Data input vignette
Reading genotype data in `snpStats`

David Clayton

October 24, 2025

Memory limitations

Before we start it is important to emphasise that the `SnpMatrix` objects that hold genotype data in `snpStats` are resident in memory, and limitations of the computer and of the R language impose limits on the maximum size of datasets that can be held at any one time. Each genotype reading uses only a single byte of memory so that large datasets can be read given the large memory capacity of modern computers. Originally, R imposed a limit of $2^{31} - 1 \sim 2 \times 10^9$ elements in a single array. This limit applied in both the 32-bit and 64-bit versions of R, these versions differing only in the *total* memory that could be used. For example, this would correspond to one million loci for two thousand subjects and would occupy two gigabytes of machine memory. However, version 3 of R removed the restriction on single arrays in the 64-bit version, and this was implemented for `SnpMatrix` and `XSnpMatrix` objects in version 1.19.2 of `snpStats`. However, experience of this code is limited and some caution is advised.

Reading pedfiles

A commonly encountered format for storing genotype data is the "pedfile" format, which originated some years ago in the LINKAGE package. Pedfiles are text files containing one line per genotyped sample, with fields separated by "white space" (TAB characters or SPACEs). The first six fields contain:

1. a pedigree or family identifier, unique to the family of which this subject is a member,
2. a further identifier, unique (within the family) to each family member,
3. the member identifier of the father of the subject if the father is also present in the data, otherwise an arbitrary code (usually `0`),
4. similarly, an identifier for the mother of the subject,

5. the sex of the subject (`1` = Male, `2` = Female), and

6. a binary trait indicator (`1` = Absent, `2` = Present).

Missing values in the last two fields are usually coded as zero.

The first few rows and columns of a sample file is shown below:

```
IBD054 430 0 0 1 0 1  3 3  1 4  1 4  2
IBD054 412 430 431 2 2 1  3 1  3 4  1 4  2
IBD054 431 0 0 2 0 3  3 3  3 1  1 2  2
IBD058 438 0 0 1 0 3  3 3  3 1  1 2  2
IBD058 470 438 444 2 2 3  3 3  3 1  1 2  2
```

Thus, the subject of line 2 has a father whose data appears on line 1 and a mother whose data is on line 3. The grandparents do not appear on the file. This subject is affected by the trait, but the trait status of her parents is not known. The genotypes of this subject at the first four loci are `1/3`, `1/3`, `4/1` and `4/2`. Note that `snpStats` will only deal with diallelic data and, although alleles are coded 1 to 4 in this file, only two of these occur with in any one locus. In fact these data are from the sample dataset distributed with the HAPLOVIEW program (Barrett et al., 2005) which uses the numbers 1–4 to denote the four nucleotides: `1` = A, `2` = C, `3` = G, `4` = . The pedfile contains data for 20 loci on 120 subjects, and is accompanied by a second file which describes the loci, the first four lines being:

```
IGR1118a_1 274044
IGR1119a_1 274541
IGR1143a_1 286593
IGR1144a_1 287261
```

(this file is rather simple, containing just the locus name and its position on a chromosome).

The (gzipped) pedfile and the locus information file are stored in the `extdata` subdirectory of the `snpStats` package as, respectively, `sample.ped.gz` and `sample.info`. Since the precise location of these files may vary between installations, we first obtain full paths to these files using the `system.file` function

```
> pedfile <- system.file("extdata/sample.ped.gz", package="snpStats")
> pedfile

[1] "/tmp/RtmprtzMqc/Rinst183cd3111262d0/snpStats/extdata/sample.ped.gz"

> infofile <- system.file("extdata/sample.info", package="snpStats")
```

The data can then be read in using the `read.pedfile` function

```
> sample <- read.pedfile(pedfile, snps=infofile)
```

