paramlink: Parametric linkage analysis in R

Magnus Dehli Vigeland

February 1, 2012

This document gives an introduction to the R package `paramlink`, which provides various functions for likelihood based pedigree analysis, including parametric LOD scores, power analysis for linkage, genotype probability distributions, and many utility functions for manipulating pedigrees and markers. Likelihoods are calculated using the Elston-Stewart algorithm.

1 Introduction

To get started, install the `paramlink` package (if it isn't already installed) and load it.

```
> install.packages("paramlink")
> require(paramlink)
```

In `paramlink`, pedigrees and associated marker data are stored as `linkdat` objects. An object of class `linkdat` is basically a list with elements like ”pedigree”, ”markerdata”, ”model”, a.s.o.

There are 3 basic ways of creating a `linkdat` object:

1. Building it from scratch in `paramlink`.
2. Reading a ped file in standard LINKAGE format, possibly accompanied by map, dat and freq files (MERLIN style).
3. Creating the pedigree first as a data frame or matrix in R and then converting this to a `linkdat` object.

The first method is usually the quickest if you don't already have a ped file describing your pedigree. In linkage projects with many markers, method 2 is the most natural. In the next subsections we will briefly demonstrate each method. Along the way we will also see how to plot, summarize and modify pedigrees and markers.

1.1 Creating, manipulating and displaying pedigrees and markers

As an example, suppose we want to create the pedigree in Figure 1, including the genotypes. With a little practice, paramlink's utility functions provide an efficient alternative to writing ped files. Below are two possible ways of building the pedigree in Figure 1. The main difference between the two lies in the labeling of individuals: In the first, correct assignments of sex/affection status/ID label are postponed until the end. This makes the commands nice and short, but usually requires plotting the pedigree after each operation to figure out who's who so far. The second alternative assigns everything correctly from the start, making the commands slightly longer but with the advantage that the whole thing can be written without visual aid from intermediate plots.
Alternative 1:

```
> x = cousinPed(degree=1)
> x = addOffspring(x, father=7, mother=8, noffs=3)
> x = addParents(x, 6)
> x = swapSex(x, c(3, 7, 9, 11))
> x = swapAff(x, 8:10, c(0,2,2))
> x = relabel(x, new=c(11:14,21:24,31:32,41:43), old=c(1,2,12,13,5,3,4,6,7:11))
```

Alternative 2:

```
> x2 = nuclearPed(noffs=2, sex=2:1)
> x2 = relabel(x2, c(11,12,22,23))
> x2 = addOffspring(x2, father=21, mother=22, noffs=1, id=31, sex=2)
> x2 = addOffspring(x2, father=23, mother=24, noffs=1, id=32, aff=0)
> x2 = addParents(x2, id=24, father=13, mother=14)
> x2 = addOffspring(x2, father=32, mother=31, noffs=3, ids=41:43, sex=c(2,1,2), aff=c(2,2,1))
```

We can check that the two objects are in fact describing the same pedigree by using the all.equal method for `linkdat` objects provided by paramlink:

```
> all.equal(x, x2)

[1] TRUE
```

To add the genotypes, we first create a `marker` object and then add it to x:

```
> m = marker(x, 14, c('A', 0), 41:42, c('A','A'), 43, c('A','B'))
> x = addMarker(x, m)
```

Note the syntax of the `marker` function: The first argument provides a pedigree (in form of a `linkdat` object), while the arguments following it alternate between pedigree member ID's and their genotypes. In other words, the above command states that individual 14 has genotype $A/-$ (0 is the default symbol for missing alleles), 41 and 42 have A/A, 43 has A/B, while the rest is missing.

