Simulations with OneMap

Cristiane Hayumi Taniguti - Statistical Genetics Lab - Department of Genetics - Luiz de Queiroz College of Agriculture - University of São Paulo

OneMap 3.0 also makes interface with PedigreeSim (link 12) software to perform simulations. There are three different ways to simulate maps in OneMap:

- Converting PedigreeSim output to OneMap raw data: function `pedsim2raw`

This function just converts the PedigreeSim files to OneMap raw data. The only source of errors is missing data controlled by `miss.perc` argument.

- Converting PedigreeSim output to VCF file simulating allele depth with different distributions: function `pedsim2vcf`

Simulating the VCF you can include other sources of errors. The map is simulated in PedigreeSim, and according to each genotype, the chosen statistical distribution will simulate the reference and alternative alleles counts. The genotypes can be reestimated according with the alleles counts simulated using the genotype callers Updog (function `updog_genotype`), polyRAD (function `polyRAD_genotype`) and Supermassa (function `supermassa_genotype` of `genotyping4onemap` package(link 1).

- Use a chromosome of a reference genome to simulate a bi-parental cross and RADseq Illumina reads for each individual: workflow `SimulateReads.wdl`

The workflow `SimulatedReads` is available in onemap_workflows (link 2) repository in Github. It performs the simulation of RADseq Illumina reads for outcrossing population, the SNP calling is made in Freebayes (link 3) and HaplotypeCaller (link 4) software, the genotype calling in updog (link 5), SuperMASSA (link 6) and polyRAD (link 7), and build genetics maps using every combination in OneMap and GUSMap (link 8). The workflow is wroten in Workflow Description Language (WDL) (link 9) and Cromwell workflow management system (link 10). The parameters of every software implemented can be easily changed. All the results can be evaluated in a shiny app (link 11).

Run PedigreeSim

First, download PedigreeSim java file (link 12). It will require java installed. You can run PedigreeSim directly and just use the output files in OneMap or you can use a function named `run_pedsim` that facilitates this procedure.

The function does not provide every possibility offered by PedigreeSim software. If you want to change any parameter that is not available in the function, please use directly the PedigreeSim software. Here is the software documentation (link 13) for more information.

```
# For outcrossing population
run_pedsim(chromosome = "Chr1", n.marker = 54, tot.size.cm = 100, centromere = 50,
           n.ind = 200, mk.types = c("A1", "A2", "A3", "A4", "B1.5", "B2.6", "B3.7",
                                     "C.8", "D1.9", "D1.10", "D1.11", "D1.12", "D1.13",
                                     "D2.14", "D2.15", "D2.16", "D2.17", "D2.18"),
           n.types = rep(3,18), pop = "F1",
           path.pedsim = "/home/cristiane/Programs/PedigreeSim/",
           name.mapfile = "mapfile.map", name.founderfile="founderfile.gen",
           name.chromfile="sim.chrom", name.parfile="sim.par",
           name.out="sim_out")
```

```
# For F2 population
run_pedsim(chromosome = c("Chr1", "Chr2"), n.marker = c(75, 75),
           tot.size.cm = c(100,100), centromere = c(50,50),
           n.ind = 200, mk.types = c("A.H.B","C.A", "D.B"),
           n.types = rep(50,3), pop = "F2",
           path.pedsim = "/home/cristiane/Programs/PedigreeSim/",
           name.mapfile = "mapfile_f2.map", name.founderfile="founderfile_f2.gen",
           name.chromfile="sim_f2.chrom", name.parfile="sim_f2.par",
           name.out="sim_f2")
```

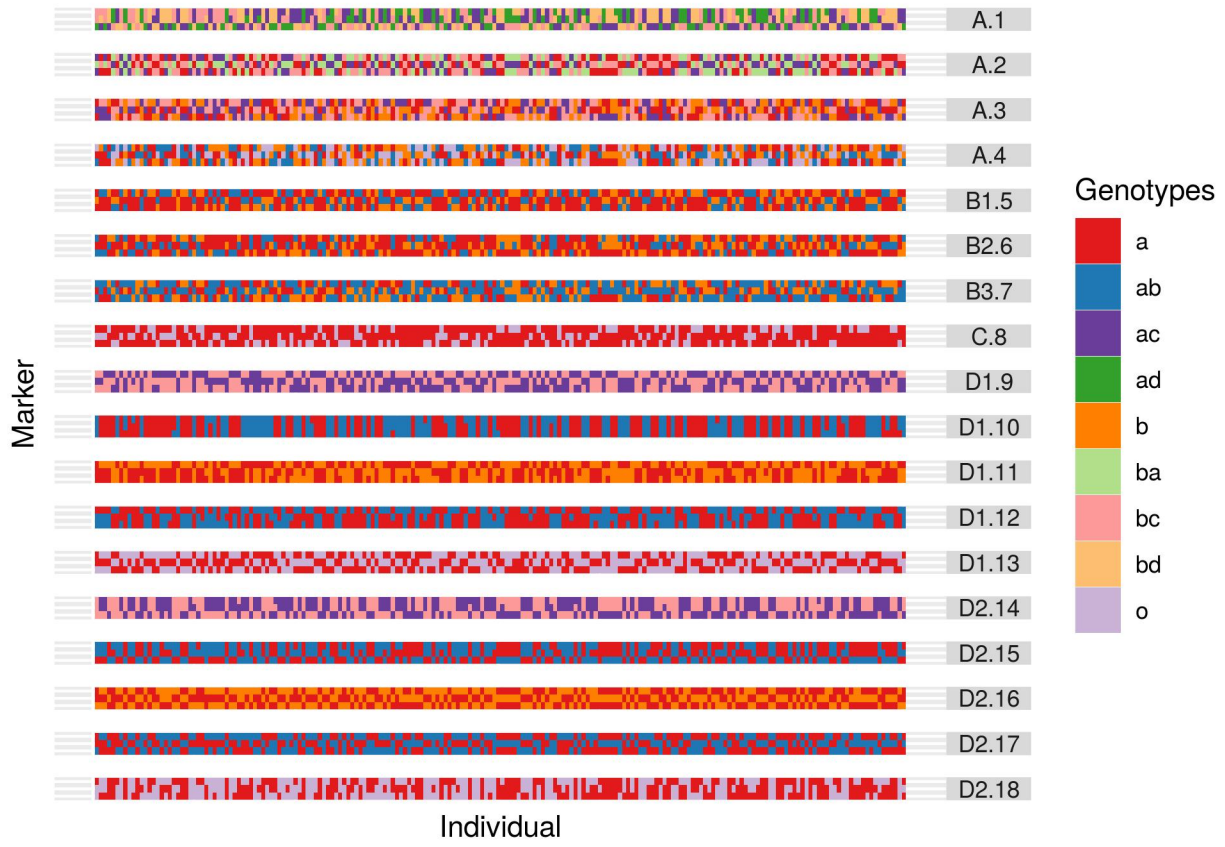
The function allows us to create f2 intercross and backcross populations from bi-parental cross of inbred lines and segregating F1 population from bi-parental cross of heterozygous parents. You can define it in `pop` argument. You must change the `path.pedsim` to the path where the PedigreeSim.jar is stored in your system. You can also define the number of chromosomes (argument `chromosome`), the number of markers in each chromosome (`n.marker`), the total size of the groups in cM (`tot.size.cm`), the position of the centromere (`centromere`), number of individuals in the population (`n.ind`), the marker types (`mk.types`, see the table in session `Creating the data file` of Outcrossing populations vignette (link 14)) and the number of markers of each type (`n.types`).

We suggest you open the output files `founderfile`, `chromfile`, `mapfile` and `parfile` to check if they agree with your intentions before proceeding to other analyses.

