined
```

We check the defined populations in the simulation routines

```
> list.populations()                  # List all defined populations
```

```
  PopName count
1      C0   230
```

We check the list of effects available for evaluating simulation populations

```
> list.effects()                      # List all defined effects
```

```
  effect weight
1 effect      1
```

In the simulation routines, marker data is not stored in marker data matrices, but instead the individual chromosomes are reconstructed, and these reconstructed chromosomes are used in a count-location simulation of meiosis, see Maurer et al. (2008) for the details. This is why this initialization step is required.

This conversion loads the chromosome parameters, linkage map, marker data and the estimated effects from a marker data set to the simulaion routines. As the coding in the for marker data sets is usually 0/1/2 and in the simulation part of the software -1/0/1 a baseline correction following Strandén and Christensen (2011), Equation (2) is applied.

1.5 Distribution of genotypic values in the base population

To get the genotypic value of an individual, several steps are required, first the marker data matrix is generated, then the list of the marker effects is used together with the marker data matrix to calculate the genotypic values of the individuals. Then the calculated genotypic values are returned.

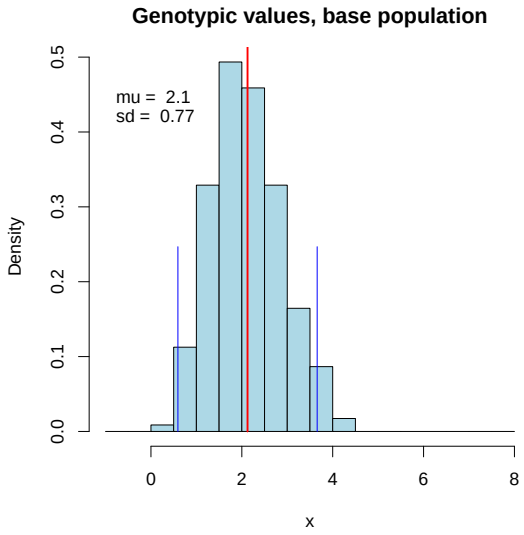
```
> genotype.population("C0")         # Construct marker data matrix
> evaluate.population("C0")         # Calculate genotypic values
> g <- get.population.gvalue("C0")$gvalue          # Save in a variable
```

We define a pretty-printing function for the histogram

```
> plt.hist <- function (x,main="") {
+   oldpar <- par(no.readonly = TRUE)
+   on.exit(par(oldpar), add = TRUE)
+   par(mar=c(4,4,3,1))
+   mu <- mean(x)
+   sd <- sdev(x)
+   h <- hist (x,xlim=c(-1,8),probability=TRUE,
+              breaks=seq(-1,8,by=0.5),
+              col="lightblue",
+              main=main)
+   ym <- max(h$density)
+   lines(c(mu, mu), c(0, 1),col="red",lwd=2)
+   lines(c(mu-2*sd, mu-2*sd), c(0, ym/2),col="blue",lwd=1)
+   lines(c(mu+2*sd, mu+2*sd), c(0, ym/2),col="blue",lwd=1)
+   text (-1,0.90*ym,paste("mu = ",format(mu,digits=2)),pos=4)
+   text (-1,0.85*ym,paste("sd = ",format(sd,digits=2)),pos=4)
+ }
```

which is then used for a plot of the genotypic values of the base population.

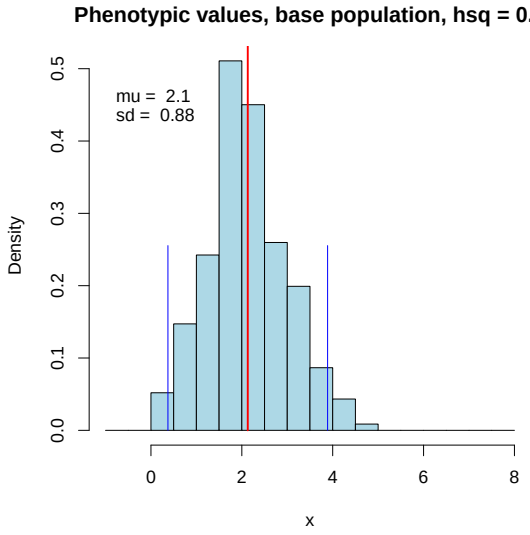
```
> plt.hist (g, main="Genotypic value, base population")
```



1.6 Distribution of the phenotypic values after a field trial

The inbred lines are phenotypically evaluated in a field trial with heritability $h^2 = 0.8$. Internally during the phenotyping step the software does the following; $h^2 = \sigma_g^2/(\sigma_g^2 + \sigma_m^2)$ is solved for the masking variance: $\sigma_m^2 = \sigma_g^2/h^2 - \sigma_g^2$. The genetic σ_g^2 is determined from the genotypic values, then a random realization of the masking variance is added to each genotypic to obtain a phenotypic value.

