
Simulating Breeding Programs

Matthias Frisch

1	Selection gain in one generation of phenotypic selection and monitoring genetic diversity in selection programs	2
1.1	Data	2
1.2	Preprocessing the marker data	4
1.3	Estimation of marker effects	5
1.4	Initialization of the simulation routines	6
1.5	Distribution of genotypic values in the base population	6
1.6	Distribution of the phenotypic values after a field trial	7
1.7	Selection for phenotypic values	8
1.8	Genotypic values of the selected fraction	9
1.9	Repeated simulation	9
1.10	Monitoring diversity	10
1.10.1	Mark selected fraction in a principal coordinate analysis	10
1.10.2	Plot gene diversity along the chromosomes	12
1.10.3	Plot the graphical genotypes of the selected fraction . .	13
2	Segregation variance and long term response to selection in DH vs SSD breeding schemes	14
2.1	Different variance depending on the population type	14
2.1.1	Select 30 best genotypes	14
2.1.2	Split the selected fraction and recobine the genotypes .	14
2.1.3	Cross the lines and make C1-DHs	14
2.1.4	Cross the lines and make C1-SSDs	15
2.2	Long term response to selection	15
2.2.1	One cycle of selection	15
2.2.2	Several cylces, replicated simulations	16
2.2.3	Carrying out a simulation study	18
3	Data structures and VCF import and export	21
3.1	Convert VCF-like data frame to STvcf	21
3.2	Import STvcf into SelectionTools	23
3.3	Export SelectionTools marker data to STvcf	24
3.4	Convert STvcf to a VCF-like data frame	26
4	References	27

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 phenotypic 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)
```

```
M (data set 'default'): No. of individuals: 264, no. of markers: 1076
```

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

```
M (data set 'default'): No. of individuals: 264, no. of markers: 1076
M (data set 'default'): No. of performance data: 264
```

and look at the loaded data

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

```
M (data set 'default'): No. of individuals: 264, no. of markers: 1076
```

```
> 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 `$genotypes` lists the complete marker data. In `$individual.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 than 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
```

```
M (data set 'default'): No. of individuals: 264, no. of markers: 867
```

```
> st.restrict.marker.data (ExHet.MIN=0.1) # Minimum gene diversity
```

```
M (data set 'default'): No. of individuals: 264, no. of markers: 828
```

```
> st.restrict.marker.data (InMis.MAX=0.1) # Max. missing per individual
```

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

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 ()
```

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

```
> 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" )
```

```
M (data set 'C0'): New data set generated 'C0'  
M (data set 'C0'): No. of individuals: 230, no. of markers: 828
```

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 ‘animal model’ based G-BLUP (Habier et al. 2007)

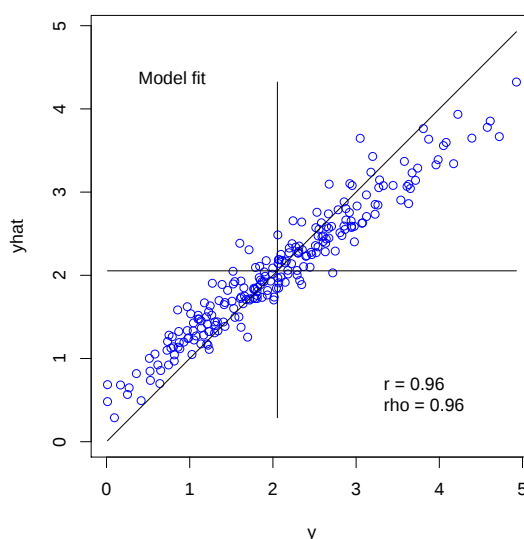
```
> gs.esteff.rr ( method="BLUP", data.set="C0")
```

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. These genotypic values are used for the simulations.

After having carried out the effect estimation, the marker data set holds 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 transformation 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 simulation 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)
+ }

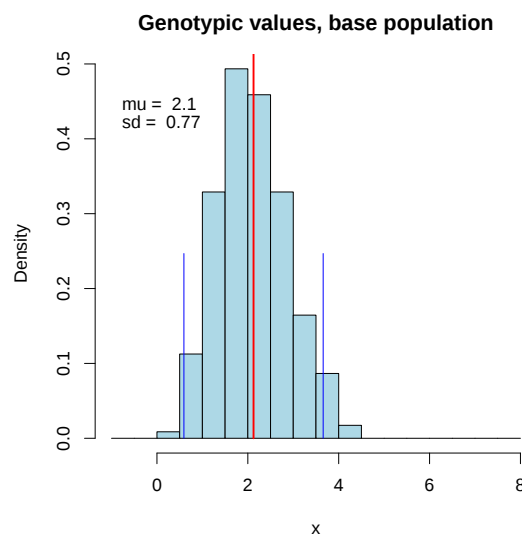
```

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