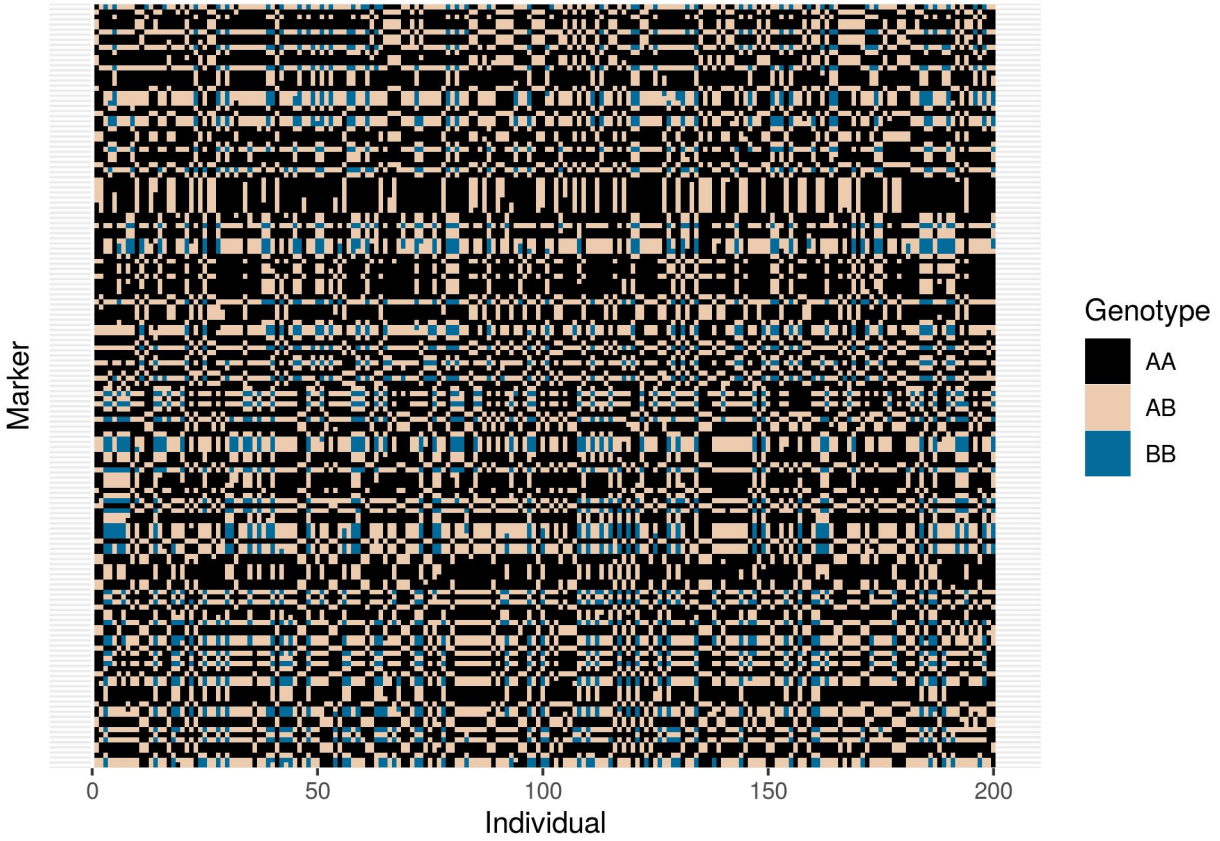
Simulate OneMap raw data

Once you run PedigreeSim, you should have the output file `genotypes.dat`. To convert this to OneMap raw file, you just need to specify the cross-type (`cross`), which ones are the parents (`parent1` and `parent2`), and if you want to include missing genotypes (`miss.perc`). Only cross types `outcross` and `f2 intercross` are supported by now.

```
# For outcrossing population
pedsim2raw(genofile = "sim_out_genotypes.dat", cross="outcross",
           parent1 = "P1", parent2 = "P2", out.file = "sim_out.example.raw", miss.perc = 0)

onemap.obj <- read_onemap("sim_out.example.raw")

p <- plot(onemap.obj, all = F)
ggsave(p, filename = "plot_onemap1.jpeg")
```




```
# For F2 population
pedsim2raw(genofile = "sim_f2_genotypes.dat", cross="f2 intercross",
           parent1 = "P1", parent2 = "P2", out.file = "sim_f2.example.raw", miss.perc = 0)

onemap.obj <- read_onemap("sim_f2.example.raw")

p <- plot(onemap.obj, all = F)
ggsave(p, filename = "plot_onemap2.jpeg")
```

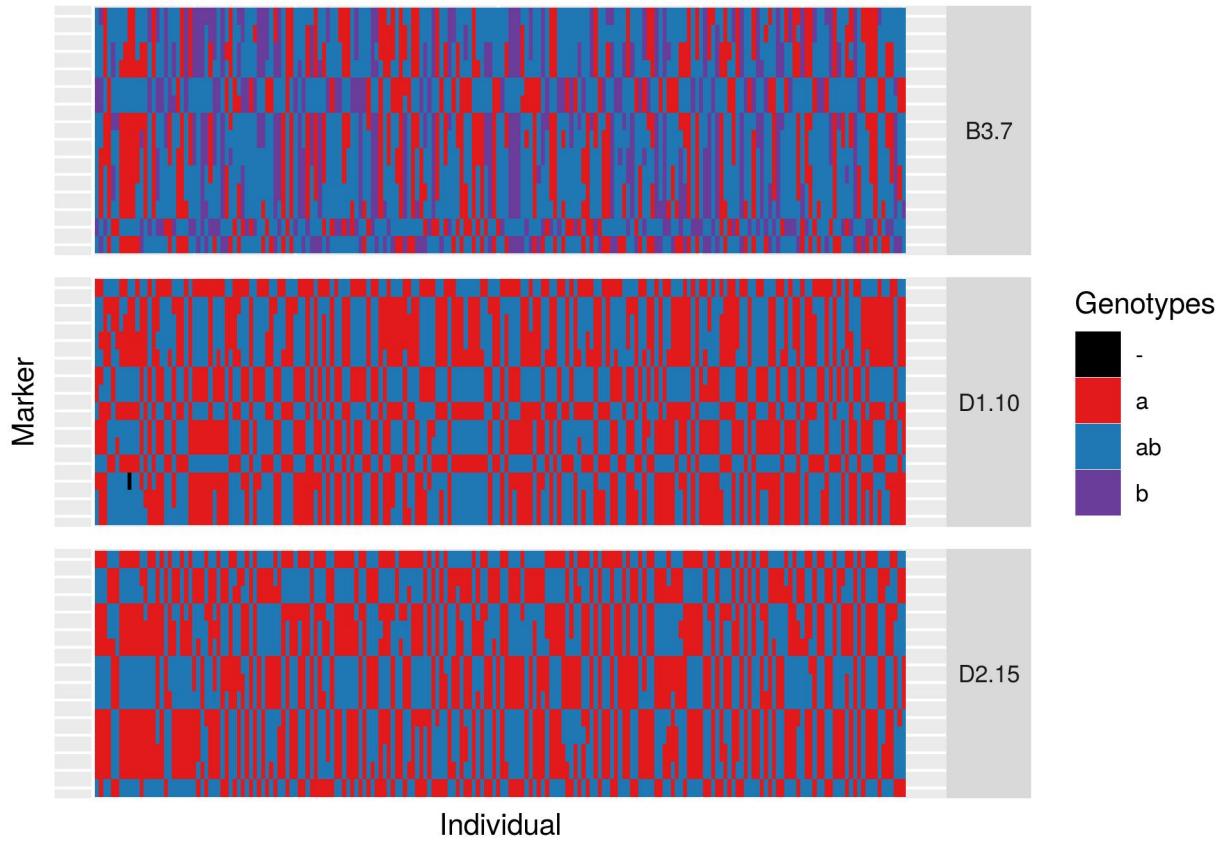


Simulate VCF file

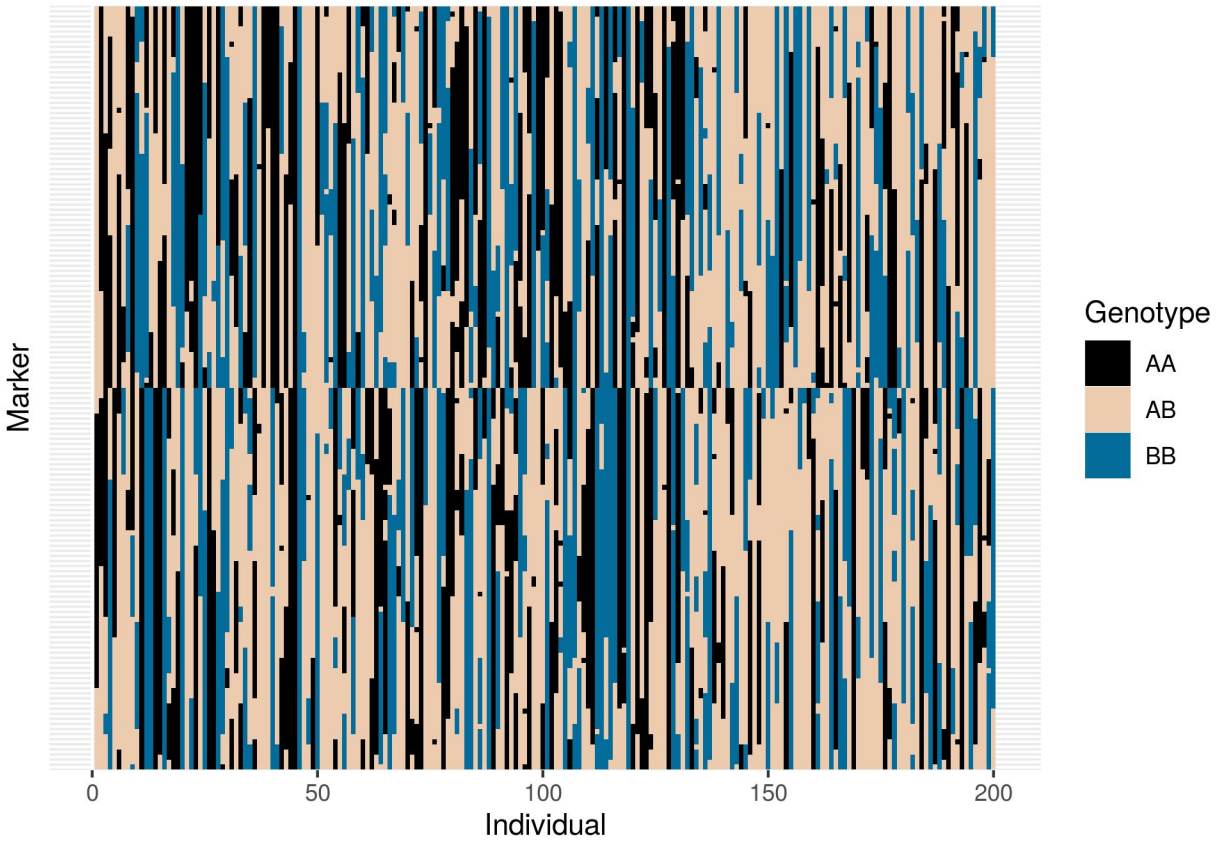
The same output file from PedigreeSim, the `genotypes.dat` can be used to simulate a VCF file together with the PedigreeSim `mapfile` and `chrom`. The advantage to simulate a VCF instead of OneMap raw file is that VCF is a standard file format and can store a lot of other information including the allele counts, usually in the field `AD` or `DPR`. The `pedsim2vcf` function can simulate the allele counts using negative binomial or updog distributions (argument `method`). The main parameters for the distributions are defined with arguments `mean.depth`, that defines the mean allele depth in the progeny, `p.mean.depth` that defines the mean allele depth in the parents, argument `disper.par` defines the dispersion parameter, `mean.phred` defines the mean Phred score of the sequencing technology used. The function can also simulate missing data (`miss.perc`). Through argument `pos` and `chr` you can define vectors with physical position and chromosome of each marker. With argument `haplo.ref` you define which one of the haplotypes in `genotypes.dat` will the considered the reference. Establishing `phase` as TRUE, the VCF will have phased genotypes. After allele counts are simulated, the genotypes are re-estimated using a binomial distribution. The VCF generated by this function only has one or two FORMAT fields, the GT and AD (if `counts = TRUE`). **Dominant markers are not supported by this function**.

```
run_pedsim(chromosome = "Chr1", n.marker = 42, tot.size.cm = 100, centromere = 50,
           n.ind = 200, mk.types = c("B3.7", "D1.10", "D2.15"),
           n.types = rep(14,3), pop = "F1",
           path.pedsim = "/home/cristiane/Programs/PedigreeSim/",
           name.mapfile = "mapfile.map", name.founderfile="founderfile.gen",
           name.chromfile="sim.chrom", name.parfile="sim.par",
           name.out="sim_out")
```

```r
# For outcrossing population
pedsim2vcf(inputfile = "sim_out_genotypes.dat",
           map.file = "mapfile.map",
           chrom.file = "sim.chrom",
           out.file = "simu_out.vcf",
           miss.perc = 0,
           counts = TRUE,
           mean.depth = 100,
           p.mean.depth = 100,
           chr.mb = 10,
           method = "updog",
           mean.phred = 20,
           bias=1,
           od=0.00001,
           pos=NULL,
           chr=NULL,
           phase = FALSE,
           disper.par=2)

library(vcfR)
vcfR_out.obj <- read.vcfR("simu_out.vcf")
onemap_out.obj <- onemap_read_vcfR(vcfR.object = vcfR_out.obj,
                                   cross = "outcross", parent1 = "P1", parent2 = "P2")

p <- plot(onemap_out.obj, all=F)
ggsave(p, filename = "plot_onemap3.jpeg")
```



```r
# For F2 population
run_pedsim(chromosome = c("Chr1", "Chr2"), n.marker = c(75, 75),
           tot.size.cm = c(100,100), centromere = c(50,50),
           n.ind = 200, mk.types = c("A.H.B"),
           n.types = 150, pop = "F2",
           path.pedsim = "/home/cristiane/Programs/PedigreeSim/",
           name.mapfile = "mapfile_f2.map", name.founderfile="founderfile_f2.gen",
           name.chromfile="sim_f2.chrom", name.parfile="sim_f2.par",
           name.out="sim_f2")

pedsim2vcf(inputfile = "sim_f2_genotypes.dat",
           map.file = "mapfile_f2.map",
           chrom.file = "sim_f2.chrom",
           out.file = "simu_f2.vcf",
           miss.perc = 0,
           counts = TRUE,
           mean.depth = 100,
           p.mean.depth = 100,
           chr.mb = 10,
           method = "updog",
           mean.phred = 20,
           bias=1,
           od=0.001,
           pos=NULL,
           chr=NULL,
           phase = FALSE,
           disper.par=2)

vcfR_f2.obj <- read.vcfR("simu_f2.vcf")
onemap_f2.obj <- onemap_read_vcfR(vcfR.object = vcfR_f2.obj,
                                  cross = "f2 intercross", parent1 = "P1",
                                  parent2 = "P2", f1 = "F1")

p <- plot(onemap_f2.obj, all=F)
ggsave(p, filename = "plot_onemap4.jpeg")
```