```
> phenotype.population("C0", hsq=0.8)
> p <- get.population.pvalue("C0")$pvalue
> plt.hist (p,main="Phenotypic values, base population, hsq = 0.8")
```



1.7 Selection for phenotypic values

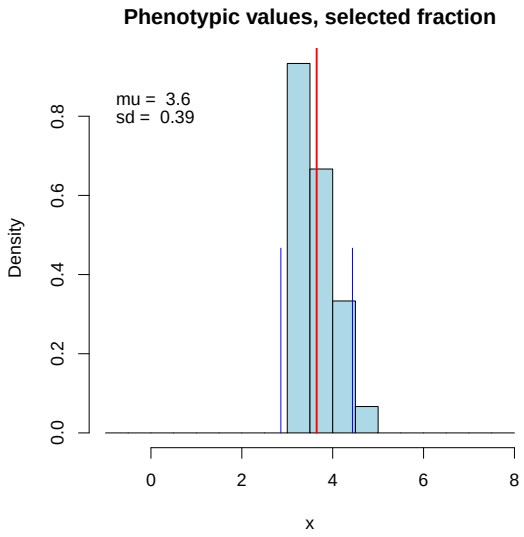
Now we sort the population according to the genotypic and put the 30 lines with the greatest score into an new population called `C0sel`

```
> population.sort("C0", decreasing=TRUE, selection.criterion="P")
> population.divide("C0sel", "C0", 30)
> list.populations()
```

```
  PopName count
1   C0sel    30
2      C0   200
```

and plot the phenotypic values of the best 30 individuals.

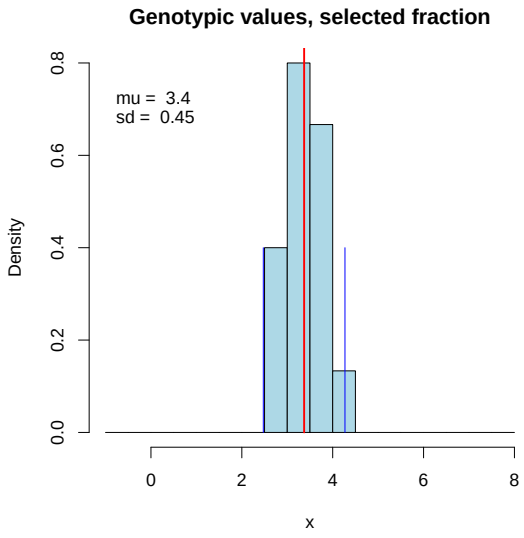
```
> p.C0sel <- get.population.pvalue("C0sel")$pvalue
> plt.hist (p.C0sel, main="Phenotypic values, selected fraction")
```



1.8 Genotypic values of the selected fraction

We determine the genotypic value of the selected fraction. Per definition the selection gain is the difference in the population means of the base population and the mean of the selected fraction.

```
> g.C0sel <- get.population.gvalue("C0sel")$gvalue   # Gen. values
> plt.hist (g.C0sel, main="Genotypic values, selected fraction")
```



1.9 Repeated simulation

We collect all code that is required for calculating selection gain and omit the code for plotting. This code is surounded by a loop for replications and defined as a function:

```
> sim01 <- function (h.sq, nsel, nrep) {
+   R <- rep(0,nrep)
+   for (i in 1:nrep) {
+     st.set.simpop ( pop.name="C0", data.set="C0" )
+     genotype.population("C0")
+     evaluate.population("C0")
+     phenotype.population("C0",h.sq)
+     population.sort("C0", decreasing=TRUE, selection.criterion="P")
+     population.divide("C0sel", "C0", nsel)
+     g.C0sel <- get.population.gvalue("C0sel")$gvalue
+     R[i] <- mean(g.C0sel) - mean(g)
+   }
+   return( mean(R) )
+ }
```

We now run the function with one set of parameters

```
> set.info.level(-1) # Only warnings and errors are printed, no messages
> sim01 ( h.sq=0.8, nsel=30, nrep=50 )
```

```
[1] 1.220982
```

Then the values are tabulated for different heritabilities and selection intensities:

```
> nsel <- c(100, 50, 30, 10 )
> h.sq <- c(0.6, 0.7, 0.8, 0.9, 0.99)
> nrep <- 100
> SG <- matrix(nrow=length(nsel),ncol=length(h.sq))
> rownames(SG) <- nsel
> colnames(SG) <- h.sq
> for (i in 1:length(nsel))
+   for (j in 1:length(h.sq)) {
+     cat (sprintf("nsel = %i, h.sq = %4.2f     \r",nsel[i],h.sq[j]))
+     SG[i,j] <- sim01 (nsel=nsel[i], h.sq=h.sq[j],  nrep=nrep )
+ }
> SG
```

```
> SG
          0.6       0.7       0.8      0.9      0.99
100 0.5511241 0.6132195 0.6430198 0.671797 0.7042547
50  0.8450615 0.9334265 0.9956299 1.040969 1.0878725
30  1.0577680 1.1767903 1.2070169 1.283798 1.3414516
10  1.4019869 1.5266215 1.5498781 1.656200 1.7221028
```

This procedure is characterized by the following features:

- Input is a data set with marker data, marker map, and field data
- Genetic architecture is estimated by genome wide prediction model
- Arbitrary genetic architecture of the trait is possible, no assumption of normally distributed traits
- Can be extended to plan number of locations, years, replications in field trials

1.10 Monitoring diversity

1.10.1 Mark selected fraction in a principal coordinate analysis

We assume a heritability of 0.8 and select 30 genotypes:

```
> nsel <- 30
> h.sq <- 0.8
> st.set.simpop ( pop.name="C0", data.set="C0" )
> genotype.population("C0")
> evaluate.population("C0")
> phenotype.population("C0",hsq=0.8)
> population.sort("C0", decreasing=TRUE, selection.criterion="P")
> population.divide("C0sel", "C0", nsel)
```

We then combine the selected individuals and the non selected ones in a new population, in which the first 30 plants are the selected ones and the remaining ones are the unselected ones. We use this property later for marking them with different colors in the plot.