The result, `sample`, is a list with three elements. The first is an object of class `SnpMatrix` containing the genotype data. We shall show summaries for the first few loci

```
> sample$genotypes

A SnpMatrix with  120 rows and  20 columns
Row names:  430 ... 17702
Col names:  IGR1118a_1 ... IGR2020a_1

> col.summary(sample$genotypes)$MAF

 [1] 0.14957265 0.14224138 0.15833333 0.15000000 0.13392857 0.15929204
 [7] 0.14678899 0.14035088 0.06578947 0.14166667 0.13839286 0.15566038
[13] 0.13750000 0.14166667 0.31250000 0.26470588 0.27155172 0.50000000
[19] 0.28947368 0.02232143

> head(col.summary(sample$genotypes))

           Calls Call.rate Certain.calls       RAF       MAF        P.AA
IGR1118a_1   117 0.9750000             1 0.8504274 0.1495726 0.008547009
IGR1119a_1   116 0.9666667             1 0.1422414 0.1422414 0.724137931
IGR1143a_1   120 1.0000000             1 0.8416667 0.1583333 0.008333333
IGR1144a_1   120 1.0000000             1 0.8500000 0.1500000 0.008333333
IGR1169a_2   112 0.9333333             1 0.8660714 0.1339286 0.000000000
IGR1218a_2   113 0.9416667             1 0.8407080 0.1592920 0.008849558
                P.AB       P.BB    z.HWE
IGR1118a_1 0.2820513 0.70940171 1.175622
IGR1119a_1 0.2672414 0.00862069 1.025043
IGR1143a_1 0.3000000 0.69166667 1.375728
IGR1144a_1 0.2833333 0.70833333 1.217161
IGR1169a_2 0.2678571 0.73214286 1.636547
IGR1218a_2 0.3008850 0.69026549 1.311673
```

The second list element is a dataframe containing the first six fields of the pedfile. We'll just display the start of this:

```
> head(sample$fam)

    pedigree member father mother sex affected
430   IBD054    430   <NA>   <NA>   1       NA
412   IBD054    412    430    431   2        2
431   IBD054    431   <NA>   <NA>   2       NA
438   IBD058    438   <NA>   <NA>   1       NA
470   IBD058    470    438    444   2        2
444   IBD058    444   <NA>   <NA>   2       NA
```

Note that the zero values in the pedfile have been read as NA; this is optional, but default, behaviour of the function. Here the pedigree-member identifiers have been used as subject identifiers, since these are not duplicated while pedigree identifiers (the first choice) were duplicated (if both sets of identifiers are duplicated, they are combined). Finally, the third list element is a dataframe containing the information read from the sample.info file, to which have been added the two alleles found at each locus:

```
> head(sample$map)

   snp.names     V2 allele.1 allele.2
1 IGR1118a_1 274044        1        3
2 IGR1119a_1 274541        3        1
3 IGR1143a_1 286593        4        1
4 IGR1144a_1 287261        4        2
5 IGR1169a_2 299755        2        1
6 IGR1218a_2 324341        3        1
```

Here we have used the default settings of read.pedfile. In particular, it is not mandatory to supply a locus description file and there are further arguments which allow additional flexibility. These options are described in the on-line help page.

PLINK files

Binary PED (BED) files written by the PLINK toolset (Purcell et al., 2007) may also be read as SnpMatrix objects. Files of type .bed are written by the plink -make-bed command and are accompanied by two text files: a .fam file containing the first six fields of a standard pedfile as described above, and a .bim file which describes the loci. The package data directory also contains .bed, .fam and .bim files for the sample dataset of the last section; the following commands recover the full file paths for these files and read the files:

```
> fam <- system.file("extdata/sample.fam", package="snpStats")
> bim <- system.file("extdata/sample.bim", package="snpStats")
> bed <- system.file("extdata/sample.bed", package="snpStats")
> sample <- read.plink(bed, bim, fam)
```

The output object is similar to that produced by read.pedfile, a list with three elements:

```
> sample$genotypes

A SnpMatrix with  120 rows and  20 columns
Row names:  430 ... 17702
Col names:  IGR1118a_1 ... IGR2020a_1

> col.summary(sample$genotypes)$MAF
```

```
 [1] 0.14957265 0.14224138 0.15833333 0.15000000 0.13392857 0.15929204
 [7] 0.14678899 0.14035088 0.06578947 0.14166667 0.13839286 0.15566038
[13] 0.13750000 0.14166667 0.31250000 0.26470588 0.27155172 0.50000000
[19] 0.28947368 0.02232143
```

```
> head(sample$fam)
```

```
    pedigree member father mother sex affected
430   IBD054    430     NA     NA   1       NA
412   IBD054    412    430    431   2        2
431   IBD054    431     NA     NA   2       NA
438   IBD058    438     NA     NA   1       NA
470   IBD058    470    438    444   2        2
444   IBD058    444     NA     NA   2       NA
```

```
> head(sample$map)
```

```
           chromosome   snp.name cM position allele.1 allele.2
IGR1118a_1         NA IGR1118a_1 NA   274044        1        3
IGR1119a_1         NA IGR1119a_1 NA   274541        1        3
IGR1143a_1         NA IGR1143a_1 NA   286593        4        1
IGR1144a_1         NA IGR1144a_1 NA   287261        4        2
IGR1169a_2         NA IGR1169a_2 NA   299755        2        1
IGR1218a_2         NA IGR1218a_2 NA   324341        3        1
```

Usually the three input files have the same filename stub with `.bed`, `.fam` and `.bim` extensions added. In this case it is sufficient to just supply the filename stub to `read.plink`.