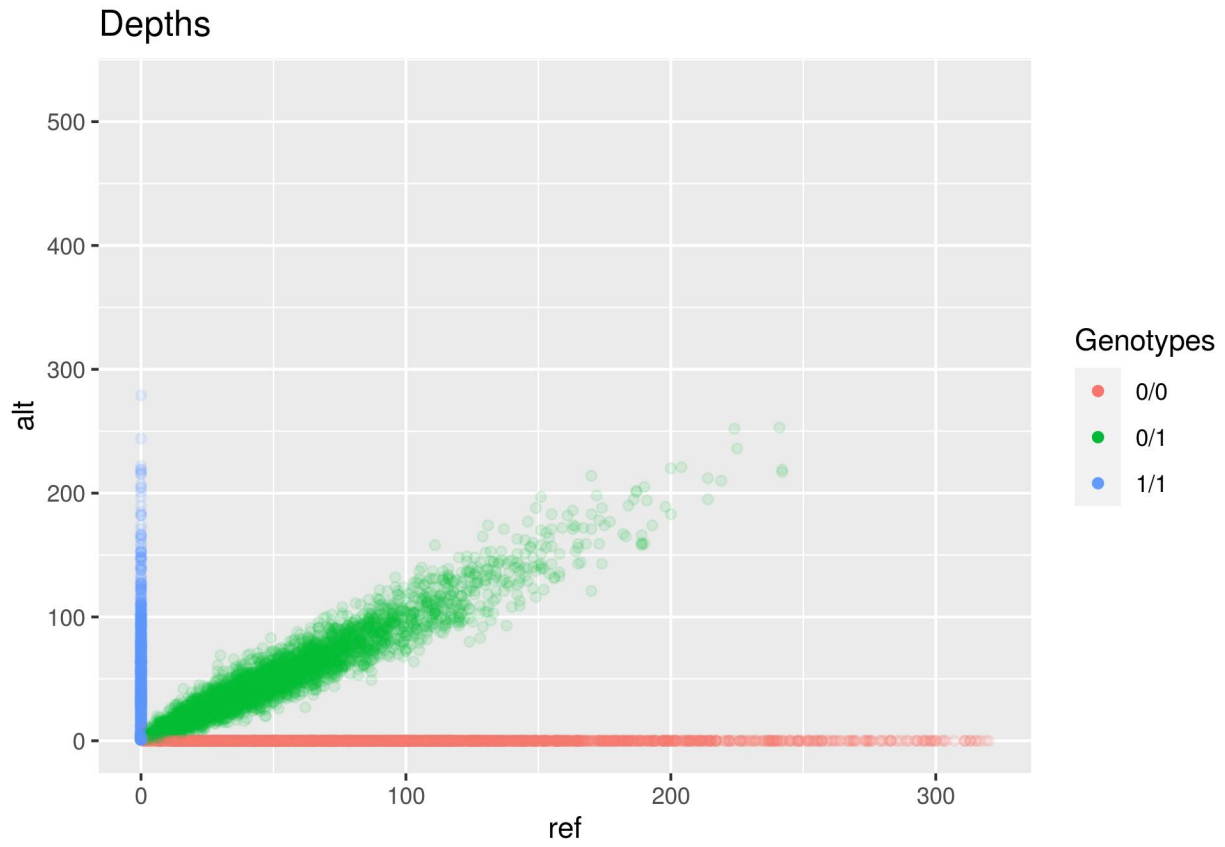


Graphical view of genotypes and allele counts

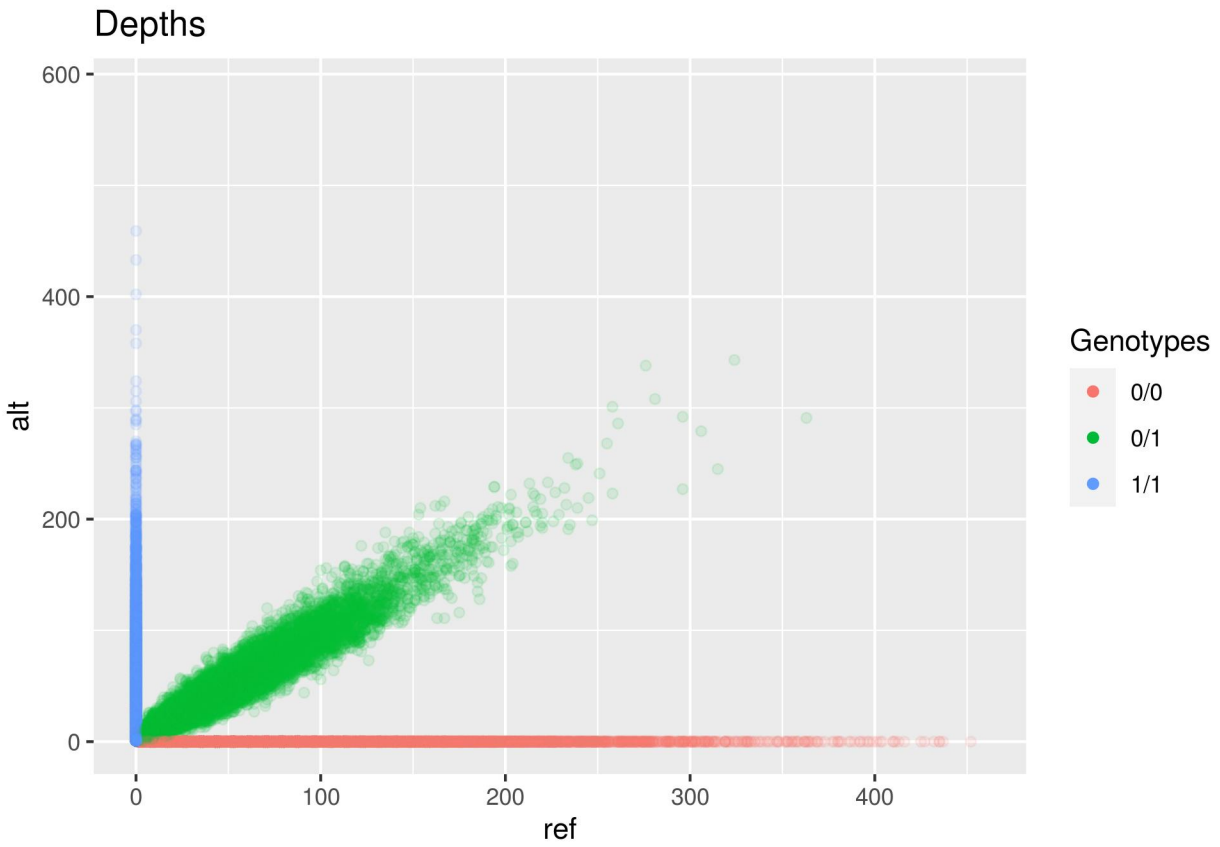
Function `create_depth_profile` generates dispersion graphics with x and y-axis pointing the reference and alternative allele counts, respectively. The function is only available for biallelic markers and for outcrossing and f2 intercross population. Each dot represents a genotype considering `mks` markers and `inds` individuals. If are established NULL for both arguments, all markers and individuals are considered. The color of the dots are according with the genotypes present in OneMap object (`GTfrom = onemap`) or in VCF file (`GTfrom = vcf`) or the color can represent the error rate (1 - highest genotype probability) of each genotype in OneMap object (`GTfrom = prob`). An rds file is generated with the data in the graphic (`rds.file`). The `alpha` argument controls the transparency of color of each dot, regulate this parameter is a good idea when having a big amount of markers and individuals.

```
# For outcrossing population
p <- create_depths_profile(onemap.obj = onemap_out.obj,
                           vcfR.object = vcfR_out.obj,
                           parent1 = "P1",
                           parent2 = "P2",
                           vcf.par = "AD",
                           recovering = FALSE,
                           mks = NULL,
                           inds = NULL,
                           GTfrom = "vcf",
                           alpha = 0.1,
                           rds.file = "depths_out.rds")
```

```
ggsave(p, filename = "depth_prof1.jpeg")
```




```
# For f2 intercross population
p <- create_depths_profile(onemap.obj = onemap_f2.obj,
                           vcfR.object = vcfR_f2.obj,
                           parent1 = "P1",
                           parent2 = "P2",
                           f1 = "F1",
                           vcf.par = "AD",
                           recovering = FALSE,
                           mks = NULL,
                           inds = NULL,
                           GTfrom = "vcf",
                           alpha = 0.1,
                           rds.file = "depths_f2.rds")

ggsave(p, filename = "depth_prof2.jpeg")
```



If you choose to simulate allele counts in `pedsim2vcf` it is also possible to reestimate the genotypes using these counts. Doing this, you will include errors in your data, coming from random sampling, if considering only Poisson or negative binomial distributions, and/or dispersion, outliers and bias errors, if using updog model. Another advantage is that reestimating genotypes with any of these software, you can obtain genotypes probabilities to be used in the map building instead of only genotypes.