```
> population.copy("t1","C0sel");
> population.copy("t2","C0")
> population.concat("t1","t2")     # Complete population, sorted
> st.get.simpop("t1","t1")         # Make a marker data set out of it
```

```
M (data set 't1'): New data set generated 't1'
M (data set 't1'): No. of individuals: 230, no. of markers: 828
```

We now calculate genetic distrances between the individuals, "mrd" is the Modified Rogers distance (see Reif et al. 2005),

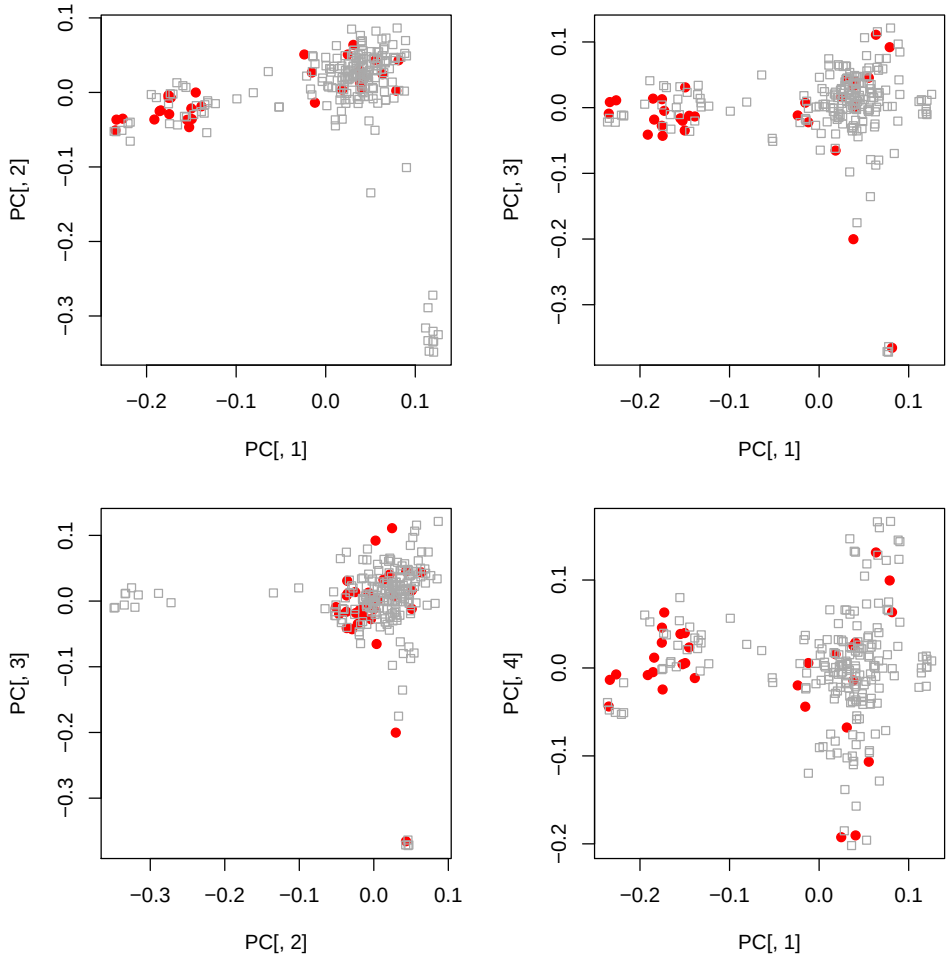
```
> dist.mat <- st.genetic.distances ( measure="mrd",
+                                    format="m" ,
+                                    data.set="t1")
```

carry out a principal coordinate analysis,

```
> PC <- cmdscale(dist.mat,4)      # Principlal coordinate analysis
```

and plot the results

```
> color  <- c( rep ("red",30),       # selected
+              rep ("darkgrey",201)) # not selected
> symbol <- c( rep (19,30),          # selected
+              rep (22,201))         # not selected
> oldpar <- par(no.readonly = TRUE)
> par(mfrow=c(2,2),mar=c(4,4,2,2))
> plot ( PC[,1], PC[,2], pch=symbol, col=color )
> plot ( PC[,1], PC[,3], pch=symbol, col=color )
> plot ( PC[,2], PC[,3], pch=symbol, col=color )
> plot ( PC[,1], PC[,4], pch=symbol, col=color )
> par(oldpar)
```



As a plant breeder we would now have a rough overview, whether the planned selection would cover a sufficient portion of the base population.

1.10.2 Plot gene diversity along the chromosomes

The gene diversity calculated from SNP data is often not considered to be useful, as we have only two variants of each SNP. Therefore we determine haplotype blocks and then determine the diversity for these haplotype blocks and plot it along the chromosomes. We first make a copy of the base population for haplotyping.

```
> set.info.level(0)
> st.copy.marker.data ( "t1h", source.data.set="default" )
```

```
M (data set 't1h'): New data set generated 't1h'
M (data set 't1h'): No. of individuals: 230, no. of markers: 828
```

Then we create a marker data set from the selected fraction

```
> st.get.simpop ( pop.name="C0sel", data.set="t3h" )
```

```
M (data set 't3h'): New data set generated 't3h'
M (data set 't3h'): No. of individuals: 30, no. of markers: 828
```

We have now `t1h` and `t3h` available for the haplotype analysis. The copies are reqiuered as haplotyping modifies the underlying populations and the result of a haplotype analysis ist a new marker data set that holds the haplotype blocks as ‘loci’ and the haplotype variants as ‘alleles’.

Haplytyping is a two-step procedure. In the first step the software goes through all chromosomes and determines the haplotype borders. This is what we are doing now. There are different possibilites to define the haplotype borders, here we use a very simple one, we make the border every three markers. State of the art would be to use a measure for linkage disequilibrium (Hedrick 1987), some of these are available in SelectionTools.