A useful feature of `read.plink` is the ability to select a subset of data from a large PLINK dataset. This is demonstrated in our small example below

```
> subset <- read.plink(bed, bim, fam, select.snps=6:10)
> subset$genotypes
```

```
A SnpMatrix with  120 rows and  5 columns
Row names:  430 ... 17702
Col names:  IGR1218a_2 ... IGR1373a_1
```

```
> col.summary(subset$genotypes)$MAF
```

```
[1] 0.15929204 0.14678899 0.14035088 0.06578947 0.14166667
```

```
> subset$map
```

```
           chromosome   snp.name cM position allele.1 allele.2
IGR1218a_2         NA IGR1218a_2 NA   324341        3        1
IGR1219a_2         NA IGR1219a_2 NA   324379        4        2
IGR1286a_1         NA IGR1286a_1 NA   358048        3        2
TSC0101718         NA TSC0101718 NA   366811        4        3
IGR1373a_1         NA IGR1373a_1 NA   395079        2        4
```

Note that, in order to select certain SNPs, the input PLINK file must be in SNP-major order *i.e.* all individuals for the first SNP, all individuals for the second SNP, and so on. This is the default mode in PLINK. However, to select certain individuals, the input PLINK file must be in individual-major order.

Long format data

The least compact, but perhaps most flexible, input format is the "long" format in which each genotype call takes up a single line. Such data can be read using the function `read.snps.long`. A simple example is provided by the small gzipped data file `sample-long.gz` provided with the package:

```
> longfile <- system.file("extdata/sample-long.gz", package="snpStats")
> longfile

[1] "/tmp/RtmprtzMqc/Rinst183cd3111262d0/snpStats/extdata/sample-long.gz"
```

The first 5 lines of the file are listed as follows:

```
> cat(readLines(longfile, 5), sep="\n")

snp1        subject1        1        1.000
snp1        subject2        2        1.000
snp1        subject3        1        1.000
snp1        subject4        1        1.000
snp1        subject5        2        1.000
```

The first field gives the SNP identifier (`snp1` to `snp18`), the second gives the sample, or subject, identifier (`subject1` to `subject100`), the third field gives the genotype call (`1=A/A`, `2=A/B`, `3=B/B`), and the last field gives a confidence measure for the call (here always `1.000`). To read in this file and inspect the data:

```
> gdata <- read.long(longfile,
+    fields=c(snp=1, sample=2, genotype=3, confidence=4),
+    gcodes=c("1", "2", "3"),
+    threshold=0.95)
> gdata
```

```
A SnpMatrix with  100 rows and  18 columns
Row names:  subject1 ... subject100
Col names:  snp1 ... snp18

> summary(gdata)

$rows
   Call.rate Certain.calls Heterozygosity
 Min.   :1   Min.   :1     Min.   :0.1111
 1st Qu.:1   1st Qu.:1     1st Qu.:0.2778
 Median :1   Median :1     Median :0.3333
 Mean   :1   Mean   :1     Mean   :0.3478
 3rd Qu.:1   3rd Qu.:1     3rd Qu.:0.3889
 Max.   :1   Max.   :1     Max.   :0.6667

$cols
     Calls       Call.rate Certain.calls      RAF              MAF
 Min.   :100   Min.   :1   Min.   :1     Min.   :0.0450   Min.   :0.0450
 1st Qu.:100   1st Qu.:1   1st Qu.:1     1st Qu.:0.2700   1st Qu.:0.1087
 Median :100   Median :1   Median :1     Median :0.4475   Median :0.2850
 Mean   :100   Mean   :1   Mean   :1     Mean   :0.4383   Mean   :0.2739
 3rd Qu.:100   3rd Qu.:1   3rd Qu.:1     3rd Qu.:0.5913   3rd Qu.:0.4300
 Max.   :100   Max.   :1   Max.   :1     Max.   :0.9100   Max.   :0.4900
      P.AA             P.AB             P.BB            z.HWE
 Min.   :0.0000   Min.   :0.0700   Min.   :0.0000   Min.   :-1.85573
 1st Qu.:0.1525   1st Qu.:0.2025   1st Qu.:0.0875   1st Qu.:-0.62145
 Median :0.2850   Median :0.4000   Median :0.1750   Median : 0.25448
 Mean   :0.3878   Mean   :0.3478   Mean   :0.2644   Mean   : 0.09661
 3rd Qu.:0.5475   3rd Qu.:0.4600   3rd Qu.:0.3350   3rd Qu.: 0.86390
 Max.   :0.9200   Max.   :0.5500   Max.   :0.8200   Max.   : 1.19206
```

A few remarks:

1. In our example, the entire file has been read. However, subsets of data may be extracted by specifying the required SNP or sample identifiers.