To inspect what we have made, we use the `print`, `summary` and `plot` methods written for `linkdat` objects:

```
> x

   ID FID MID SEX AFF  V1
1  11   0   0   1   1 -/-
2  12   0   0   2   1 -/-
3  22  11  12   2   1 -/-
4  23  11  12   1   1 -/-
5  21   0   0   1   1 -/-
6  13   0   0   1   1 -/-
7  14   0   0   2   1 A/-
8  24  13  14   2   1 -/-
9  31  21  22   2   1 -/-
10 32  23  24   1   0 -/-
11 41  32  31   2   2 A/A
12 42  32  31   1   2 A/A
13 43  32  31   2   1 A/B

> summary(x)

Pedigree:
---------
13 individuals
5 founders, 8 nonfounders; bit size = 11
1 nuclear subfamily
2 affected by disease, 10 unaffected, 1 with unknown affection status

Marker data:
------------
1 marker in total
9 individuals with no available genotypes: 11, 12, 22, 23, 21, 13, 24, 31, 32
12.5 % missing alleles (excluding ungenotyped individuals)

Chromosome distribution of markers:
 chromosome unknown: 1 marker
```

```
Allele number distribution:
 2 alleles: 1 marker

Model parameters:
-----------------
No model parameters set
```

```
> plot(x, marker=1)
```

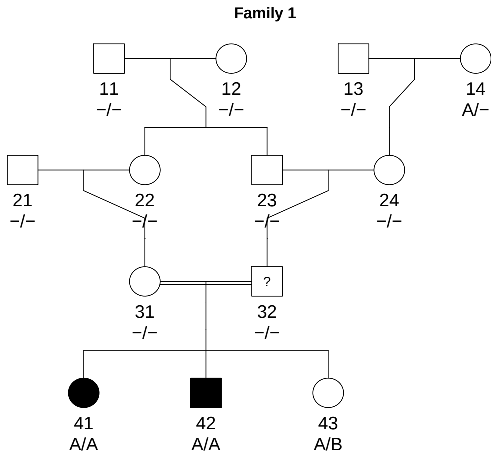


Figure 1

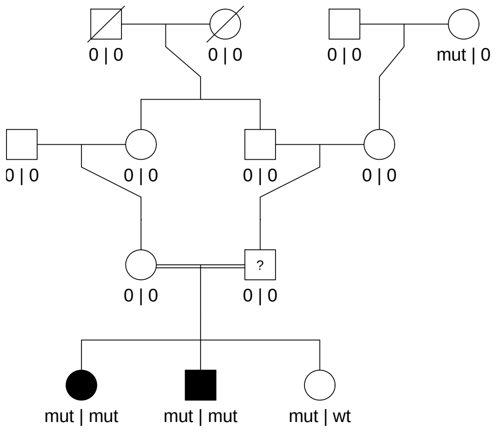


Figure 2

The plot is shown in Figure 1. The `plot` function has many optional arguments. For example, in Figure 2 we see the same pedigree and marker, but without id labels and title, with genotypes displayed differently, and some individuals marked as deceased. The plot is produced by the command:

```
> plot(x, marker=1, alleles=c('mut', 'wt'), missing="0", sep=" | ", id.labels="",
+      title="", deceased=11:12)
```

See `?plot.linkdat` for other options and more details.

To prepare for the next section, we modify our marker so that it includes information about allele frequencies and map information.

```
> x = modifyMarker(x, marker=1, afreq=c(0.1, 0.9), chr=1, pos=10, name="SNP1")
```

1.2 Writing and reading input files

In linkage analysis projects, the pedigree and markers are usually describing in a set of files generated by the genotyping software. While various formats exist, `paramlink` reads and writes files in the same format as MERLIN, one of the most popular programs for linkage analysis. If you are unfamiliar with these formats, please consult the web tutorial for MERLIN at `http://www.sph.umich.edu/csg/abecasis/Merlin/tour/`.

Writing all the information contained in our `linkdat` object `x` to a collection of files, is done by the following command.

```
> write.linkdat(x, prefix="ex")
```

The 4 files created by default are ”ex.ped”, ”ex.dat”, ”ex.map” and ”ex.freq”. If we had set a model for `x` (see next section), a file ”ex.model” would also have been made. If only a subset of these files are needed, say the .map and .dat files, this could be achieved by calling `write.linkdat(x, prefix="ex", what=c("map", "dat"))`.