```
> st.def.hblocks ( hap = 3,            # Combine Markers in a 3 cM
+                  hap.unit=2,         # window to a haplotype block
+                  data.set="t1h" )
```

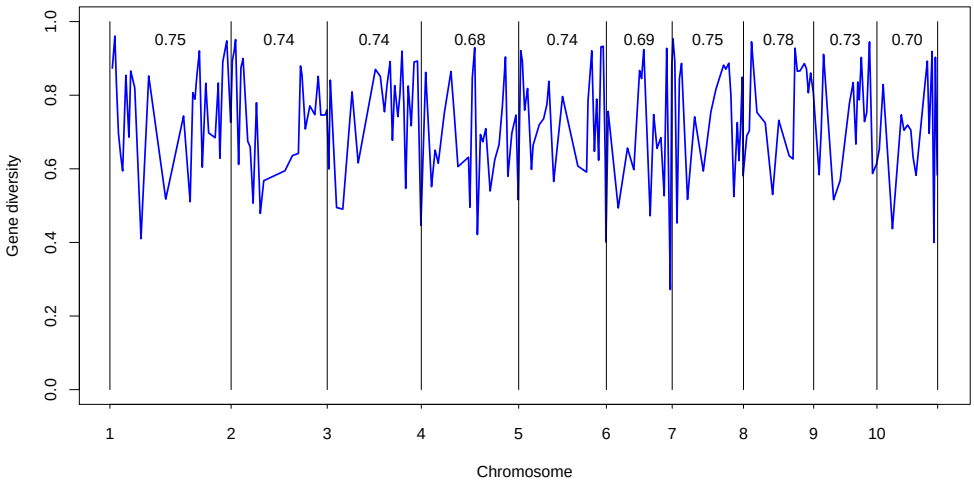
```
M (data set 't1h'): Haplotyp borders every 3 centiMorgans
```

The next step is to search for identical sequences of markers between these borders, and to determine what is somtimes called a haploblock. For inbred lines this is easy, this is the `.hil` in the function name 'haplotype blocks for inbred lines'. Other possibilites are available in SelectionTools, some require that you phase your data before determining the haplotype variants. There is an interface for the phasing software Beagle (Browning and Browning 2007).

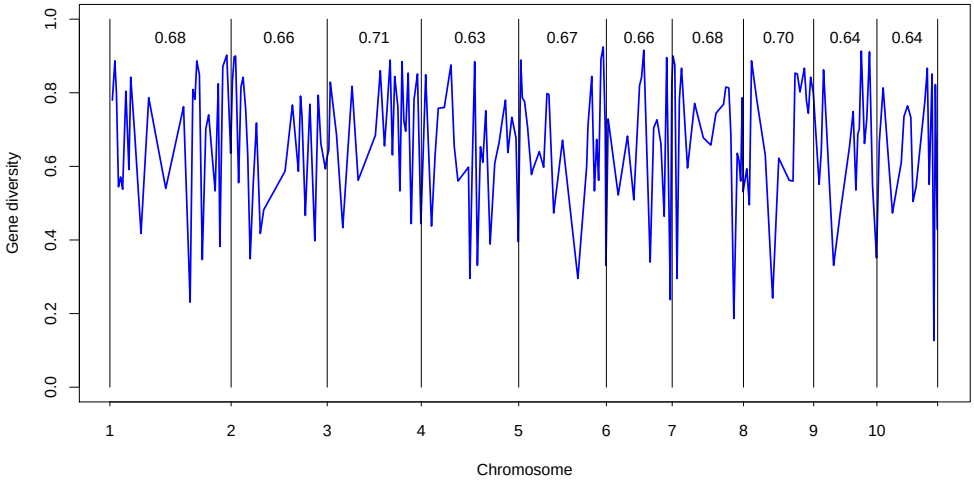
```
> st.recode.hil  (data.set="t1h")
```

```
M (data set 't1h'): No. of individuals: 230, no. of markers: 207
```

```
> st.plot.gene.diversity (data.set="t1h") # Base population
```

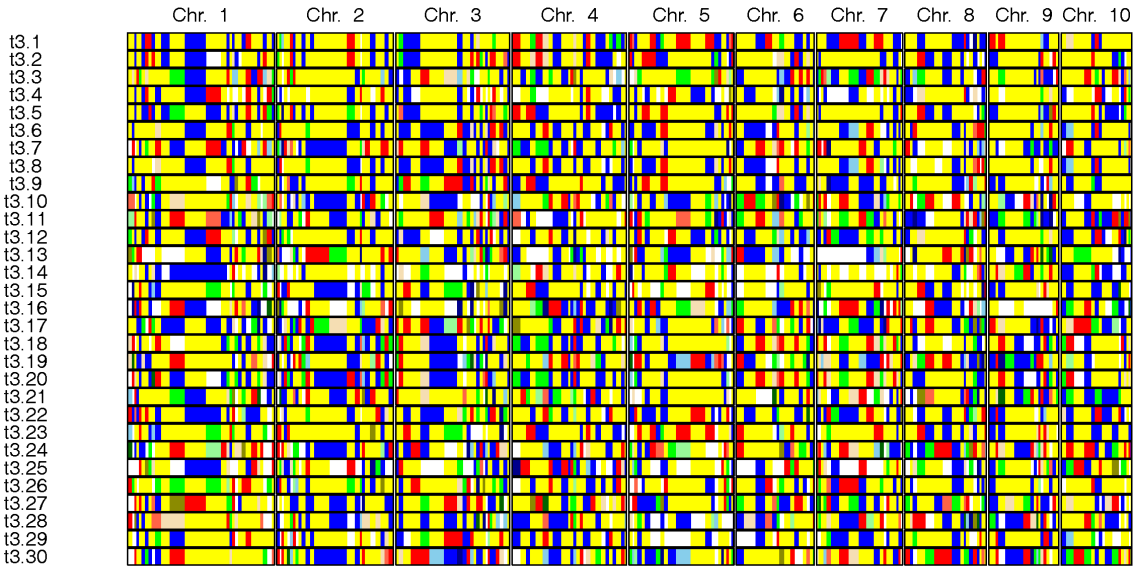