warning: The re-estimation of genotypes is only performed in biallelic codominant markers.

Updog (link 5), polyRAD (link 7), and SuperMASSA (link 6) are software designed for genotyping polyploid species, which involve a more complex procedure compared with diploid species. These software consider not only the proportion of alleles to define the genotypes but other aspects as the expected distribution in the progeny according to parent's genotypes. See more about them in their manuals.

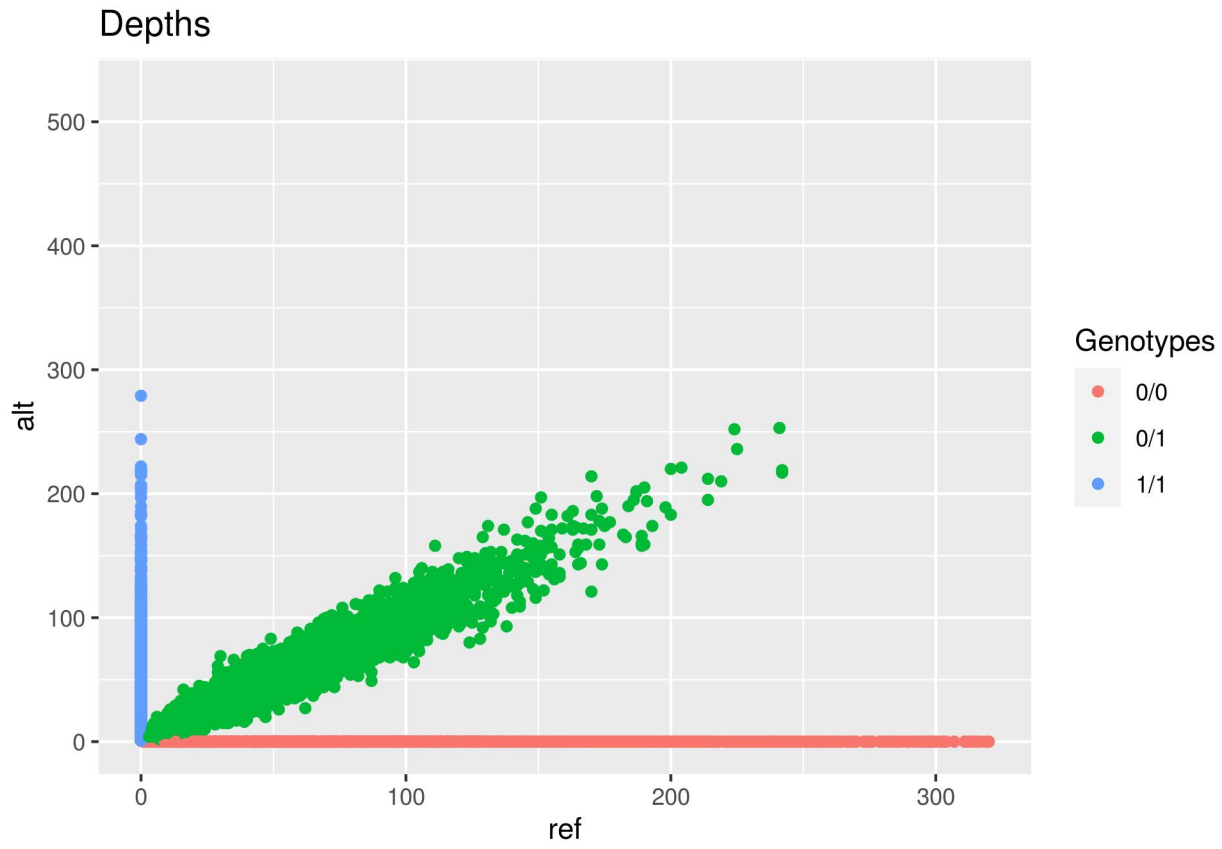
Genotypes probabilities usage

OneMap 3.0 has three options to define error probability in the HMM emission phase. The default method is to consider a global error rate for every genotype 10^{-5}. Until this present version, only this procedure was implemented. Now, with the function `create_probs`, we offer the option to change the global error rate (`global_error` argument) or consider an error rate by genotype (`genotypes_errors`) or a genotype probability for each genotype (`genotypes_probs`).

With updog

The function `updog_genotype` make the interface of OneMap with Updog to perform the genotype calling.

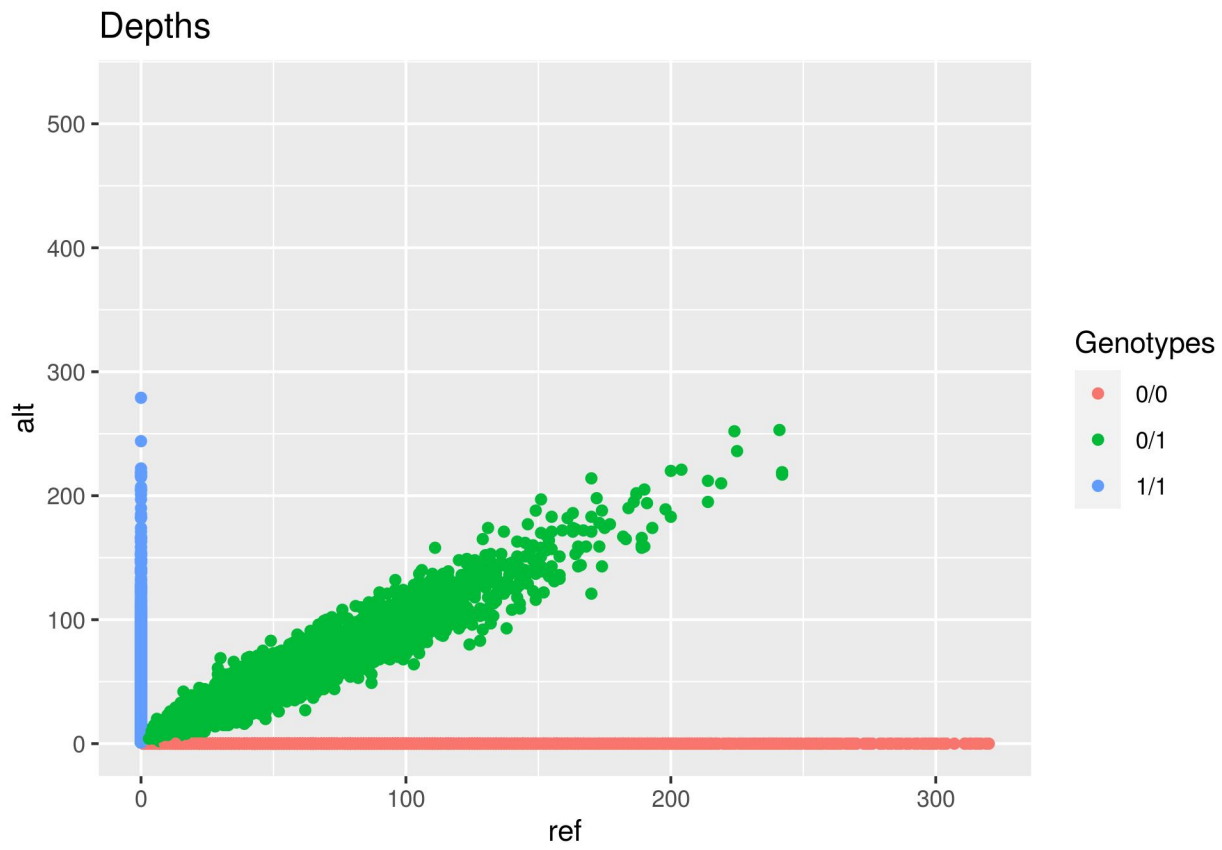
```
onemap_geno.updog <- updog_genotype(vcfR.object=vcfR_out.obj,
                                    onemap.object= onemap_out.obj,
                                    vcf.par = "AD",
                                    parent1="P1",
                                    parent2="P2",
                                    f1=NULL,
                                    recovering = FALSE,
                                    mean_phred = 20,
                                    cores = 4,
                                    depths = NULL,
                                    global_error = NULL,
                                    use_genotypes_errors = TRUE,
                                    use_genotypes_probs = FALSE)

p <- create_depths_profile(onemap.obj = onemap_geno.updog,
                           vcfR.object = vcfR_out.obj,
                           parent1 = "P1",
                           parent2 = "P2",
                           vcf.par = "AD",
                           recovering = FALSE,
                           GTfrom = "vcf")

ggsave(p, filename = "depth_prof3.jpeg")
```