To build a `linkdat` object based on the files created above, we use the `linkdat` function:

```
> y = linkdat("ex.ped", dat="ex.dat", map="ex.map", freq="ex.freq")
```

```
Family name: 1.
Pedigree read, 13 individuals.
Loop(s) detected.
1 marker read.
```

```
> all.equal(y,x)
```

```
[1] TRUE
```

The default symbol for missing alleles in the ped file is "0". If your file uses something else, say "x", simply add `missing="x"` as an argument in the `linkdat` command.

If the frequency file is not included, all markers will be taken as equifrequent. In particular, markers with zero or one non-missing alleles present in the ped file will be interpreted as monoallelic.

If map/dat files are not included, the markers present in the ped file will have missing name, chromosome number and position. In certain analysis where these are required (e.g. in MERLIN runs), a dummy map is then created, placing the markers on chromosome one, 1 cM apart, and with names "M1", "M2", a.s.o.

In general, the first argument of the `linkdat` function can be either the (path and) name of a pedigree file (as above), or a `matrix` or `data.frame` in LINKAGE format. The latter can be useful in cases where the pedigree file has a non-standard format. One can then load it into R with `read.table`, do the necessary tweaking, and then feed it to `linkdat`.

2 Parametric linkage analysis

2.1 Setting the disease model

Before linkage analysis can be performed, parameters describing a disease model must be specified. The parameters include whether the disease is autosomal or X-linked, penetrance values, phenocopy rate and allele frequencies at the disease locus. All of these are controlled by the `setModel` function. For the simplest cases, there is a shortcut: Setting `model=i` for an integer `i` in the range 1-4, is interpreted as follows:

1 = Autosomal dominant

2 = Autosomal recessive

3 = X-linked dominant

4 = X-linked recessive

If nothing else is indicated, the other parameters are given default values: Full penetrance, no phenocopies, disease allele frequency=0.00001.

For our example pedigree in Figure 1 we want an autosomal recessive model:

```
> x = setModel(x, model=2)
> x$model
```

```
Autosomal inheritance with penetrances: (f0, f1, f2) = (0, 0, 1)
Disease allele frequency: 1e-05
```

The shortcut `model=2` above is equivalent to the following command, where the parameters are specified individually:

```
> x = setModel(x, chrom="autosomal", penetrances=c(0, 0, 1), dfreq=0.00001)
```

Note in particular the penetrance parameters, given as a vector of length 3 of the form penetrances = (f_0, f_1, f_2) using the conventional notation for penetrance values:

$$f_i = P(\text{affected} \mid i \text{ copies of the disease allele}).$$

If the input `linkdat` object to `setModel` already has a model, `setModel` uses the existing model values for any missing arguments. This makes it easy to change one parameter while keeping everything else as before. For example, the command `x = setModel(x, penetrances=c(0.01, 0.01, 0.9))` alters the penetrance values (to indicate 1% phenocopy rate and 90% penetrance) keeping everything else as before.

2.2 Singlepoint LOD scores

The function `lod` computes singlepoint LOD scores for all the markers of any `linkdat` object. There is no automatic loop breaking implemented at the moment, so loop breakers must be specified manually:

```
> lod(x, loop_breaker=32)

      SNP1
0 1.193642
```

By default, the recombination fraction between the disease locus and the marker is set to $\theta = 0$, but this can be changed using the `theta` argument:

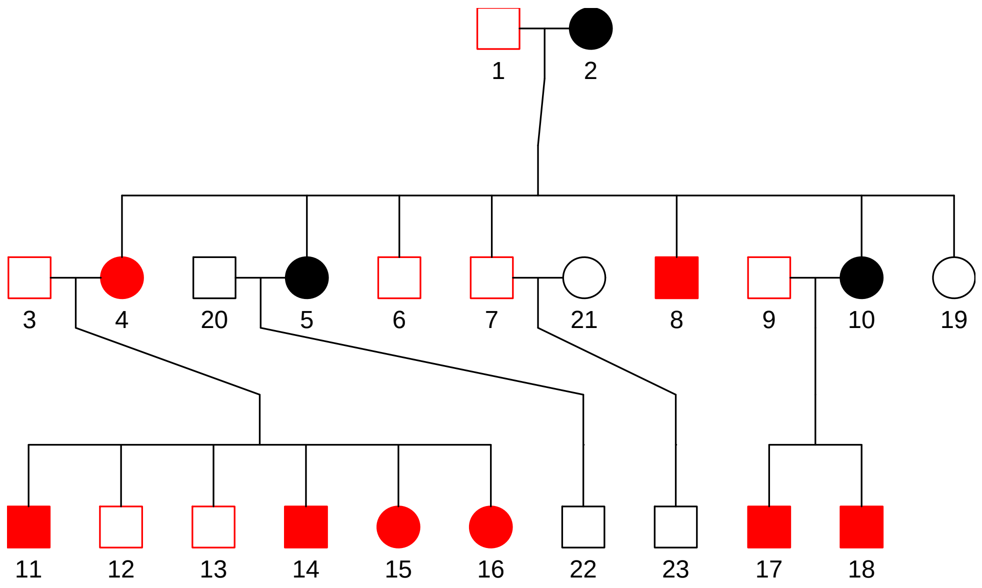
```
> lod(x, theta=c(0, 0.1, 0.2), loop_breaker=32)

         SNP1
0   1.1936416
0.1 0.8599471
0.2 0.5405310
```

For a more interesting example, let us load the `dominant` dataset: A data frame describing a pedigree with 23 members and genotypes at 650 SNP markers.

```
> data(dominant)
> y = linkdat(dominant)

Family name: 1.
Pedigree read, 23 individuals.
5 maximal nuclear subfamilies. Peeling order set.
650 markers read.
```

As usual we plot the pedigree before doing any analysis. Note the `available=T` option, which displays the genotyped individuals in red. Running `summary(y)` (not done here) is also a good idea to get an overview of the data.

```
> plot(y, available=T)
```



The pedigree suggests an autosomal dominant mode of inheritance, so we set this using the shortcut `model=1` explained in the previous section.

```
> y = setModel(y, 1)
> y$model
```

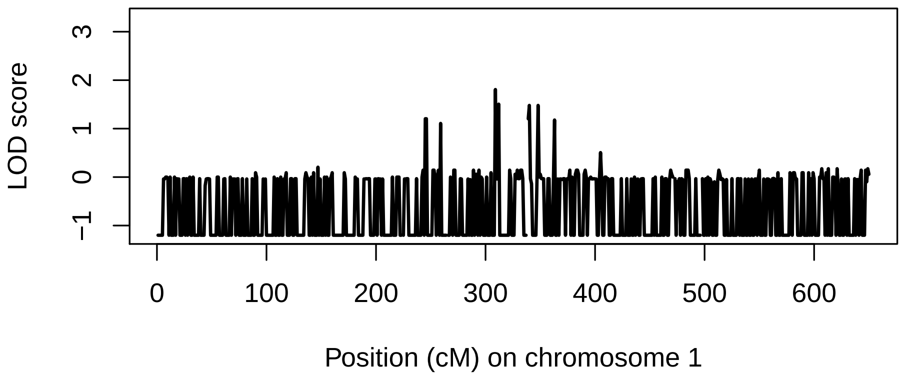
```
Autosomal inheritance with penetrances: (f0, f1, f2) = (0, 1, 1)
Disease allele frequency: 1e-05
```

Now we compute the LOD scores, and plot the results. Since we haven't provided map info for the 650 markers, a dummy map will be used in the plot, placing all markers on chromosome 1.

```
> lods = lod(y)
> summary(lods)
```

```
Max LOD score: 1.80618
Achieved at marker(s): M309
```

```
> plot(lods)
```



There is a peak near position 300, but it is far from the standard significance threshold of LOD = 3.3.