```

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

```



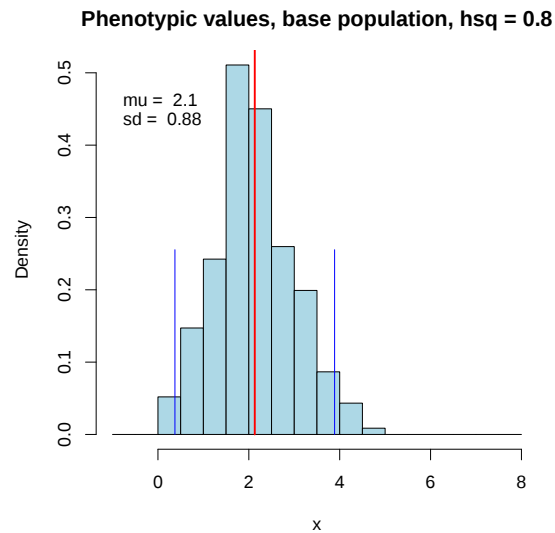
1.6 Distribution of the phenotypic values after a field trial

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

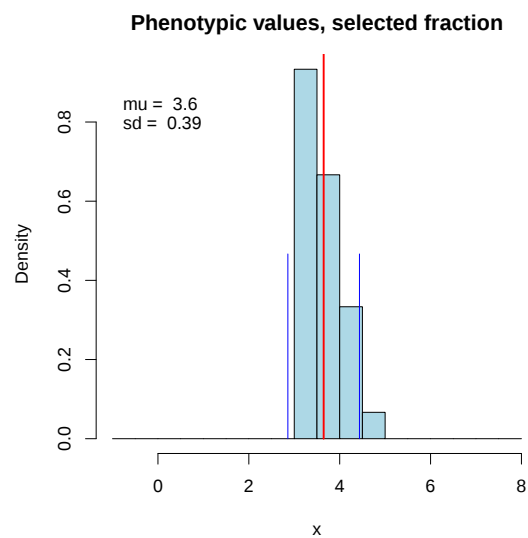
Now we sort the population according to the genotypic and put the 30 lines with the greatest score into a new population called C0sel

```
> 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.

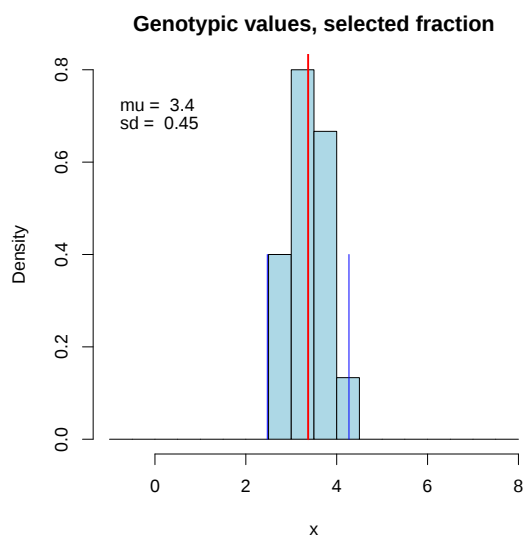
```
> p.C0sel <- get.population.pvalue("C0sel")$pvalue
> plt.hist (p.C0sel, main="Phenotypic values, selected fraction")
```



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 surrounded 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 distances between the individuals, "mrd" is the Modified Rogers distance (see Reif et al. 2005),

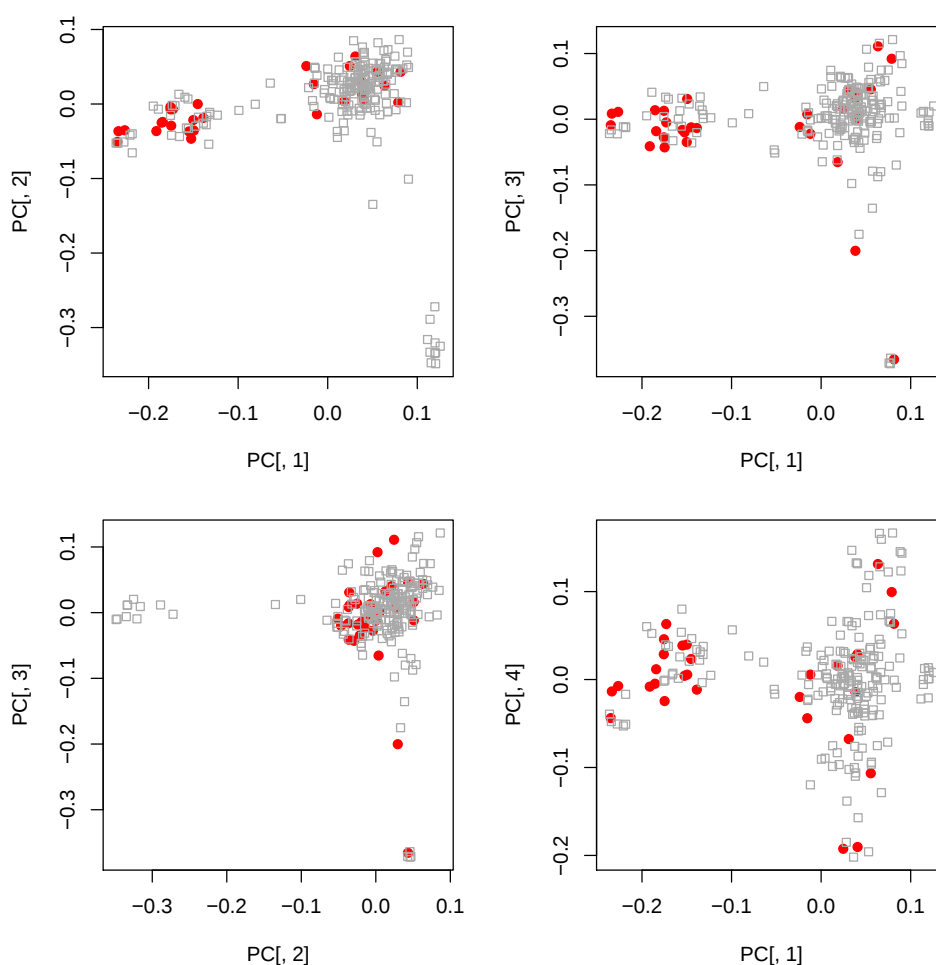
```
> dist.mat <- st.genetic.distances ( measure="mrd",
+                                   format="m" ,
+                                   data.set="t1")
```