With polyRAD

The function `polyRAD_genotype` make the interface of OneMap with polyRAD to perform the genotype calling.

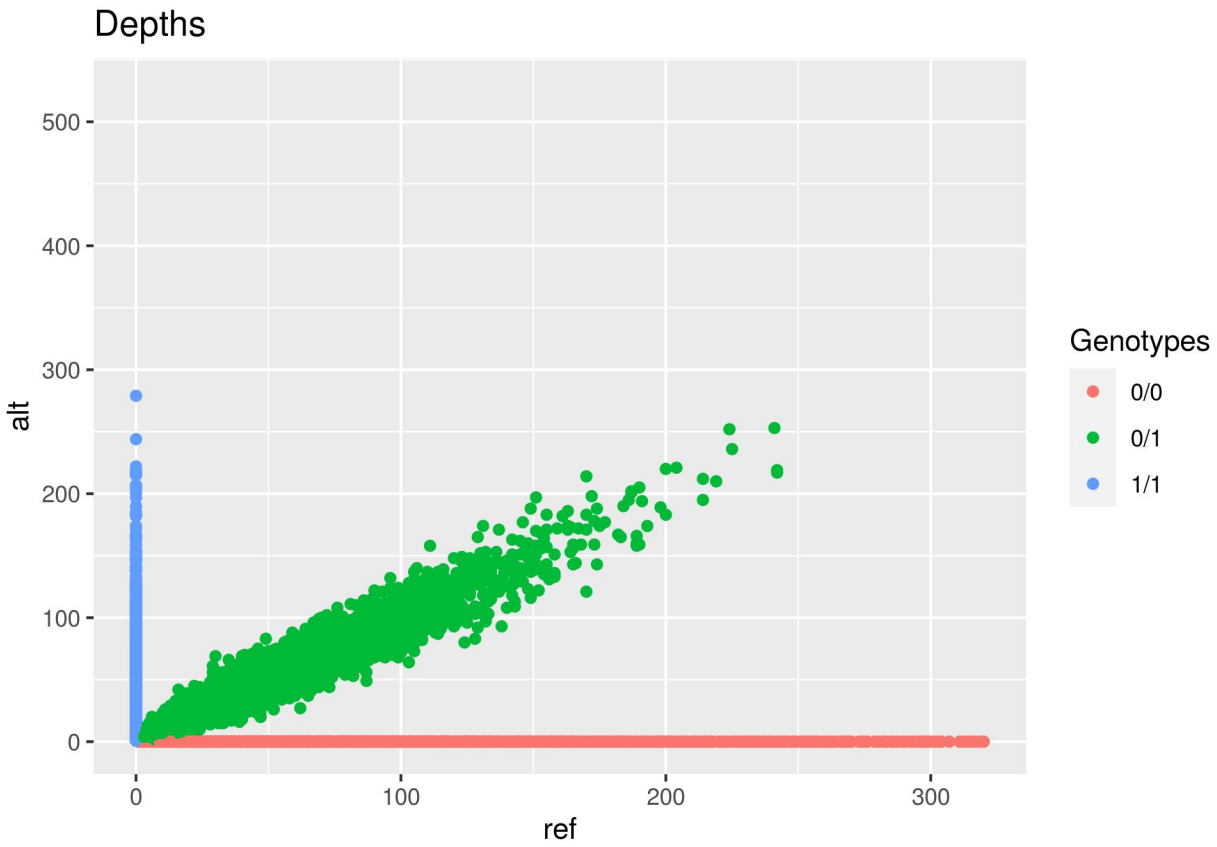
```
onemap_geno.polyRAD <- polyRAD_genotype(vcf="simu_out.vcf",
                                        onemap.obj = onemap_out.obj,
                                        parent1="P1",
                                        parent2="P2",
                                        f1="F1",
                                        crosstype="outcross",
                                        global_error = NULL,
                                        use_genotypes_errors = TRUE,
                                        use_genotypes_probs = FALSE,
                                        rm_multiallelic = F)


p <- create_depths_profile(onemap.obj = onemap_geno.polyRAD,
                           vcfR.object = vcfR_out.obj,
                           parent1 = "P1",
                           parent2 = "P2",
                           vcf.par = "AD",
                           recovering = FALSE,
                           GTfrom = "vcf")
ggsave(p, filename = "depth_prof4.jpeg")
```



With SuperMASSA

The function `supermassa_genotype` make the interface of OneMap with SuperMASSA to perform the genotype calling.

```
onemap_geno.supermassa <- supermassa_genotype(vcfR.object=vcfR_out.obj,
                                              onemap.object= onemap_out.obj,
                                              vcf.par = "AD",
                                              parent1="P1",
                                              parent2="P2",
                                              f1=NULL,
                                              recovering = FALSE,
                                              mean_phred = 20,
                                              cores = 4,
                                              depths = NULL,
                                              global_error = NULL,
                                              use_genotypes_errors = FALSE,
                                              use_genotypes_probs = TRUE,
                                              rm_multiallelic = F)

p <- create_depths_profile(onemap.obj = onemap_geno.supermassa,
                           vcfR.object = vcfR_out.obj,
                           parent1 = "P1",
                           parent2 = "P2",
                           vcf.par = "AD",
                           recovering = FALSE,
                           GTfrom = "vcf")

ggsave(p, filename = "depth_prof5.jpeg")
```



Example

Here we want to compare the best map using updog counts simulations and updog, polyRAD and SuperMASSA genotyping and genotypes probabilities. As a simple example, we will simulate only one family and one chromosome. Simulation scenarios:

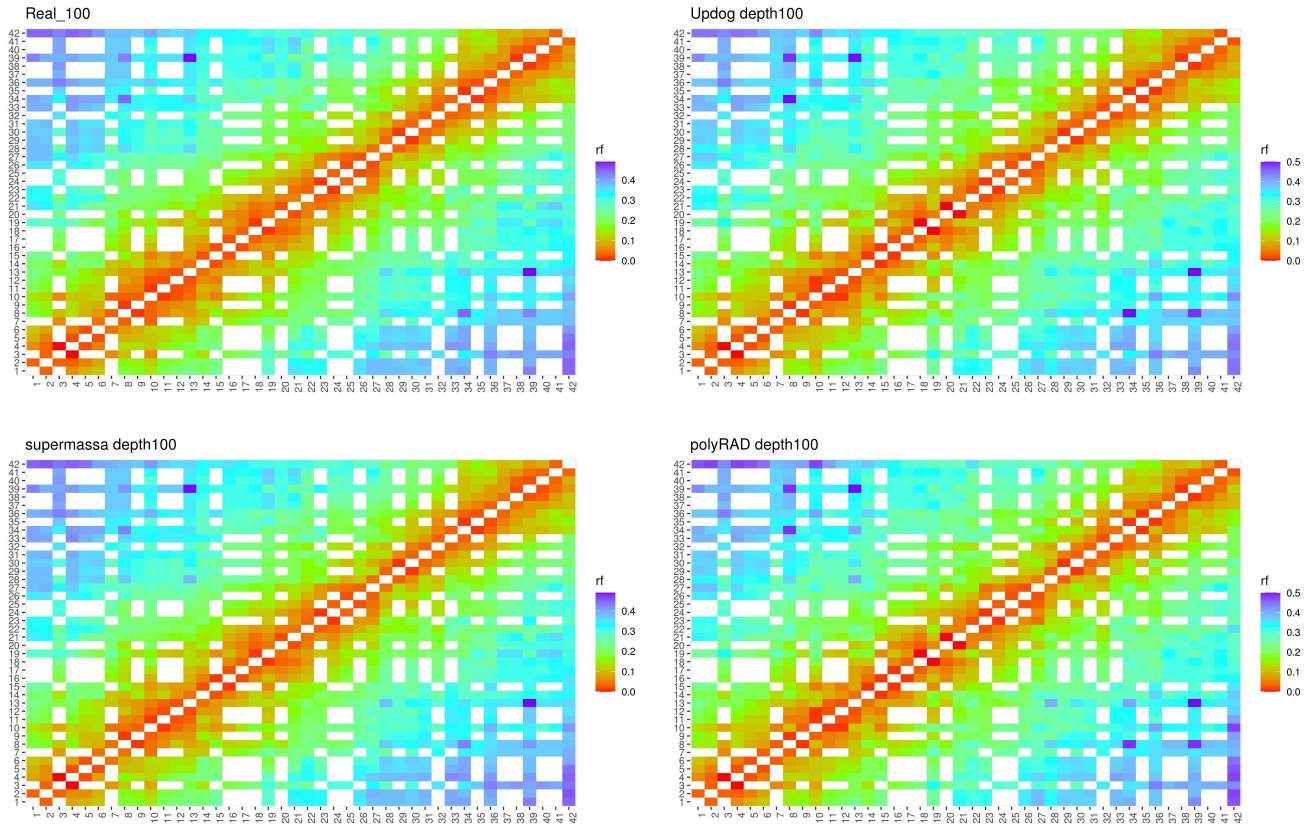
- Mean depth: 10, 50, and 100;
- Genetic map size: 100 cM
- Number of markers: 42
- Marker types: 14 B3.7; 14 D1.10; 14 D2.15