```
> st.def.hblocks ( hap = 3, hap.unit=2, data.set="t3h" )
> st.recode.hil ( data.set="t3h" )
> st.plot.gene.diversity (data.set="t3h") # Selected fraction
```



1.10.3 Plot the graphical genotypes of the selected fraction

```
> st.plot.ggt ( data.set = "t3h" )
```



2 Segregation variance and long term response to selection in DH *vs* SSD breeding schemes

2.1 Different variance depending on the population type

We consider a population of inbred lines, for which phenotypic data, marker data, and a linkage map are available. The best 30 lines are selected and recombined according to the following scheme: 1×2, 3×4, ..., 29×30. From each cross 20 lines are developped. We compare DH with S2 lines.

2.1.1 Select 30 best genotypes

We make a copy of the marker data and estimate and select the best lines

```
> st.set.info.level(-2)  # Errors only
> st.copy.marker.data ( "C0", source.data.set="default" )
> gs.esteff.rr ( method="BLUP", hsq=0.85, maxiter=0, data.set="C0")
> st.set.simpop ( pop.name="C0", data.set="C0" )
> genotype.population("C0")
> evaluate.population("C0")
> phenotype.population("C0",1)
> population.sort("C0", decreasing=TRUE,selection.criterion = "P")
> population.divide("C0sel", "C0", 30)
```

2.1.2 Split the selected fraction and recobine the genotypes

```
> population.copy("tmp","C0sel")
> for (ii in 1:30) {
+   nme <- sprintf("l%02i",ii)
+   population.divide (nme,"tmp",1)
+ }
> list.populations()[c(1:5,25:33),]
```

```
   PopName count
1      l30     1
2      l29     1
3      l28     1
4      l27     1
5      l26     1
25     l06     1
26     l05     1
27     l04     1
28     l03     1
29     l02     1
30     l01     1
31     tmp     0
32   C0sel    30
33      C0   200
```

2.1.3 Cross the lines and make C1-DHs

Cross 1x2, 3x4, 5x6, ... 29x30. 15 crosses. From each cross 20 DH lines are generated, and appended to the population `C1dh`

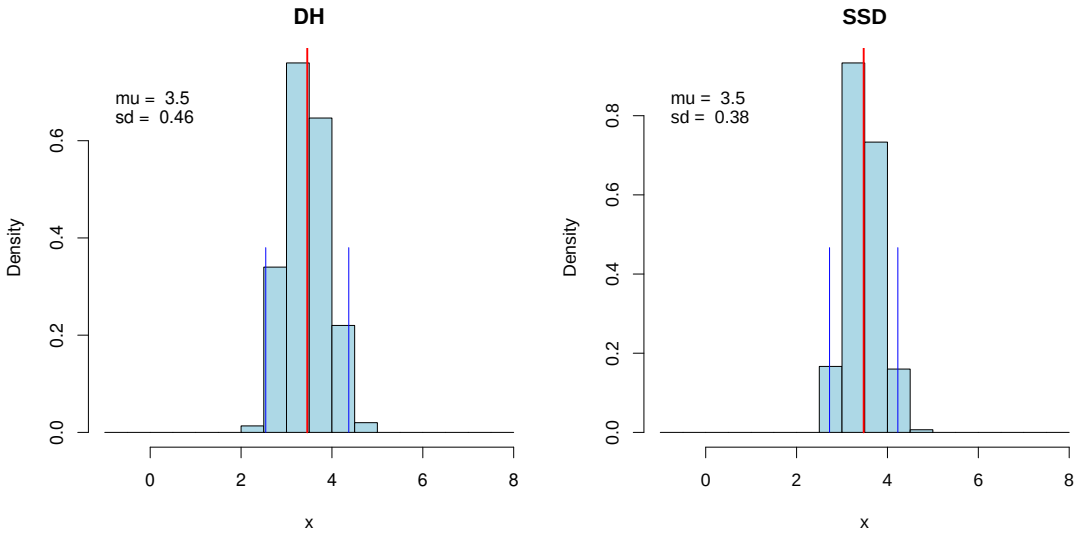
```
> for (ii in seq(1,29,2)) {
+   nme1 <- sprintf("l%02i",ii)
+   nme2 <- sprintf("l%02i",ii+1)
```

```
+   cross ("F1",nme1,nme2,1)
+   dh ("DH","F1",20)
+   population.append("C1dh","DH")
+ }


> genotype.population("C1dh")
> evaluate.population("C1dh")
> g.C1dh <- get.population.gvalue("C1dh")$gvalue
> plt.hist(g.C1dh,main="SSD-lines")
```

2.1.4 Cross the lines and make C1-SSDs

Cross 1x2, 3x4, 5x6, . . . 29x30. 15 crosses. From each cross 20 SSD lines are generated:

```
> for (ii in seq(1,29,2)) {
+   nme1 <- sprintf("l%02i",ii)
+   nme2 <- sprintf("l%02i",ii+1)
+   cross ("F1",nme1,nme2,1)
+   ssd.mating ("SSD","F1",20)
+   population.append("C1ssd","SSD")
+ }


> genotype.population("C1ssd")
> evaluate.population("C1ssd")
> g.C1ssd <- get.population.gvalue("C1ssd")$gvalue
> plt.hist(g.C1ssd)
```



The genetic segregation variance in DH lines is greater than in SSD lines. Therefore we shold be able to realize more selection gain with DH lines.

2.2 Long term response to selection

2.2.1 One cycle of selection

