ElstonStewart package vignette
Version 1.0

Hervé Perdry

June 9, 2014

1 Introduction

The Elston-Stewart algorithm allows to compute probability functions in pedigrees.

Consider individuals indexed by $i = 1, 2, \ldots,$ with genotype G_i and phenotype X_i. The set of possible genotypes for individual i is g_i and the phenotype is x_i. The algorithm allows to compute probabilities of the form

$$P_\theta\left(\bigcap_i (G_i \in g_i, X_i = x_i)\right).$$

It needs the specification of a *model*, consisting in

1. genotype probabilities for founder $P_\theta(G = g)$
2. genotype transition probabilities from parents to offspring
$$P_\theta(G_{\text{of}} = g_{\text{of}} | G_{\text{fa}} = g_{\text{fa}}, G_{\text{mo}} = g_{\text{mo}})$$
3. phenotype probabilities, conditional to genotype $P_\theta(X = x|G = g)$.

This package implements Elston-Stewart algorithm, including for pedigrees with imbreding, allowing the user to specify the model of her choice. It uses memoization (aka known as dynamic programming), allows vectorization, and, when various pedigrees are considered at the same time, allows parallel computing.

2 An example model

The functions `Elston` and `Likelihood` have a parameter `model`, which is a list specifying the model. Its composnents are

- `name`: the model name, used in the memoization: if you use different models, be sure to give them different names

- `proba.g`: a function giving genotypes probabilities
- `trans`: a function giving transition probabilities from parent to offspring
- `p.pheno`: a function giving phenotype probabilities, conditional to genotype

The package provides an example model, `model.di` which can be a template for other models. It's a model for a di-allelic loci, the genotypes being coded additively (0, 1, 2 according to the number of the alternate allele). Let's examine the model components.

`modele.di$proba.g` gives Hardy-Weinberg probabilities, using `theta$p` which is the reference allele frequency:

```
> modele.di$proba.g
```

```
function (g, theta)
{
    if (g == 0)
        return(theta$p^2)
    if (g == 1)
        return(2 * theta$p * (1 - theta$p))
    if (g == 2)
        return((1 - theta$p)^2)
    stop("Unknown Genotype")
}
<environment: namespace:ElstonStewart>
```

`modele.di$trans` gives transition probabilities, according to Mendel's rules:

```
> modele.di$trans
```

```
function (of, fa, mo)
{
    if (fa == 0) {
        if (mo == 0 & of == 0)
            return(1)
        if (mo == 1 & (of == 0 | of == 1))
            return(0.5)
        if (mo == 2 & of == 1)
            return(1)
    }
    if (fa == 1) {
        if (mo == 0 & (of == 0 | of == 1))
            return(0.5)
        if (mo == 1 & (of == 0 | of == 2))
            return(0.25)
        if (mo == 1 & of == 1)
```

```
            return(0.5)
        if (mo == 2 & (of == 1 | of == 2))
            return(0.5)
    }
    if (fa == 2) {
        if (mo == 0 & of == 1)
            return(1)
        if (mo == 1 & (of == 1 | of == 2))
            return(0.5)
        if (mo == 2 & of == 2)
            return(1)
    }
    return(0)
}
<environment: namespace:ElstonStewart>
```

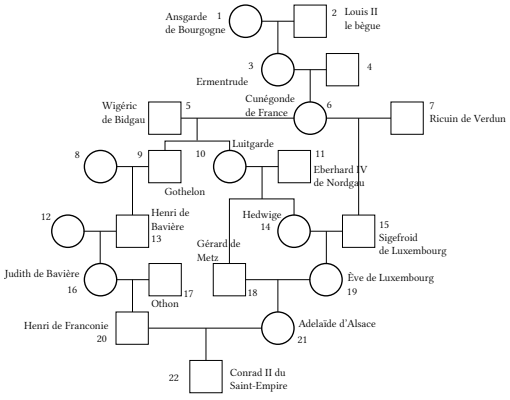
In this model, the phenotype will be ignored. More general functions can use any component of `theta`:

```
> modele.di$p.pheno

function (x, g, theta)
1
<environment: namespace:ElstonStewart>
```

3 Example: probabilities for an imbred pedigree

This is the pedigree of Conrad II, Holy Roman Emperor. The names on this pedigree appear in french.



Conrad has an imbreding coefficient $f = 1/64$. For example, his probability to have a recessive disease due to a mutation with frequency $q = 0.02$ is $(1 - f)q^2 + fq = 0.00070625$, to be compared with $q^2 = 0,0004$ for non-imbred individuals.