carry out a principal coordinate analysis,

```
> PC <- cmdscale(dist.mat,4)      # Principal 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 required as haplotyping modifies the underlying populations and the result of a haplotype analysis is a new marker data set that holds the haplotype blocks as 'loci' and the haplotype variants as 'alleles'.

Haplotyping 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 possibilities 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" )
```

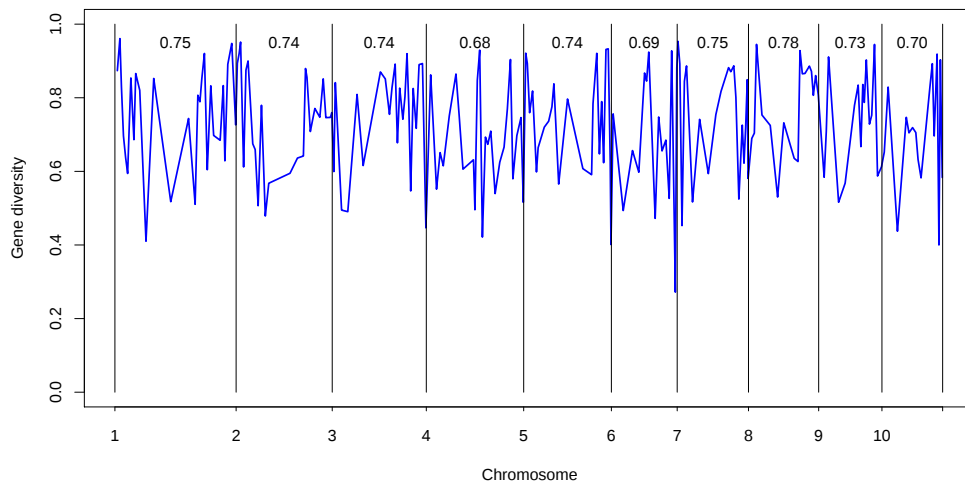
```
M (data set 't1h'): Haplotyp borders every 3 centiMorgans
```

The next step is to search for identical sequences of markers between these borders, and to determine what is sometimes called a haploblock. For inbred lines this is easy, this is the .hil in the function name 'haplotype blocks for inbred lines'. Other possibilities 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).

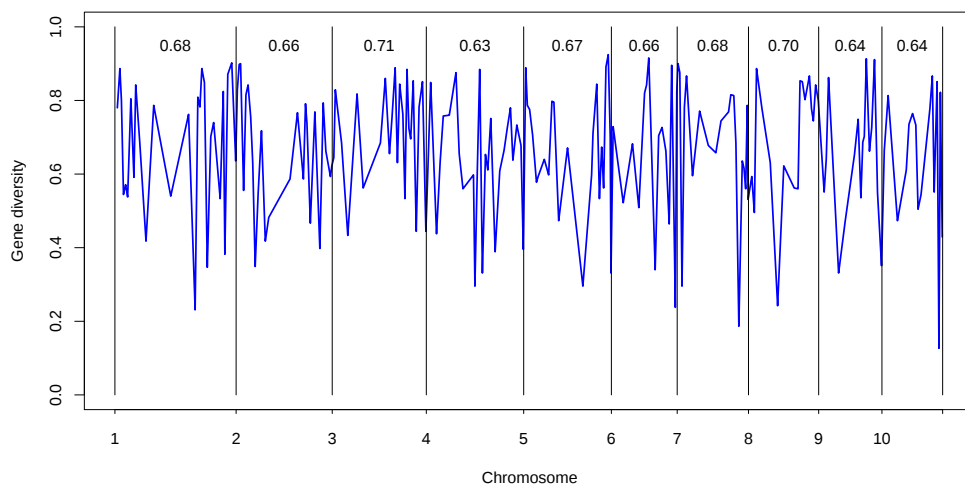
```
> st.recode.hil (data.set="t1h")
```