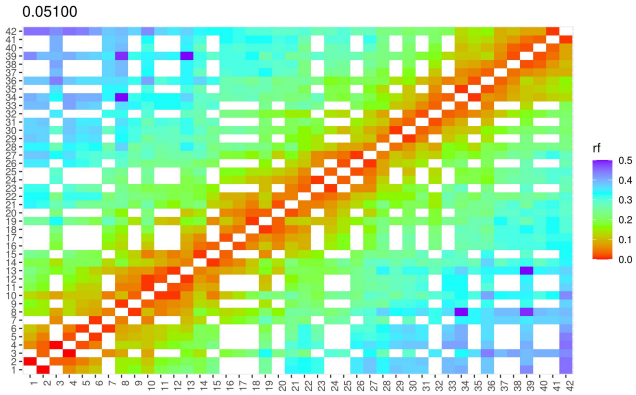
```
for(depth in c(100, 50, 10)){
  run_pedsim(chromosome = "Chr1", n.marker = 42,
             tot.size.cm = 100, centromere = 50,
             n.ind = 200, mk.types = c("B3.7","D1.10", "D2.15"),
             n.types = rep(14,3), pop = "F1",
             path.pedsim = "/home/cristiane/Programs/PedigreeSim/",
             name.mapfile = "mapfile.map", name.founderfile="founderfile.gen",
             name.chromfile="sim.chrom", name.parfile="sim.par",
             name.out="sim_out")

  # VCF with unmodified genotypes
  pedsim2vcf(inputfile = "sim_out_genotypes.dat",
             map.file = "mapfile.map",
             chrom.file = "sim.chrom",
```

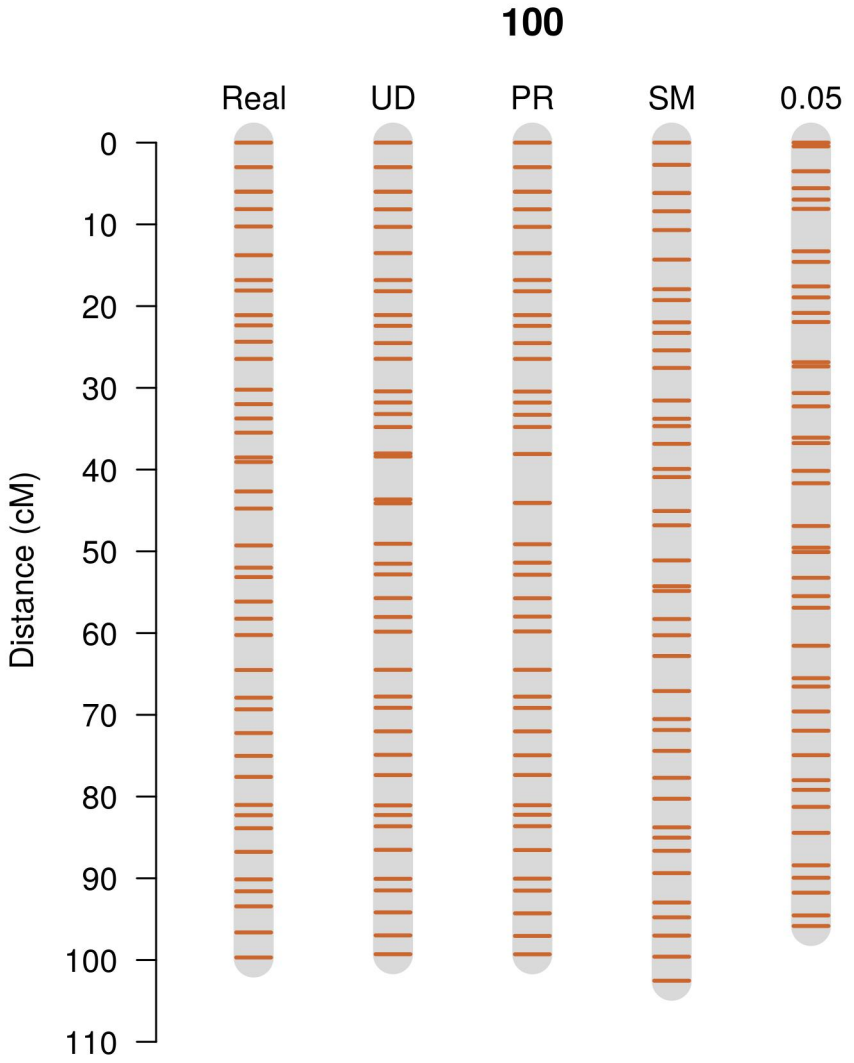
```
            out.file = "simu_out.vcf",
            miss.perc = 0,
            counts = FALSE)

vcfR_out.obj <- read.vcfR("simu_out.vcf")
onemap_out.obj <- onemap_read_vcfR(vcfR.object = vcfR_out.obj,
                                   cross = "outcross", parent1 = "P1", parent2 = "P2")

twopts <- rf_2pts(onemap_out.obj)
seq_ord <- make_seq(twopts, order(as.numeric(onemap_out.obj$POS)))
map_real <- map(seq_ord, phase_cores = 4)
p <- rf_graph_table(map_real, main = paste0("Real_",depth))
ggsave(p, filename = paste0("real", depth,".jpeg"))

# Simulating counts
pedsim2vcf(inputfile = "sim_out_genotypes.dat",
            map.file = "mapfile.map",
            chrom.file = "sim.chrom",
            out.file = "simu_out.vcf",
            miss.perc = 0,
            counts = TRUE,
            mean.depth = depth,
            p.mean.depth = depth,
            chr.mb = 10,
            method = "updog",
            mean.phred = 20,
            bias=0.8,
            od=0.0001,
            pos=NULL,
            chr=NULL,
            phase = FALSE,
            disper.par=2)

vcfR_out.obj <- read.vcfR("simu_out.vcf")
onemap_out.obj <- onemap_read_vcfR(vcfR.object = vcfR_out.obj,
                                   cross = "outcross", parent1 = "P1", parent2 = "P2")

onemap_geno.updog <- updog_genotype(vcfR.object=vcfR_out.obj,
                                    onemap.object= onemap_out.obj,
                                    vcf.par = "AD",
                                    parent1="P1",
                                    parent2="P2",
                                    f1=NULL,
                                    recovering = TRUE,
                                    mean_phred = 20,
                                    cores = 4,
                                    depths = NULL,
                                    global_error = NULL,
                                    use_genotypes_errors = FALSE,
                                    use_genotypes_probs = TRUE,
                                    rm_multiallelic = F)

twopts <- rf_2pts(onemap_geno.updog)
```

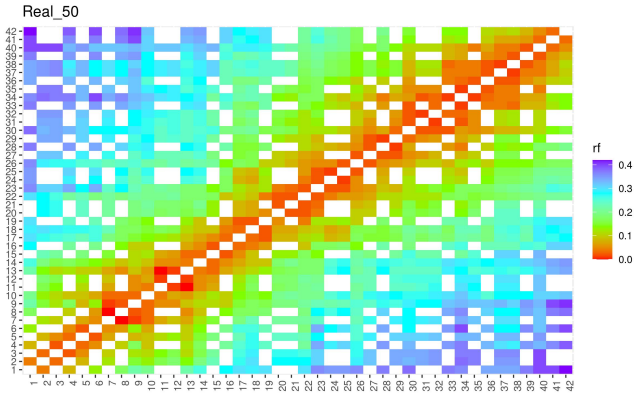
```r
seq_ord <- make_seq(twopts, order(as.numeric(onemap_geno.updog$POS)))
map_updog <- map(input.seq = seq_ord, phase_cores = 4, rm_unlinked = T)
# If HMM find problems between two markers, one of them will be automatically
# discarted and the sequence of markers without it will be returned
while(is(map_updog, "vector")){
  # if the result is a sequence of marker numbers, then HMM is run again
  # This will be repeteated until HMM can run with no problems
  seq_temp <- make_seq(twopts, map_updog)
  map_updog <- map(input.seq = seq_temp, phase_cores = 4, rm_unlinked = T)
}
p <- rf_graph_table(map_updog, main = paste0("Updog depth",depth))
ggsave(p, filename = paste0("updog", depth,".jpeg"))

onemap_geno.polyRAD <- polyRAD_genotype(vcf="simu_out.vcf",
                                        onemap.obj = onemap_out.obj,
                                        parent1="P1",
                                        parent2="P2",
                                        f1="F1",
                                        crosstype="outcross",
                                        global_error = NULL,
                                        use_genotypes_errors = FALSE,
                                        use_genotypes_probs = TRUE,
                                        rm_multiallelic = F)

twopts <- rf_2pts(onemap_geno.polyRAD)
seq_ord <- make_seq(twopts, order(as.numeric(onemap_geno.polyRAD$POS)))
map_polyRAD <- map(input.seq = seq_ord, phase_cores = 4, rm_unlinked = T)
while(is(map_polyRAD, "vector")){
  seq_temp <- make_seq(twopts, map_polyRAD)
  map_polyRAD <- map(input.seq = seq_temp, phase_cores = 4, rm_unlinked = T)
}
p <- rf_graph_table(map_polyRAD, main = paste0("polyRAD depth", depth))
ggsave(p, filename = paste0("polyrad", depth,".jpeg"))

onemap_geno.supermassa <- supermassa_genotype(vcfR.object=vcfR_out.obj,
                                              onemap.object= onemap_out.obj,
                                              vcf.par = "AD",
                                              parent1="P1",
                                              parent2="P2",
                                              f1=NULL,
                                              recovering = FALSE,
                                              mean_phred = 20,
                                              cores = 4,
                                              depths = NULL,
                                              global_error = NULL,
                                              use_genotypes_errors = FALSE,
                                              use_genotypes_probs = TRUE,
                                              rm_multiallelic = F)

twopts <- rf_2pts(onemap_geno.supermassa)
seq_ord <- make_seq(twopts, order(as.numeric(onemap_geno.supermassa$POS)))
map_supermassa <- map(input.seq = seq_ord, phase_cores = 4, rm_unlinked = T)
while(is(map_supermassa, "vector")){
```