```
> st.set.info.level(-2)  # Errors only
> st.copy.marker.data ( "C0", source.data.set="default" )
> gs.esteff.rr ( method="BLUP", hsq=0.85, maxiter=0, data.set="C0")
> st.set.simpop ( pop.name="C0", data.set="C0" )
```

```
> hsq             <- 0.85
> n.selected      <- 30     #  n.selected/2 crosses
> lines.per.cross <- 20     #  15 crosses with 20 lines = 300 lines
> copy.population("base","C0")
> genotype.population("base")
> evaluate.population("base")
> mean(get.population.gvalue("base")$gvalue)
```

```
[1] 2.051092
```

```
> population.sort("base", decreasing=TRUE)
> population.divide("selected", "base", n.selected)
> remove.population("base")
> mean(get.population.gvalue("selected")$gvalue)
```

```
[1] 3.398705
```

Split the selected fraction in populations consisting of single lines

```
> for (ii in 1:n.selected) {
+   nme <- sprintf("l%02i",ii)
+   population.divide (nme,"selected",1)
+ }
```

Carry out the crosses, make DH lines, and append the lines to the base population of the next selection cycle.

```
> for (ii in seq(1,n.selected-1,2)) {
+   nme1 <- sprintf("l%02i",ii)
+   nme2 <- sprintf("l%02i",ii+1)
+   cross ("Fx",nme1,nme2,1)
+   dh ("DH","Fx",lines.per.cross)
+   population.append("base","DH")
+ }
```

Evaluate the base population for the next selection cycle.

```
> genotype.population("base")
> evaluate.population("base")
> mean(get.population.gvalue("base")$gvalue)
```

```
[1] 3.421653
```

2.2.2 Several cylces, replicated simulations

Define the simulation parameters

```
> st.set.simpop ( pop.name="C0", data.set="C0" )
> n.selected      <- 30
> lines.per.cross <- 20
> cycles          <- 10
> replications    <-  5
> perf <- matrix(0,nrow=cycles,ncol=replications)
```

The loop for the simulations

```
> for (r in 1:replications)
+   {
+     copy.population("base","C0")
+     genotype.population("base")
+     evaluate.population("base")
+     phenotype.population("base",hsq)
+     #
+     for (c in 1:cycles)
+       {
+         cat (sprintf("rep = %i, cycle = %i     \r",r,c))
+         population.sort("base", decreasing=TRUE, selection.criterion="P")
+         population.divide("selected", "base", n.selected)
+         remove.population("base")
+         #
+         for (ii in 1:n.selected) {
+           nme <- sprintf("l%02i",ii)
+           population.divide (nme,"selected",1)
+         }
+         #
+         for (ii in seq(1,n.selected-1,2)){
+           nme1 <- sprintf("l%02i",ii)
+           nme2 <- sprintf("l%02i",ii+1)
+           cross ("Fx",nme1,nme2,1)
+           dh ("DH","Fx",lines.per.cross)
+           population.append("base","DH")
+         }
+         genotype.population("base")
+         evaluate.population("base")
+         perf[c,r] <- mean(get.population.gvalue("base")$gvalue)
+         phenotype.population("base",hsq)
+       }
+   }
```

Listing the result

```
> perf
```

```
          [,1]     [,2]     [,3]     [,4]     [,5]
 [1,] 3.286169 3.241493 3.336559 3.318149 3.313786
 [2,] 4.070093 4.032043 4.128204 4.134745 4.181712
 [3,] 4.649487 4.580164 4.859451 4.685930 4.783495
 [4,] 5.028966 5.086314 5.331981 5.150224 5.237249
 [5,] 5.365961 5.625528 5.728588 5.593313 5.622474
 [6,] 5.857460 5.986584 6.166683 6.067476 5.966513
 [7,] 6.311010 6.379242 6.509935 6.607749 6.277857
 [8,] 6.638357 6.730861 6.778558 6.866087 6.644975
 [9,] 6.944484 7.064493 7.080740 7.155925 6.920576
[10,] 7.203681 7.329087 7.321460 7.442130 7.217765
```