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

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

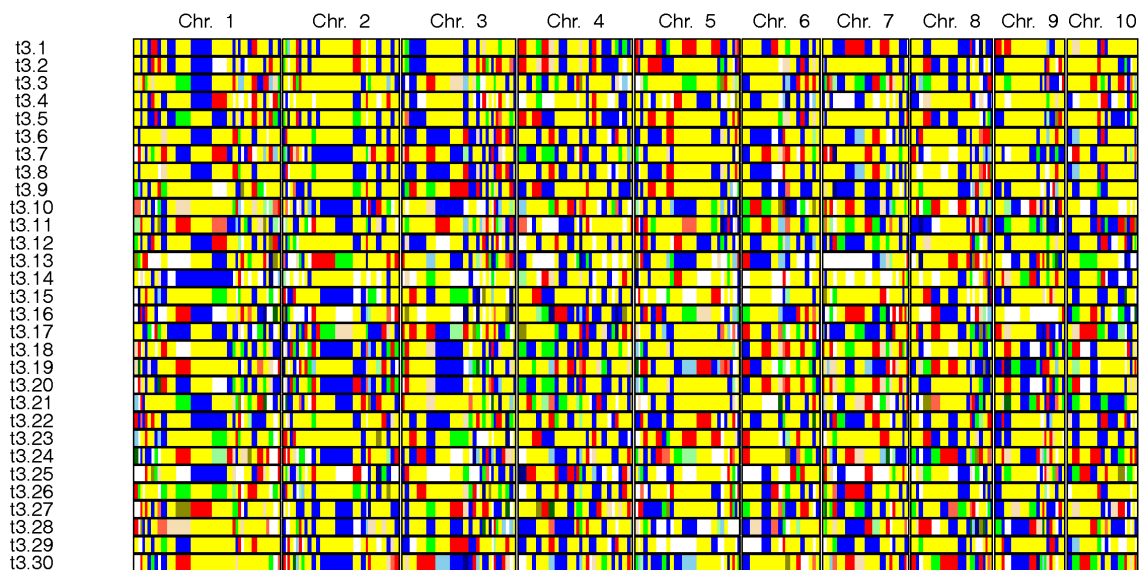


```
> st.def.hblocks ( hap = 3, hap.unit=2, data.set="t3h" )
> st.recode.hil ( data.set="t3h" )
> st.plot.gene.diversity (data.set="t3h") # Selected fraction
```



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: $1 \times 2, 3 \times 4, \dots, 29 \times 30$. From each cross 20 lines are developed. 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 $1 \times 2, 3 \times 4, 5 \times 6, \dots, 29 \times 30$. 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

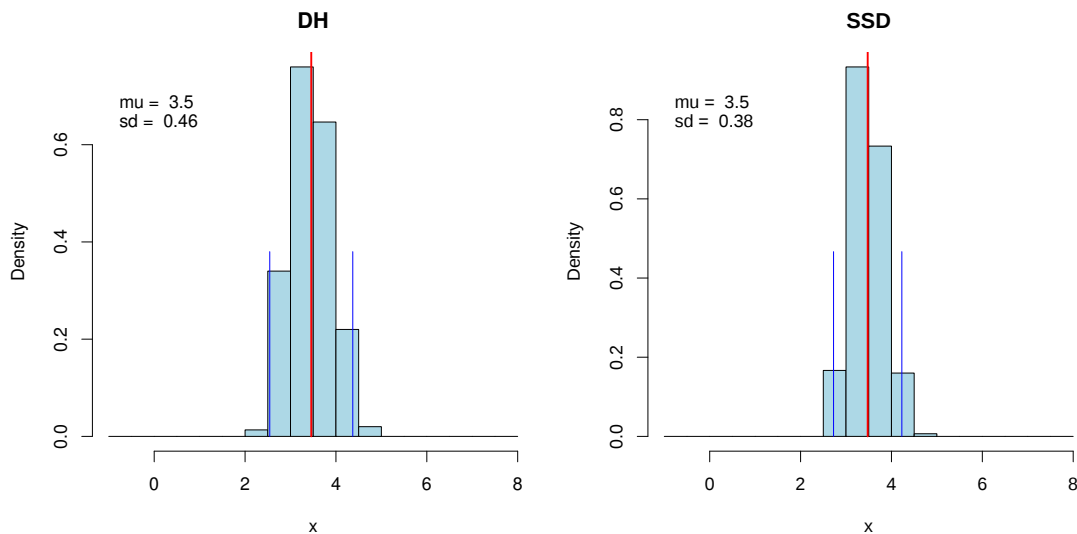
Cross 1x2, 3x4, 5x6, ... 29x30. 15 crosses. From each cross 20 SSD lines are generated:

```

> for (ii in seq(1,29,2)) {
+   nme1 <- sprintf("%02i",ii)
+   nme2 <- sprintf("%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 should 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] 3.409638
```