2.3 Multipoint analysis: Running MERLIN from paramlink

If MERLIN is installed, and properly pointed to in the PATH variable on your computer, `paramlink` offers a convenient wrapper function, simply called `merlin`. Its default action is to generate (and afterwards remove) files named "merlin.ped", "merlin.dat", "merlin.map", "merlin.freq" and "merlin.model" and run the following command via `system`:

merlin -p merlin.ped -d merlin.dat -m merlin.map -f merlin.freq –model merlin.model

For example, let us check MERLIN's evaluation of the marker in Figure 1:

```
> merlin(x)
```

```
   SNP1
0 1.446
```

Comparing this with the result $LOD = 1.19$ that we found in the previous section, we see that MERLIN gives a different score than `paramlink` for this marker. This is because of the partial genotype $A/0$ of individual 14. While `paramlink` handles partial genotypes, MERLIN treats them simply as missing. To check that this in fact the reason, we recalculate the LOD score in `paramlink` after setting the genotype of individual 14 to be missing:

```
> x = modifyMarker(x, marker=1, id=14, genotype=c(0,0))
> lod(x, loop_breaker=32)
```

```
      SNP1
0 1.446123
```

This agrees with the MERLIN result.

Turning to the `dominant` dataset, the complete family is too big for MERLIN to cope with directly. (Running `summary(y)` shows that the bit size is 28, which is larger than MERLIN's default limit of 24.) However, we can trim the pedigree and do an "affected only" analysis. The trimming is done by the `trim` function, whose second argument decides if one should keep all *affected* or all *available* individuals:

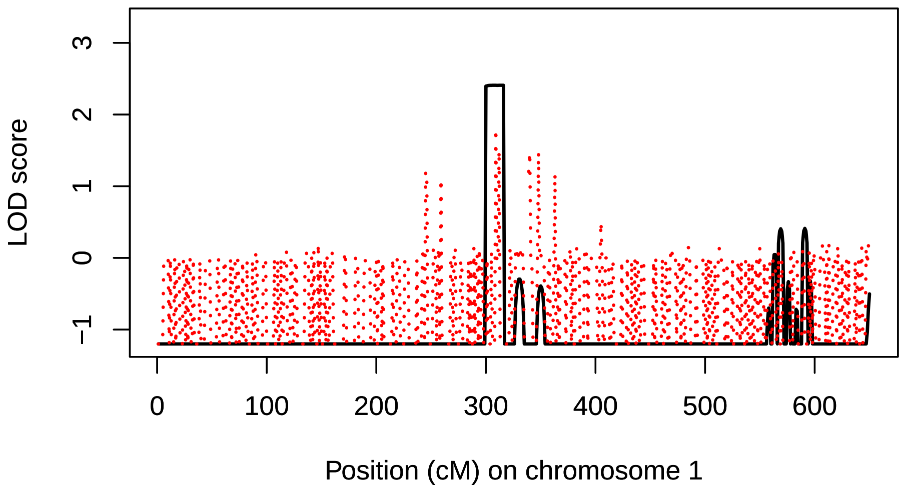
```
> y_aff = trim(y, keep="affected")

Removing individuals: 6, 7, 12, 13, 19, 20, 21, 22 and 23
```

The resulting pedigree is small enough for MERLIN (`summary(y_aff)` tells us the bit size is down to 16). Multipoint LOD scores can then be obtained by calling the wrapper function `merlin`:

```
> mlods = merlin(y_aff)
```

We plot the results together with the singlepoint scores (dashed red line) that we computed in the last section for the complete family:

```
> plot(mlods)
> par(new=T); plot(lods, lty=3, col="red")
```