```
> apply(perf,1,mean)
```

```
 [1] 3.471427 4.301084 4.816770 5.333437 5.783474 6.158898 6.536495
 [8] 6.876718 7.166341 7.372415
```

2.2.3 Carrying out a simulation study

We define one function that takes the parameters for on simulated scenario

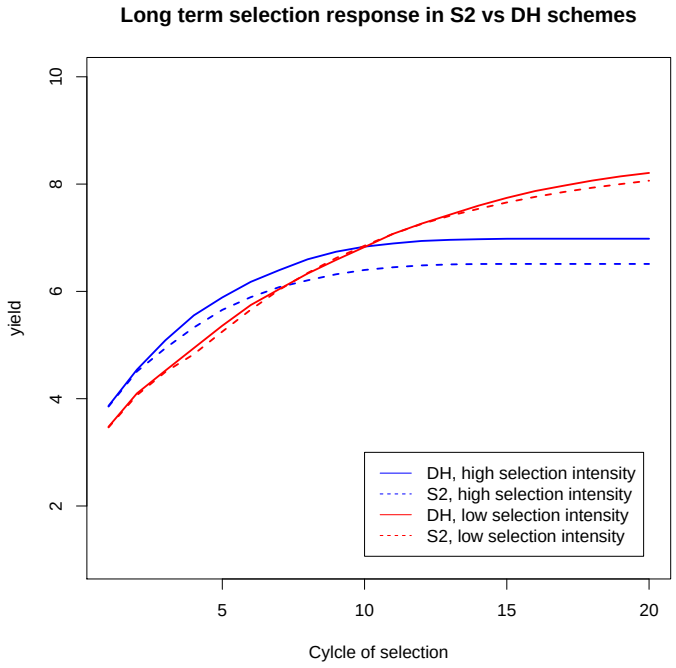
```
> lt.res <- function( n.selected      ,
+                     lines.per.cross ,
+                     cycles          ,
+                     replications    ,
+                     crossing.scheme  )
+ {
+   perf <- matrix(0,nrow=cycles,ncol=replications)
+   for (r in 1:replications)
+   {
+     copy.population("base","C0")
+     genotype.population("base")
+     evaluate.population("base")
+     phenotype.population("base",hsq)
+     #
+     for (c in 1:cycles)
+       {
+         cat (sprintf("\r rep = %i, cycle = %i\r",r,c))
+         population.sort("base", decreasing=TRUE, selection.criterion="P")
+         population.divide("selected", "base", n.selected)
+         remove.population("base")
+         #
+         for (ii in 1:n.selected) {
+           nme <- sprintf("l%02i",ii)
+           population.divide (nme,"selected",1)
+         }
+         #
+         if ("simple-dh"==crossing.scheme)
+           {
+             for (ii in seq(1,n.selected-1,2)){
+               nme1 <- sprintf("l%02i",ii)
+               nme2 <- sprintf("l%02i",ii+1)
+               cross ("Fx",nme1,nme2,1)
+               dh ("DH","Fx",lines.per.cross)
+               population.append("base","DH")
+             }
+           }
+          else if ("simple-ssd"==crossing.scheme)
+            {
+              for (ii in seq(1,n.selected-1,2)){
+                nme1 <- sprintf("l%02i",ii)
+                nme2 <- sprintf("l%02i",ii+1)
+                cross ("Fx",nme1,nme2,1)
+                ssd.mating ("SSD","Fx",lines.per.cross)
+                for (i in 1:1){
+                  copy.population("Parents","SSD")
+                  ssd.mating ("SSD","Parents",1)
+                }
+                population.append("base","SSD")
+              }
+            }
+         else {print("Error");return (NULL);}
+         genotype.population("base")
+         evaluate.population("base")
+         perf[c,r] <- mean(get.population.gvalue("base")$gvalue)
+         phenotype.population("base",hsq)
+       }
+   }
+   P <- apply(perf,1,mean)
+   return (P)
+ }
```

Then we run four scenarios

```
> P1 <-
+ lt.res ( n.selected      =  10,
+          lines.per.cross =  50,
+          cycles          =  20,
+          replications    = 100,
+          crossing.scheme = "simple-dh")
> P2 <-
+ lt.res ( n.selected      =  10,
+          lines.per.cross =  50,
+          cycles          =  20,
+          replications    = 100,
+          crossing.scheme = "simple-ssd")
> P3<-
+ lt.res ( n.selected      =  30,
+          lines.per.cross =  10,
+          cycles          =  20,
+          replications    = 100,
+          crossing.scheme = "simple-dh")
> P4 <-
+ lt.res ( n.selected      =  30,
+          lines.per.cross =  10,
+          cycles          =  20,
+          replications    = 100,
+          crossing.scheme = "simple-ssd")
```

and print the final result

```
> plot ( 1:length(P1), P1, type="n", ylim=c(1,10),
+       xlab="Cylcle of selection", ylab="yield",
+       main="Long term selection response in S2 vs DH schemes")
> points ( 1:length(P1), P1, type="l",lty=1,lwd=2,col="blue" )
> points ( 1:length(P2), P2, type="l",lty=2,lwd=2,col="blue" )
> points ( 1:length(P3), P3, type="l",lty=1,lwd=2,col="red" )
> points ( 1:length(P4), P4, type="l",lty=2,lwd=2,col="red" )
> legend(10,3,legend=c("DH, high selection intensity",
+                      "S2, high selection intensity",
+                      "DH, low selection intensity",
+                      "S2, low selection intensity"),
+        col=c("blue","blue","red","red"),
+        lty=c(1,2,1,2))
```



3 Data structures and VCF import and export

SelectionTools uses `STvcf` as a compact exchange format for diploid marker genotypes together with genetic map positions. It follows VCF allele indexing, with index 0 for the reference allele and indices 1, 2, ... for alternative alleles, but it is not intended to retain all fields of a VCF file. The object contains only the information needed for transfer between R, the SelectionTools marker-data structures, and the simulation routines.

An `STvcf` object is a list of class `STvcf` with nine components: `CHROM`, `POS`, `ID`, `REF`, `ALT`, `individual`, `allele1`, `allele2`, and `CHROM.levels`. `CHROM` contains consecutive chromosome codes, `POS` contains genetic positions in cM, and `CHROM.levels` stores the corresponding chromosome labels. The two allele matrices contain one VCF allele index for each homologue and each marker-individual combination.

Marker and individual identifiers must be unique, nonempty alphanumeric ASCII strings shorter than 256 bytes. Only diploid genotypes are supported. Multiallelic markers are allowed, with at most 253 ALT alleles per marker. The compact representation normally uses raw allele matrices; raw value 255 denotes missing data. Allele order is retained, but the original slash or vertical-bar separator is not retained, so STvcf does not imply statistical phase inference.

Four functions provide the main conversion path:

`st.dataframe.to.STvcf()` converts a VCF-like R data frame to STvcf;

`st.load.vcf.data()` loads STvcf into a SelectionTools marker data set and installs its linkage map;

`st.markerdata.to.STvcf()` converts a SelectionTools marker data set back to STvcf; and

`st.STvcf.to.dataframe()` expands STvcf to a VCF-like data frame.

The detailed validation rules for these functions are given in their help pages. The most important conventions are summarized below.

3.1 Import from VCF-like data

For `st.dataframe.to.STvcf()`, the first five columns must be `CHROM` or `#CHROM`, `POS`, `ID`, `REF`, and `ALT`, in that order. If a valid `CM` metadata column is present, its values are used as genetic positions; otherwise `POS` is interpreted as cM. This distinction is important for ordinary VCF files, in which `POS` usually denotes a physical base-pair position.

A `FORMAT` column is optional. When present, it must define exactly one nonempty `GT` field for every marker. Only GT is retained; other FORMAT subfields and VCF metadata such as QUAL, FILTER, and INFO are not represented in STvcf. Both `0/1` and `0|1` are accepted. Missing and partially missing genotypes are normalized to the STvcf missing representation. REF/ALT definitions and all observed allele indices are checked during conversion.

The converter is deliberately strict: it is intended to create only objects that can subsequently be loaded by SelectionTools. Full rules for names, FORMAT layouts, missing values, REF/ALT definitions, and genotype syntax are documented in `?st.dataframe.to.STvcf`.

A small example is:

```
vcf <- data.frame(
   CHROM  = c("1", "1"),
   POS    = c(102341, 284020),
   ID     = c("M1", "M2"),
   REF    = c("A", "C"),
   ALT    = c("G", "T"),
   CM     = c(1.25, 3.80),
   FORMAT = c("GT", "GT"),
   S1     = c("0/0", "0/1"),
   S2     = c("0|1", "./."),
   check.names = FALSE
)

x <- st.dataframe.to.STvcf(vcf)
```

3.2 Transfer between STvcf and SelectionTools

`st.load.vcf.data()` converts an STvcf object to the ordinary SelectionTools marker and linkage-map structures. Thus STvcf is an exchange format and not a second internal analysis representation. The object is validated again when it is loaded. Marker statistics are then constructed from the imported marker set. Multiallelic markers can be loaded, but the ordinary genomic-selection routines that use one design-matrix column per marker require an appropriate biallelic marker set.

The reverse conversion is performed with `st.markerdata.to.STvcf()`. SelectionTools marker data do not retain the historical textual VCF REF and ALT labels. Consequently, export normalizes the observed allele states at each marker and creates new VCF-style REF/ALT labels while preserving the genotype states and the current linkage-map positions. Only marker genotypes and genetic-map information are part of STvcf; performance data, estimated effects, and other derived quantities are not exported.

Simulation populations can be included in the same chain. First use `st.get.simpop()` to return a simulation population to a marker data set and then use `st.markerdata.to.STvcf()`. Individual identifiers created by `st.get.simpop()` are new simulation-derived names, for example `P1I1`, `P1I2`, ...; original pre-simulation names are not reconstructed.

```
data("v-tropmaize-vcf")
st.load.vcf.data(v.tropmaize.vcf, data.set = "vcf01")

x2 <- st.markerdata.to.STvcf(data.set = "vcf01")
df2 <- st.STvcf.to.dataframe(x2)
```

```
## Reverse transfer from a simulation population:
st.get.simpop("P1", data.set = "fromsim")
sim.vcf <- st.markerdata.to.STvcf(data.set = "fromsim")
```

3.3 Export to a VCF-like data frame

`st.STvcf.to.dataframe()` returns the fixed columns `CHROM`, `POS`, `ID`, `REF`, `ALT`, and `FORMAT`, followed by one genotype column per individual. FORMAT is written as `GT`. Genotype alleles are written with VCF indices and a slash separator; missing genotypes are written as `./.`. Because STvcf does not retain QUAL, FILTER, INFO, non-GT FORMAT fields, phase-set information, or an original physical POS that was replaced by CM, these fields cannot be reconstructed.

Conversion from a VCF-like data frame to STvcf and back therefore preserves the STvcf information, not the complete original VCF representation.

4 References

Browning SR, Browning BL (2007) Rapid and accurate haplotype phasing and missing-data inference for whole-genome association studies by use of localized haplotype clustering. American Journal of Human Genetics 81:1084–1097.

Crossa J, de los Campos G, Pérez P, Gianola D, Burgueño J, et al. (2010) Prediction of genetic values of quantitative traits in plant breeding using pedigree and molecular markers. Genetics 186:713–724.

Habier D, Fernando RL, Dekkers JCM (2007) The impact of genetic relationship information on genome-assisted breeding values. Genetics 177:2389–2397.

Hedrick PW (1987) Gametic disequilibrium measures: proceed with caution. Genetics 117:331–341.

Hofheinz N, Frisch M (2014) Heteroscedastic ridge regression approaches for genome-wide prediction with a focus on computational efficiency and accurate effect estimation. G3 4:539–546.

Reif JC, Melchinger AE, Frisch M (2005) Genetical and Mathematical Properties of Similarity and Dissimilarity Coefficients Applied in Plant Breeding and Seed Bank Management. Crop Science 45:1-7

Maurer HP, Melchinger AE, Frisch M (2008) Population genetic simulation and data analysis with Plabsoft. Euphytica 161:133–139.

Meuwissen THE, Hayes BJ, Goddard ME, (2001) Prediction of total genetic value using genome-wide dense marker maps. Genetics 157:1819–1829.

Shen X, Alam M, Fikse F, Rönnegård L, (2013) A novel generalized ridge regression method for quantitative genetics. Genetics 193:1255–1268.

Strandén I, Christensen OF, (2011) Allele coding in genomic evaluation. Genetics Selection Evolution 43:25. Equation (2).

Whittaker JC, Thompson R, Denham MC, (2000) Marker-assisted selection using ridge regression. Genet Res 75:249–252.