2.2.2 Several cycles, 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)
+ }
```

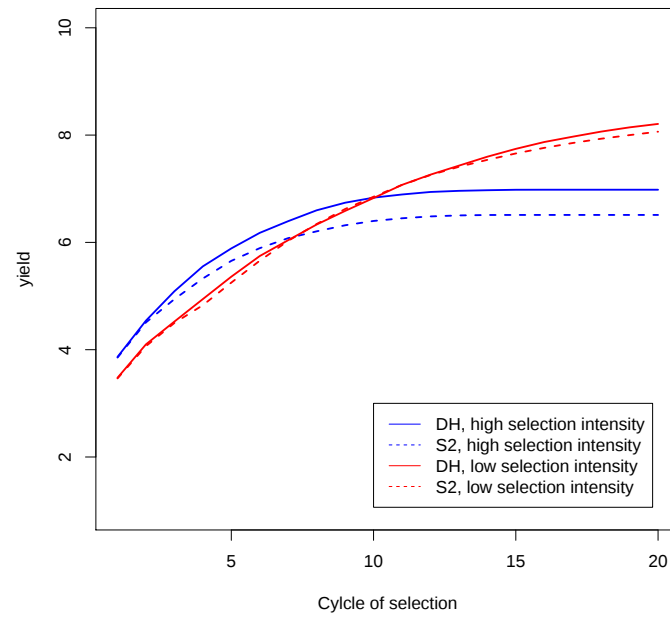
Then we run four scenarios