2. Any calls for which the call confidence is less than `threshold` is set to `NA` (this did not affect any calls in this simple example).

3. Here, calls were represented by a single genotype code. It is also possible to read calls as pairs of alleles. The function then returns a list whose first argument is the `SnpMatrix` object, and whose second object is a dataframe containing the allele codes. This option is demonstrated below, using an alternative coding of the same data (all SNPs are `CT` SNPs):

```
> allelesfile <- system.file("extdata/sample-long-alleles.gz", package="snpStats")
> cat(readLines(allelesfile, 5), sep="\n")
```

```
snp1        subject1        C        C        1.000
snp1        subject2        C        T        1.000
snp1        subject3        C        C        1.000
snp1        subject4        C        C        1.000
snp1        subject5        C        T        1.000
```

```
> gdata <- read.long(allelesfile,
+    fields=c(snp=1, sample=2, allele.A=3, allele.B=4, confidence=5),
+    threshold=0.95)
> gdata
```

```
$genotypes
A SnpMatrix with  100 rows and  18 columns
Row names:  subject1 ... subject100
Col names:  snp1 ... snp18

$alleles
      allele.A allele.B
snp1         C        T
snp2         C        T
snp3         C        T
snp4         C        T
snp5         C        T
snp6         C        T
snp7         T        C
snp8         C        T
snp9         T        C
snp10        T        C
snp11        C        T
snp12        T        C
snp13        C        T
snp14        T        C
snp15        C        T
snp16        C        T
snp17        C        T
snp18        C        T
```

```
> gdata$genotypes
```

```
A SnpMatrix with  100 rows and  18 columns
Row names:  subject1 ... subject100
Col names:  snp1 ... snp18
```

```
> gdata$alleles
```

```
      allele.A allele.B
snp1         C        T
snp2         C        T
snp3         C        T
snp4         C        T
snp5         C        T
snp6         C        T
snp7         T        C
snp8         C        T
snp9         T        C
snp10        T        C
snp11        C        T
snp12        T        C
snp13        C        T
snp14        T        C
snp15        C        T
snp16        C        T
snp17        C        T
snp18        C        T
```

Note that the assignment of alleles depends on the order in which they were encountered.

This function has many options and the online help page needs to be read carefully.

Other formats

Imputation

A further source of input data is programs which can *impute* genotype data for a set of study individuals, using genome-wide SNP-chip data for the study subjects plus HapMap or 1,000 genomes project datasets. `snpStats` provides the functions `read.beagle`, `read.impute`, and `read.mach` to read in files produced by the leading imputation programs. For more details of such data, see the imputation and meta-analysis vignette.

VCF format

The 1,000 genomes data are released in the VCF format. `snpStats` does not yet include a function to read data files in this format, but the `GGtools` package does contain such a function (`vcf2sm`).

X, Y and mitocondrial SNPs

The `SnpMatrix` class is designed for diploid SNP genotypes. SNPs which can be haploid are stored in objects of the `XSnpMatrix` class, which has an addition slot, named `diploid`. Since, for the X chromosome, ploidy depends on sex and may vary from row to row, this (logical) vector has the same number of elements as the number of rows in the SNP data matrix. Most input routines do not allow for reading an `XSnpMatrix` and simply read into a `SnpMatrix`, coding haploid calls as (homozygous) diploid. Such objects may then be coerced into the `XSnpMatrix` class using `as(..., "XSnpMatrix")` or `new("XSnpMatrix, ..., diploid=...)`. If `as` is used, ploidy is inferred from homozygosity while, if `new` is used, it must be supplied (if all rows have the same ploidy, this argument can be a scalar). In either case, calls presumed to be haploid but coded as heterozygous will be set to `NA`.

Reference

Barrett JC, Fry B, Maller J, Daly MJ.(2005) Haploview: analysis and visualization of LD and haplotype maps. *Bioinformatics*, 2005 Jan 15, [PubMed ID: 15297300]

Purcell S, Neale B, Todd-Brown K, Thomas L, Ferreira MAR, Bender D, Maller J, Sklar P, de Bakker PIW, Daly MJ and Sham PC (2007) PLINK: a toolset for whole-genome association and population-based linkage analysis. *American Journal of Human Genetics*, **81**