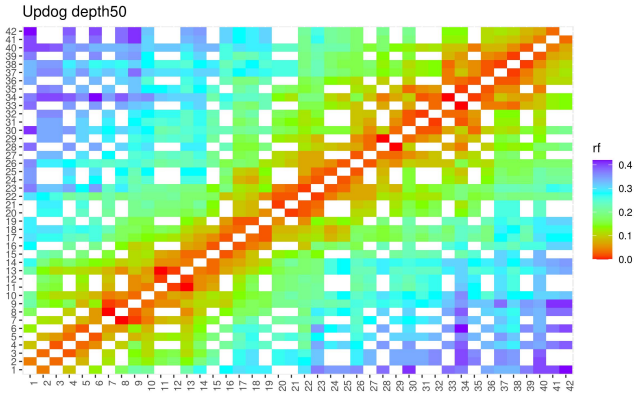
```
    seq_temp <- make_seq(twopts, map_supermassa)
    map_supermassa <- map(input.seq = seq_temp, phase_cores = 4, rm_unlinked = T)
  }
  p <- rf_graph_table(map_supermassa, main = paste0("supermassa depth", depth))
  ggsave(p, filename = paste0("supermassa", depth,".jpeg"))

  onemap_0.05 <- create_probs(onemap_geno.updog, global_error = 0.05)

  twopts <- rf_2pts(onemap_0.05)
  seq_ord <- make_seq(twopts, order(as.numeric(onemap_0.05$POS)))
  map_0.05 <- map(input.seq = seq_ord, phase_cores = 4, rm_unlinked = T)
  while(is(map_0.05, "vector")){
    seq_temp <- make_seq(twopts, map_0.05)
    map_0.05 <- map(input.seq = seq_temp, phase_cores = 4, rm_unlinked = T)
  }
  p <- rf_graph_table(map_0.05, main = paste0("0.05", depth))
  ggsave(p, filename = paste0("0.05", depth,".jpeg"))

  draw_map2(map_real, map_updog, map_polyRAD,
            map_supermassa, map_0.05, main = depth,
            group.names = c("Real", "UD", "PR", "SM", "0.05"),
            output = paste0(depth,"_maps.jpeg"))
}
```