```
> 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))
```

Long term selection response in S2 vs DH schemes



3 Data structures and VCF import and export

SelectionTools uses the compact STvcf R representation as an exchange format for diploid marker genotypes together with the genetic linkage-map position of every marker. The representation follows the VCF convention that allele index 0 denotes the reference allele and allele indices 1, 2, ... refer to alternative alleles. It is deliberately not a complete VCF object. Only the information required for the SelectionTools marker-data and simulation workflow is retained.

An STvcf object is a list of class STvcf with exactly nine components:

CHROM Integer running chromosome codes. The codes start at 1, are consecutive, and every code is used.

POS Double-precision genetic map positions in cM. Positions must be finite and nonnegative.

ID Unique marker identifiers.

REF One reference-allele label for each marker.

ALT Either . or a comma-separated list of alternative allele labels.

individual Unique individual identifiers.

allele1 A marker-by-individual raw or integer matrix containing the first VCF allele index.

allele2 A marker-by-individual raw or integer matrix containing the second VCF allele index.

CHROM.levels Character labels corresponding to the running chromosome codes.

Marker and individual identifiers used by these interfaces contain only ASCII letters A-Z, a-z, and digits 0-9, must be nonempty and unique, and must contain fewer than 256 bytes. Names are never silently modified. In particular, dots, underscores, hyphens, whitespace, punctuation, and non-ASCII characters are not allowed.

The compact R representation uses two allele matrices rather than character strings such as 0/1. If all allele indices that can occur are at most 254, the matrices are raw matrices and raw value 255 represents missing data. Both matrices always have the same storage type. This is only a storage optimization; it does not change genotype meaning.

The STvcf representation is compact in R and on disk. When an object is loaded, the genotypes are expanded into the ordinary SelectionTools C marker and linkage-map structures. STvcf therefore does not introduce a second internal analysis representation.

Four functions are available for data conversion: `st.dataframe.to.STvcf()` converts VCF-like tabular data to STvcf, `st.STvcf.to.dataframe()` expands STvcf to a VCF-like data frame, `st.load.vcf.data()` installs STvcf in a SelectionTools marker data set, and `st.markerdata.to.STvcf()` converts an existing SelectionTools marker data set back to STvcf. The examples in this section are shown as code only and are not evaluated while the vignette is built.

3.1 Convert VCF-like data frame to STvcf

`st.dataframe.to.STvcf(data)` converts a wide diploid VCF-like R data frame to the compact STvcf representation used by `st.load.vcf.data()`. The converter is deliberately strict. Its purpose is to create only objects that can subsequently be accepted by the SelectionTools marker-data and simulation-transfer path. Invalid marker or individual names, malformed genotypes, inconsistent REF/ALT definitions, invalid genetic-map positions, and unsupported allele indices are rejected during conversion rather than being deferred to a later consumer.

The first five columns are fixed by position and must occur in the following order:

CHROM or #CHROM, POS, ID, REF, ALT.

The positional convention is intentional. Sample names such as POS, INFO, or FORMAT are legal because they are alphanumeric, so fixed fields cannot safely be identified merely by searching all column names. All data-frame column names must be non-missing and nonempty. Duplicate column names are permitted only where a sample happens to have the same name as a fixed or metadata field; sample names themselves must be unique.

Marker IDs are mandatory and unique. Marker and sample names obey the common SelectionTools/STvcf name convention described above. These restrictions are checked again by `st.load.vcf.data()` and by the reverse marker-data conversion.

Input chromosome labels can be numeric or character. They are replaced by running integer codes 1, 2, ... in order of first appearance, while the original labels are stored in `CHROM.levels`. An information message at level 0 reports this mapping. Chromosome labels are not marker or individual identifiers and are therefore not subject to the alphanumeric identifier restriction.

STvcf positions are genetic positions in centimorgan. If a valid metadata column named CM is supplied, it is used for the genetic position and the input POS column is not used for mapping. Otherwise POS is interpreted as cM. This distinction is important for conventional VCF data, where POS normally contains a physical base-pair position. Factor CM or POS columns are converted through their printed character values before numeric conversion, so factor level numbers can never accidentally become map positions. Genetic positions are stored as R double-precision values without intentional rounding and must be finite and nonnegative.

A FORMAT column is recognized only when every marker-specific FORMAT definition is nonempty and contains exactly one nonempty GT field. Empty FORMAT fields, empty FORMAT subfields, a trailing colon, and duplicated GT fields are invalid. The FORMAT definition may differ among markers. When FORMAT is present, columns between ALT and FORMAT may be the VCF metadata fields QUAL, FILTER, INFO, and the SelectionTools-specific CM field. These metadata fields must be unique. Every column after FORMAT is treated as a sample column irrespective of its name. Thus sample identifiers such as POS, INFO, CM, and FORMAT remain unambiguous.

For each sample cell only the GT subfield specified by the marker-specific FORMAT definition is retained. Other FORMAT subfields such as DP, AD, GQ, and phase-set fields are ignored and are not stored. A non-missing sample field that is too short to contain the declared GT subfield, or that contains an empty GT subfield, is malformed and is rejected rather than converted to missing.

If FORMAT is absent, all columns after the first five fixed fields are direct GT sample columns except for one special case. An alphanumeric column named CM whose complete contents are valid, finite, nonnegative numeric values is interpreted as the genetic-map position field rather than as a sample. This is the convention used to distinguish an optional CM column from a sample that itself is named CM. More than one column that can be interpreted as FORMAT, or more than one valid CM column in the no-FORMAT layout, is rejected.

REF must contain exactly one non-missing allele label. It may not be empty, ., or comma-separated. ALT may be ., meaning that no alternative allele is defined, or a comma-separated list. Every ALT entry must be nonempty, may not be ., must differ from REF, and must be unique within the marker. A trailing comma is therefore invalid. Definitions such as REF=A with ALT=A or ALT=C,C are rejected.

Multiallelic markers are accepted. VCF allele index 0 denotes REF, index 1 the first ALT allele, index 2 the second ALT allele, and so on. Every observed non-missing allele index must be present in the corresponding REF/ALT definition. At most 65533 ALT alleles can be defined for a marker, matching the range that can subsequently be represented by the SelectionTools simulation transfer. The SelectionTools marker-statistics backend permits at most 4096 observed allele states at one marker, including the missing state when it occurs; this observed-state limit is checked during conversion.

Only diploid genotypes are supported. Both slash and vertical-bar separators are accepted, for example 0/1 and 0|1. The supplied allele order is retained in `allele1` and `allele2`, but the separator itself is not stored. Retaining the allele order is not a statistical phasing procedure and does not imply phase inference. Phase-set boundaries and other phase metadata are outside the STvcf representation.

Complete missingness can be represented by R NA, an empty field, `.`, `./.`, `.|.`, or the character value NA. Partially missing diploid genotypes such as 0/., ./1, 0|., and .|1 are normalized to complete missingness. The non-missing member of such a partial pair must nevertheless be a syntactically valid nonnegative integer allele index within the marker's REF/ALT definition. Malformed fields, including an empty allele component or a nonnumeric non-missing component, are rejected rather than being hidden by the partial-missing rule.

The returned object has class STvcf and the nine components listed at the beginning of this section. Marker order and individual order are retained. Marker ordering changes only later, when an STvcf object is installed in the internal SelectionTools linkage-map structure.

The conversion is not a byte-for-byte VCF round trip. QUAL, FILTER, INFO, FORMAT subfields other than GT, phase-set information, a physical POS field when CM is supplied, and the distinction between slash and vertical-bar GT separators are not retained.

A small non-running example with physical POS and an explicit genetic CM column is:

```
vcf <- data.frame(
  CHROM = c("1", "1", "2"),
  POS   = c(102341, 284020, 91733),
  ID    = c("M1", "M2", "M3"),
  REF   = c("A", "C", "G"),
  ALT   = c("G", "T", "A,C"),
  CM    = c(1.25, 3.80, 0.60),
  FORMAT = c("GT:DP", "GT:DP", "GT:DP"),
  S1    = c("0/0:18", "0/1:21", "2/1:16"),
  S2    = c("0|1:23", "./.:0", "0/2:19"),
  check.names = FALSE,
  stringsAsFactors = FALSE
)

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

3.2 Import STvcf into SelectionTools

`st.load.vcf.data(STvcf, data.set="default")` loads compact STvcf marker and linkage-map data directly into a SelectionTools marker data set without intermediate marker or map text files. The STvcf object is an exchange representation; loading converts it to the existing SelectionTools marker and map structures rather than creating a parallel analysis representation.