The `merlin` command has several optional arguments, described in detail in the help page `?merlin`. For instance, use `outputfile` to write the MERLIN output to a file; `markers` if only a subset of the markers should be considered; and `options` if additional parameters should be passed on to MERLIN. (The contents of `options` (which must be a single character string) is simply pasted onto the command, so it must include dashes and spaces. See example below.) In some cases you might not want the "–model merlin.model" part of the command: this is removed by setting `model=FALSE`. For example, the MERLIN syntax for computing the likelihood of all the marker data, is

merlin -p merlin.ped -d merlin.dat -m merlin.map -f merlin.freq –lik

The corresponding command in `paramlink` would be:

```
> merlin(y_aff, model=F, options="--lik")
```

Note: When `model=F`, no parsing is done of the results; the whole MERLIN output is returned as a character vector.

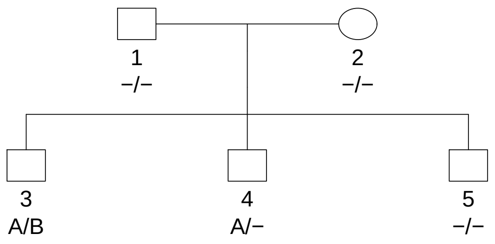


Figure 3

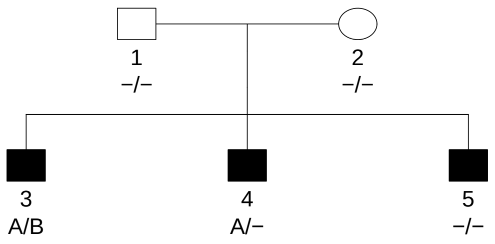


Figure 4

3 Genotype probability distributions

The `paramlink` package provides two functions for calculating genotype distributions of specific family members, conditional on the known alleles at the involved markers: `oneMarkerDistribution` and `twoMarkerDistribution`.

The first works on a single marker and finds the joint genotype distribution for any number of individuals. If the pedigree is affected with a monogenic disease for which a model has been set, the distribution can be computed conditional on a given recombination fraction between the marker and the disease locus.

The second function, `twoMarkerDistribution`, takes a single individual as input, and computes the joint distribution of his/her genotypes at two markers, separated by a given recombination fraction.

For a simple illustration on the use of these functions, we create a nuclear family with three offspring, and a SNP marker for which some alleles are known.

```
> z = nuclearPed(3)
> marker1 = marker(z, 3, c('A','B'), 4, c('A', 0))
> z = addMarker(z, marker1)
> plot(z, 1)
```

The plot is shown in Figure 3. To begin with, let us find the genotype distribution of individual 5:

```
> dist1 = oneMarkerDistribution(z, id=5, partialmarker=1)

Assuming that the marker is autosomal.

Partial marker data:
 ID V1
  1 --
  2 --
  3 AB
  4 A-
  5 --

Allele frequencies:
  A   B
0.5 0.5

Genotype probability distribution for individual 5:
    AA     BB     AB
0.2115 0.1346 0.6538
```

Written as integer fractions, the conditional probabilities are $P(AA) = 11/52$, $P(BB) = 7/52$, $P(AB) = 34/52$. At first glance these numbers appear somewhat more complicated than we might have expected. The reason is of course that the missing allele of individual 4 leaves open many possibilities for the genotypes of the parents. In fact, their joint distribution is as follows (this time we add `verbose=F` to reduce the output on the screen):

```
> parents = oneMarkerDistribution(z, id=1:2, partialmarker=1, verbose=F)
```

The rows of the result matrix correspond to the genotypes of the father (individual 1), and the columns to those of the mother. After a little trial and error, we find an integral representation with 13 as the common denominator:

```
> parents * 13
```

```
   AA BB AB
AA  0  2  2
BB  2  0  1
AB  2  1  3
```