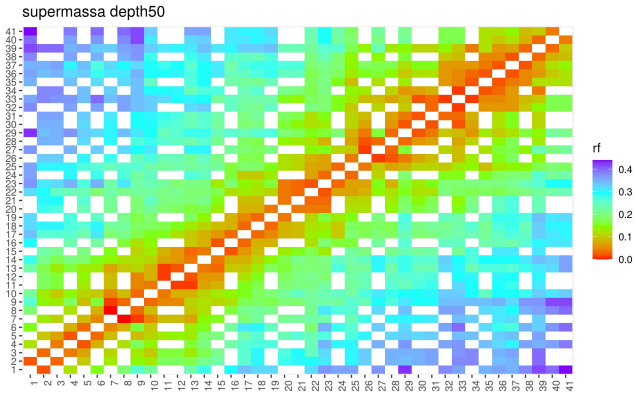


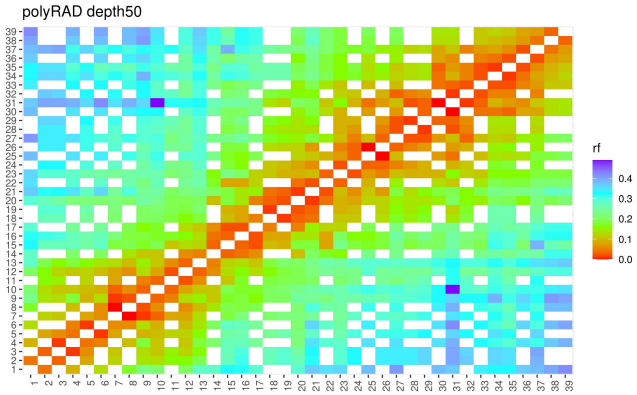

0.05100
rf
0.5
0.4
0.3
0.2
0.1
0.0

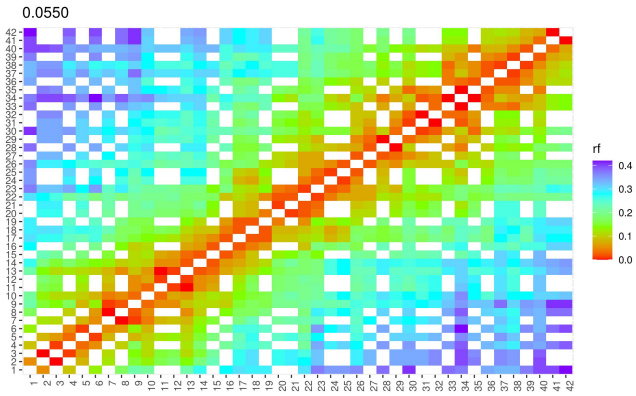

100
Real
UD
PR
SM
0.05
Distance (cM)
0
10
20
30
40
50
60
70
80
90
100
110

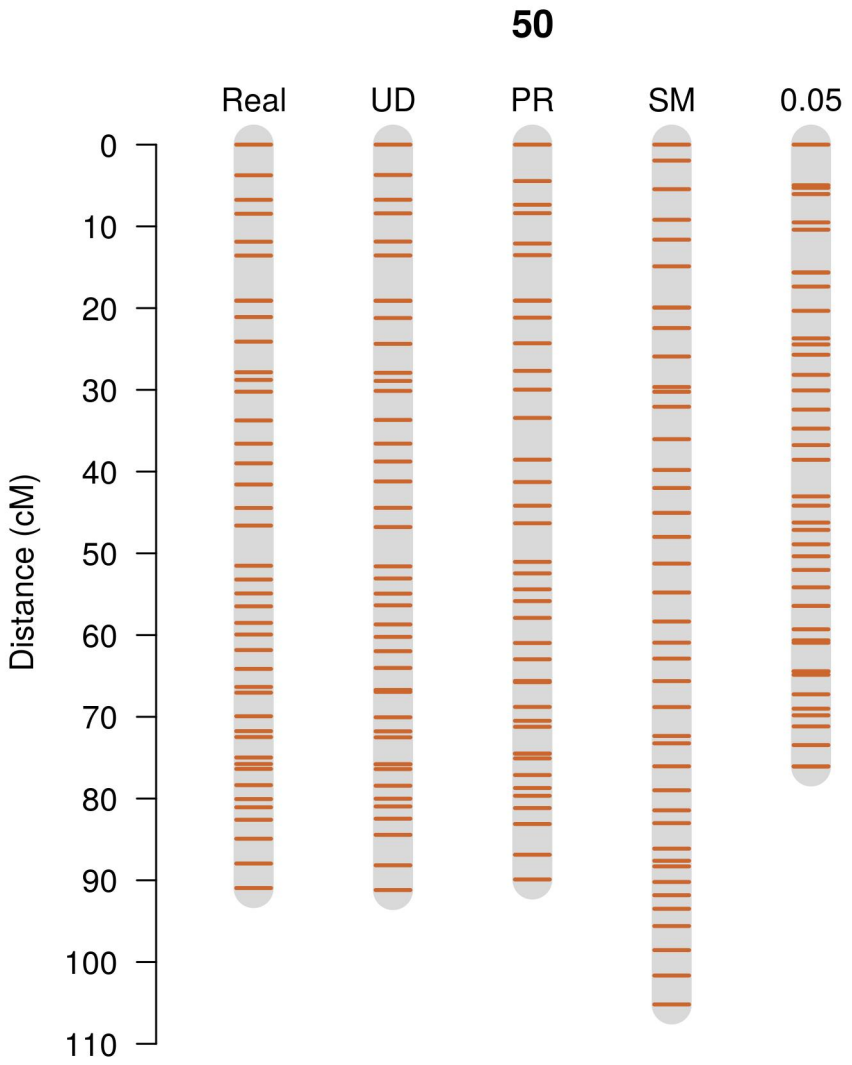

Real_50
rf
0.4
0.3
0.2
0.1
0.0

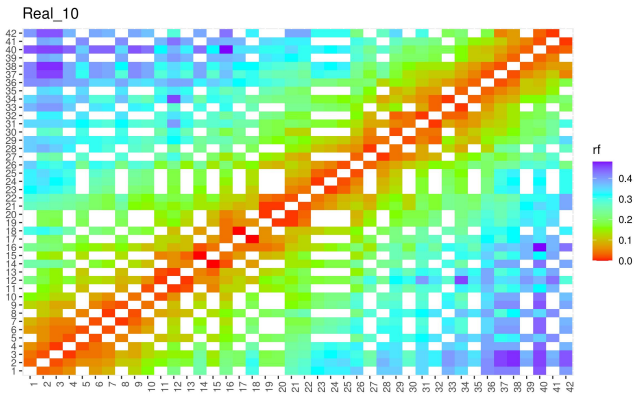

Updog depth50
rf
0.4
0.3
0.2
0.1
0.0

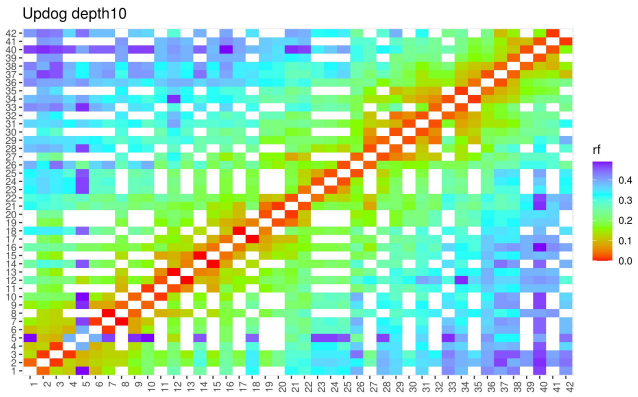

supermassa depth50
rf
0.4
0.3
0.2
0.1
0.0

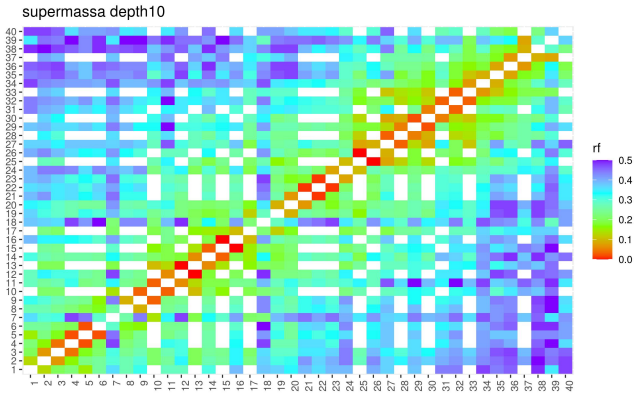

polyRAD depth50
rf
0.4
0.3
0.2
0.1
0.0

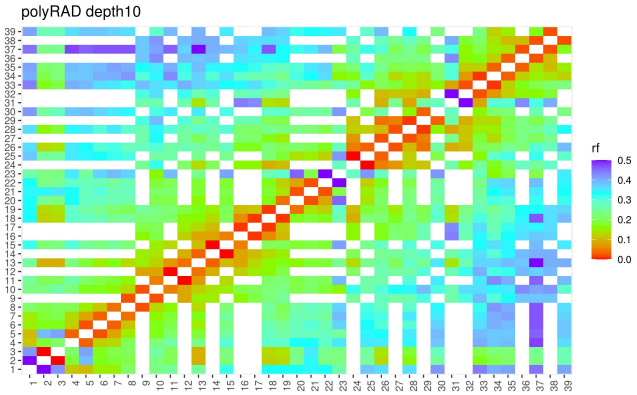

0.0550
rf
0.4
0.3
0.2
0.1
0.0

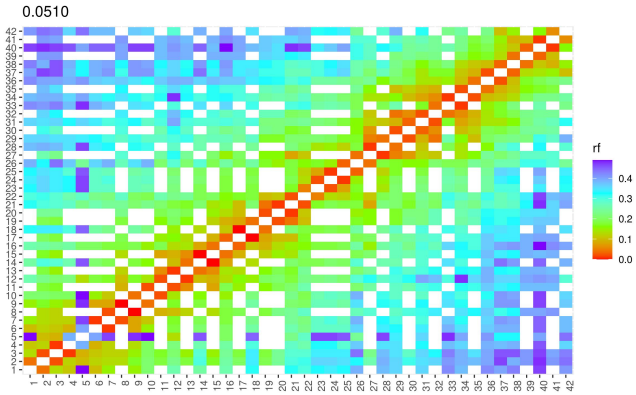

50
Real
UD
PR
SM
0.05
Distance (cM)
0
10
20
30
40
50
60
70
80
90
100
110

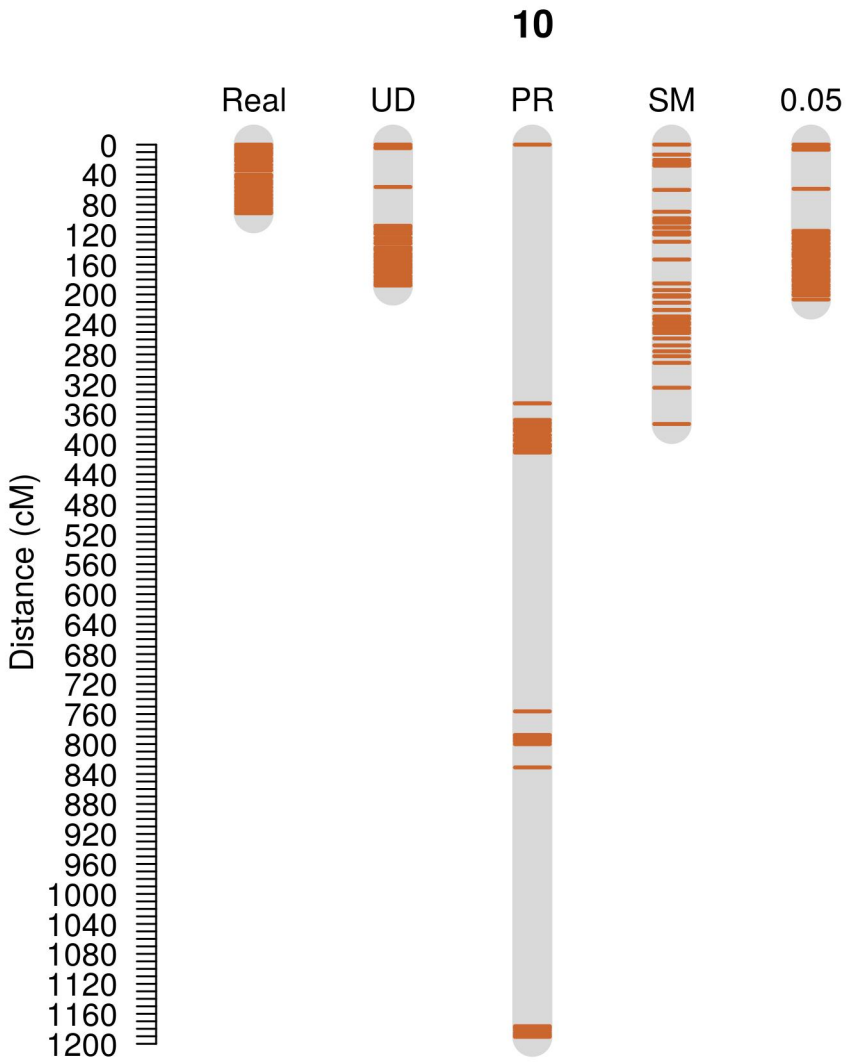

Real_10
rf
0.4
0.3
0.2
0.1
0.0


Updog depth10
rf
0.4
0.3
0.2
0.1
0.0


supermassa depth10
rf
0.5
0.4
0.3
0.2
0.1
0.0


polyRAD depth10
rf
0.5
0.4
0.3
0.2
0.1
0.0


0.0510
rf
0.4
0.3
0.2
0.1
0.0



Conclusions

The functions `pedsim2raw` and `pedsim2vcf` can be useful to test new algorithms and genetic questions which not need to consider genotyping errors. Besides updog has a robust model to simulate and identify sequencing errors, it can not be used to test its own performance. To be possible to make a fair comparison, errors must be simulated with another approach, as the one implemented in OneMap Workflows. The simulation performed here was just to demonstrate the usage; we can not make any conclusions based on that.

Simulate Illumina reads

This one is a bit more complex, and its tools are stored in the GitHub repository onemap_workflows (link 2). Please, access it for more information.

Remove generated files

```
system("rm sim* founderfile* mapfile* *rds ")
```

Links

This is a PDF version of an HTML document, which has links as references to software and documentation. For the PDF version, the links mentioned are available below:

1 - https://github.com/Cristianetaniguti/genotyping4onemap

2 - https://github.com/Cristianetaniguti/onemap_workflows

3 - https://github.com/ekg/freebayes

4 - https://gatk.broadinstitute.org/hc/en-us

5 - https://github.com/dcgerard/updog

6 - https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0030906

7 - https://github.com/lvclark/polyRAD

8 - https://github.com/tpbilton/GUSMap

9 - https://openwdl.org/

10 - https://cromwell.readthedocs.io/en/stable/

11 - https://github.com/Cristianetaniguti/onemap_workflows_shiny

12 - https://github.com/PBR/pedigreeSim

13 - https://github.com/PBR/pedigreeSim/tree/master/Manual

14 - http://augusto-garcia.github.io/onemap/vignettes_highres/Outcrossing_Populations.html

15 - https://github.com/guilherme-pereira/vcf2sm