The C loader independently requires exactly the nine STvcf components and validates them again. Thus manually assembled objects cannot bypass the interface rules. In particular, marker and individual names must be unique, nonempty, contain only ASCII letters and digits, and contain fewer than 256 bytes. Dots, underscores, hyphens, whitespace, punctuation, and non-ASCII characters are rejected rather than sanitized.

The required components have the following meaning when loaded: `CHROM` contains consecutive running chromosome codes from 1; `POS` contains finite, nonnegative double-precision cM positions; `ID` contains marker identifiers; `REF` contains one reference label; `ALT` contains `.` or a comma-separated ALT list; `individual` contains individual identifiers; `allele1` and `allele2` are marker-by-individual raw or integer matrices; and `CHROM.levels` contains the character labels associated with the running chromosome codes.

VCF allele index 0 is translated to internal SelectionTools allele 1, index 1 to internal allele 2, index 2 to internal allele 3, and so on. Missing alleles become internal value -1. Raw STvcf

matrices use 255 for missing and can represent allele indices through 254. Integer matrices use `NA_integer_` for missing data. The two allele matrices must have the same type. Allele definitions and observed indices must fit the range of the diploid `SelectionTools` simulation transfer.

REF and ALT are validated independently of the R converter. A REF value repeated in ALT, a duplicated ALT value, an empty ALT token, a trailing comma, or a dot embedded inside an ALT list is rejected. Thus a malformed manually constructed object cannot create separate internal integer alleles for identical textual allele labels.

The marker-statistics backend permits at most 4096 observed states at one marker, counting missing when it occurs. Multiallelic markers are valid input if they satisfy this limit. The ordinary statistical genomic-selection functions that build one design-matrix column per marker are biallelic; multiallelic data must therefore be restricted to an appropriate biallelic marker set before those functions are used. Monomorphic markers are likewise unsuitable for that ordinary biallelic design-matrix construction.

Only diploid data are supported. If either stored allele is missing, both internal alleles of that genotype are set to -1. The order of two non-missing alleles is retained. These two arrays become the two homologues when marker data are transferred to the simulation routines, but retaining the order does not constitute statistical phase inference.

Every STvcf marker must have a genetic position. Markers are installed in chromosome and cM order. Exact position ties retain STvcf row order and are separated by small deterministic internal offsets because the existing `SelectionTools` linkage-map and simulation structures require distinct ordered positions. The supplied STvcf object itself is not changed. Consequently, `st.get.map()` reports the map positions actually used internally, which may include these small tie-separating offsets.

Normal `SelectionTools` marker statistics are constructed after conversion and are calculated from exactly the markers present in the STvcf object. The compact raw or integer R matrices are expanded into the existing C genotype and marker-statistics structures during loading.

Replacement of `data.set` is transactional. The previous contents of the named data set are replaced only after names, allele definitions, genotype matrices, linkage-map information, and marker statistics have all been validated and successfully constructed. If loading fails, the previous contents remain intact. Performance data are not installed by this function. The function is called for its effect and returns invisibly.

The package contains the compact tropical-maize example `v.tropmaize.vcf`. A non-running example is:

```
vignette.data <- new.env(parent = emptyenv())
data("v-tropmaize-vcf", package = "SelectionTools", envir = vignette.data)
st.load.vcf.data(vignette.data$v.tropmaize.vcf, data.set = "trop")
```

3.3 Export `SelectionTools` marker data to STvcf

`st.markerdata.to.STvcf()` converts marker genotypes and linkage map of an existing `SelectionTools` marker data set to the compact diploid STvcf representation. This is the reverse marker-data step. Together with the `st.get.simpop()` function it permits a simulation population to be returned first to the ordinary `SelectionTools` marker-data set and then converted to STvcf.

The marker data set must contain genotype data and a usable linkage map. Each marker-matrix column must have exactly one map location. Map-only locations are ignored. Missing or duplicate marker locations, inconsistent marker and map names, invalid chromosome numbers, non-finite positions, and negative positions cause conversion to fail.

The current marker order is retained. Map positions are returned as the current double-precision cM values without intentional rounding. Current numeric chromosome numbers are

converted to running STvcf codes 1, 2, ... in order of first appearance; `CHROM.levels` stores the corresponding numeric chromosome labels as character strings.

Current marker and individual names become the STvcf identifiers and must satisfy the usual uniqueness, alphanumeric, and fewer-than-256-byte rules. Names are never silently changed by this conversion.

There is one intentional naming consequence when the marker data originated from a simulation population. `st.get.simpop()` creates new marker-data individual identifiers rather than recovering names that may have existed before the population entered simulation. For a simulation population named P1, the resulting identifiers are P1I1, P1I2, and so on. The population name itself must contain only letters and digits, ensuring that the generated identifiers satisfy the STvcf name rules. Population names containing dots, punctuation, whitespace, or other non-alphanumeric characters are rejected. Reconstruction of the original pre-simulation individual names is not part of this transfer.