Given the distribution of the parents it is easy to compute the probabilities for individual 5, checking the results above. For example, if G denotes the genotype of individual 5, we have

$$\begin{aligned} Pr(G = AA) =& Pr(G = AA \mid \text{father } AA, \text{ mother } AB) \cdot Pr(\text{father } AA, \text{ mother } AB) \\ &+ Pr(G = AA \mid \text{father } AB, \text{ mother } AA) \cdot Pr(\text{father } AB, \text{ mother } AA) \\ &+ Pr(G = AA \mid \text{both parents } AB) \cdot Pr(\text{both parents } AB) \\ =& \frac{1}{2} \cdot \frac{2}{13} + \frac{1}{2} \cdot \frac{2}{13} + \frac{1}{4} \cdot \frac{3}{13} = \frac{11}{52} \end{aligned}$$

which is the same as we found above.

Next, suppose the three offspring are affected with a recessive disorder, and that the marker is completely linked to the causal gene.

```
> z2 = swapAff(z, 3:5)
> z2 = setModel(z2, model=2)
> plot(z2, 1)
```

The plot is shown in Figure 4. How does this affect the distribution of individual 5?

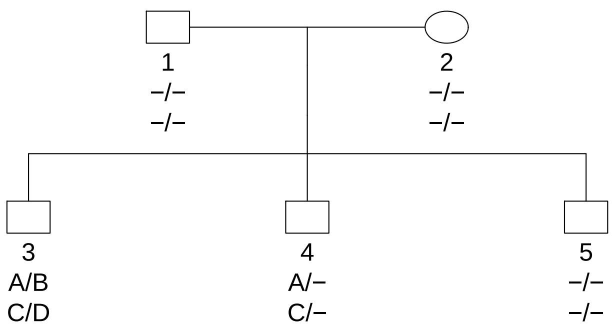
```
> oneMarkerDistribution(z2, id=5, partialmarker=1, theta=0, verbose=F)
```

```
AA BB AB
 0  0  1
```

We see that the genotype of individual 5 is now forced to be AB. This was to be expected: Since we assume no recombination between the marker and disease loci, all the affected offspring must carry the same haplotypes. Since individual 3 has genotype AB, so must 4 and 5.

Finally we turn to `twoMarkerDistribution`. Suppose there is another SNP for which the available genotype data is similar to that of `marker1`. For clarity we let the allele names of the new marker be C and D.

```
> marker2 = modifyMarker(z, marker1, alleles=c('C', 'D'))
> z = addMarker(z, marker2)
> plot(z, marker=1:2)
```



Suppose first that the two markers segregate independently of each other, i.e. that their recombination fraction is $\theta = 0.5$. The joint distribution of the genotypes of individual 5 is then:

```
> twoMarkerDistribution(z, id=5, partialmarker1=1, partialmarker2=2, theta=0.5)

Conditioning on the following marker data:
 ID M1 M2
  1 -- --
  2 -- --
  3 AB CD
  4 A- C-
  5 -- --

Allele frequencies, marker 1:
  A   B
0.5 0.5

Allele frequencies, marker 2:
  C   D
0.5 0.5

Recombination rate between marker loci: 0.5

Joint genotype distribution at the two markers for individual 5:
       CC     DD     CD
AA 0.0447 0.0285 0.1383
BB 0.0285 0.0181 0.0880
AB 0.1383 0.0880 0.4275
```

Since the two markers are assumed to be independent, an alternative way of finding their joint distribution would be to take the (tensor) square of the vector `dist1` obtained above:

```
> dist1 %o% dist1

           AA         BB         AB
AA 0.04474852 0.02847633 0.13831361
BB 0.02847633 0.01812130 0.08801775
AB 0.13831361 0.08801775 0.42751479
```

However, if the two markers were completely linked, the distribution would be different:

```
> twoMarkerDistribution(z, id=5, partialmarker1=1, partialmarker2=2, theta=0, verbose=F)

           CC         DD         CD
AA 0.06104651 0.03488372 0.11337209
BB 0.03488372 0.03488372 0.06976744
AB 0.11337209 0.06976744 0.46802326
```

Although not impossible, calculating these probabilities by hand is not a pleasant task even in this simple example.