SelectionTools marker data store integer allele states but do not retain the historical textual VCF REF and ALT labels. Export therefore normalizes the observed non-missing states independently for every marker. The states are sorted by their current integer values. The first observed state becomes `REF="1"`; further states become `ALT="2, 3, ..."`; and genotypes are written as VCF indices 0, 1, 2, ... in that order.

For marker data that were originally loaded from STvcf, the internal alleles are already 1 for REF, 2 for ALT1, 3 for ALT2, and so forth. Transfer to and from simulation preserves these integer states. For marker data created by other routes, all distinct observed genotype states are retained, but their historical integer labels are normalized during STvcf export.

An all-missing marker is retained as `REF="1"`, `ALT="."`, with all genotypes missing. A marker with exactly one observed non-missing state is also written with `ALT="."`; that state uses VCF allele index 0. If either homologue of a SelectionTools genotype is missing, the exported STvcf genotype is completely missing, consistent with the STvcf partial-missing convention.

The current two marker-data allele arrays are retained in order as `allele1` and `allele2`. For data obtained through `st.get.simpop()` they correspond to the two simulation homologues. No new phasing calculation is performed.

Raw STvcf matrices are used when the normalized allele indices do not exceed 254, with raw value 255 denoting missing data. Otherwise integer matrices and `NA_integer_` are used. Marker allele values that cannot be represented by the simulation allele type, and marker state sets exceeding the existing 4096-state marker-statistics capacity, are rejected.

The linkage-map class is not part of STvcf. Exact map-position ties may already have been dispersed by the SelectionTools linkage-map or simulation machinery. This function returns the positions currently stored internally and cannot reconstruct an earlier duplicate position that is no longer represented.

Only marker genotype and genetic-map information are returned. Performance data, estimated effects, genomic relationship matrices, selection values, simulation GValue/PValue arrays, and other derived objects are not part of STvcf. On success the result is a list of class STvcf containing normalized VCF-style allele indices, the current marker and individual names, the current double-precision cM positions, and the chromosome-code mapping. If conversion fails, no STvcf object is returned.

A conversion example is:

```
## Starting from the data set installed in the preceding example:
x2 <- st.markerdata.to.STvcf(data.set = "trop")
df2 <- st.STvcf.to.dataframe(x2)

## A simulation population can use the same reverse chain:
st.get.simpop("P1", data.set = "fromsim")
sim.vcf <- st.markerdata.to.STvcf(data.set = "fromsim")
```

3.4 Convert STvcf to a VCF-like data frame

`st.STvcf.to.dataframe(STvcf)` expands a compact STvcf list to a wide VCF-like R data frame with one marker per row and one genotype column per individual. The object is validated before expansion. It must contain exactly the nine documented components, each exactly once. Extra, missing, duplicated, unnamed, or unknown components are rejected. Component types and dimensions must agree, chromosome codes must be consecutive and used, genetic positions must be finite and nonnegative, allele indices must agree with the REF/ALT definitions, and marker and individual names must satisfy the same rules as `st.load.vcf.data()`.

Marker and individual identifiers are unique, shorter than 256 bytes, and contain only ASCII letters and digits. REF may not be missing, empty, `.`, or comma-separated. ALT may be `.` or a comma-separated list; its entries must be nonempty, unique, different from REF, and not `..`. Malformed or duplicated allele labels, including an empty trailing ALT entry, are rejected.

The returned data frame contains CHROM, POS, ID, REF, ALT, and FORMAT, followed by one sample column for every entry in `STvcf$individual`. FORMAT is written as GT for every marker. Including the FORMAT column is deliberate because it makes the sample boundary unambiguous even if an individual itself is named POS, INFO, CM, or FORMAT.

CHROM is expanded using `CHROM.levels`. POS is the current STvcf double-precision genetic position in cM and is written without intentional rounding.

Genotypes are written with VCF allele indices: 0 for REF and 1, 2, ... for ALT alleles. Allele order is retained. Every non-missing genotype is written with a slash, for example `0/1`; a vertical bar is never written because STvcf does not retain the original separator and must not imply inferred phasing. If either stored allele is missing, the exported genotype is `./..`.

The two allele matrices must have the same storage type. Raw matrices use raw 255 for missing data and indices 0 through 254 for alleles. Integer matrices use `NA_integer_` for missing data. Allele indices are validated against REF/ALT, the simulation-transfer range, and the 4096-observed-state marker limit.

STvcf does not retain QUAL, FILTER, INFO, non-GT FORMAT fields, phase-set metadata, or a physical POS value that was superseded by CM, and these fields cannot be reconstructed. For an object produced with `st.dataframe.to.STvcf()`, expansion to a data frame and conversion back to STvcf preserve the STvcf information under the documented normalizations: GT separators are written as slash and missing genotypes are written as `./..`.

An example is:

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

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.