The data set `conrad2` provides the pedigree structure.

```
> data(conrad2)
> conrad2
```

```
   id father mother sex
1   1      0      0   2
2   2      0      0   1
3   3      2      1   2
4   4      0      0   1
5   5      0      0   1
6   6      4      3   2
7   7      0      0   1
8   8      0      0   2
9   9      5      6   1
10 10      5      6   2
11 11      0      0   1
12 12      0      0   2
13 13      9      8   1
14 14     11     10   2
15 15      7      6   1
16 16     13     12   2
17 17      0      0   1
18 18     11     10   1
19 19     15     14   2
20 20     17     16   1
21 21     18     19   2
22 22     20     21   1
```

3.1 A computation with `modele.di`

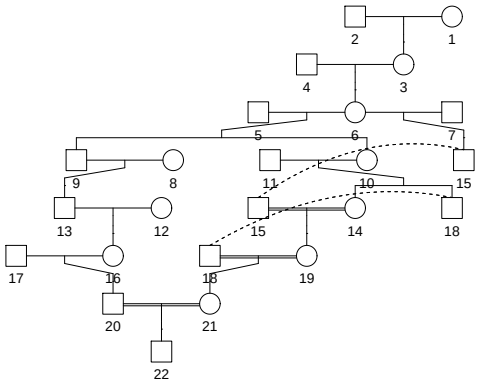
Creation of an `es.pedigree` object with genotype = 2 for Conrad, and 0, 1 or 2 for all other members of the family.

```
> genotypes <- c( rep(list(0:2), 21), 2 )
> X <- es.pedigree( id = conrad2$id, father = conrad2$father, mother = conrad2$mother,
+       sex = conrad2$sex, pheno = rep(0, 22), geno = genotypes )
> X
```

```
An es.pedigree object with 22 individuals
```

We can plot it.

```
> plot(X)
```



And we can compute the probability of this pedigree

```
> r <- Elston(X, modele.di, list(p = 0.98))
> r$result
```

```
[1] 0.00070625
```

The second component of `r` is an environment, containing intermediate results of the computation. Providing it to Elston as parameter `mem` will speed up subsequent computations, even with a different parameter p.

```
> # using the memoization...
> system.time(r <- Elston(X, modele.di, list(p = 0.98)))
```

```
   user  system elapsed
  0.939   0.000   0.941
```

```
> system.time(r <- Elston(X, modele.di, list(p = 0.98), r$mem))
```

```
   user  system elapsed
  0.001   0.000   0.001
```

```
> system.time(r <- Elston(X, modele.di, list(p = 0.99), r$mem))
```

```
   user  system elapsed
  0.636   0.007   0.646
```

3.2 A computation with a model for recessive diseases

We create a model for recessive traits:

```
> modele.rec <- list( name = "recessive", proba.g = modele.di$proba.g,
+    trans = modele.di$trans,
+    p.pheno = function(x, g, theta)
+        ifelse( is.na(x) | (x == 1 & g == 2) | (x == 0 & g < 2) , 1, 0)
+    )
```

Setting all genotypes to unknown, we can compute the probability for Conrad II to have the recessive phenotype:

```
> genotypes <- rep(list(0:2), 22)
> X <- es.pedigree( id = conrad2$id, father = conrad2$father, mother = conrad2$mother,
+       sex = conrad2$sex, pheno = c( rep(NA, 21), 1), geno = genotypes )
> r <- Elston(X, modele.rec, list(p = 0.98), r$mem)
> r$result

[1] 0.00070625
```

An other interesting result is the probability of disease for Conrad II, knowing that no other individuals in the pedigree is diseased:

```
> X <- es.pedigree( id = conrad2$id, father = conrad2$father, mother = conrad2$mother,
+       sex = conrad2$sex, pheno = c( rep(0, 21), 1), geno = genotypes )
> r <- Elston(X, modele.rec, list(p = 0.98), r$mem)
> r$result

[1] 0.0004563312
```

This could have been computed with `model.di` too, setting the possible genotypes to the appropriate values:

```
> genotypes <- c( rep(list(0:1), 21), 2 )
> X <- es.pedigree( id = conrad2$id, father = conrad2$father, mother = conrad2$mother,
+       sex = conrad2$sex, pheno = rep(0, 22), geno = genotypes )
> r <- Elston(X, modele.di, list(p = 0.98), r$mem)
> r$result

[1] 0.0004563312
```

4 Example: likelihood maximization for a set of pedigrees

The data frame `fams` contains 50 pedigrees. The genotypes for a di-allelic locus are known only for a subset of individuals.

```
> data(fams)
> head(fams,15)
```

```
   fam id father mother sex genotype
1    1  1      0      0   1       NA
2    1  2      0      0   2       NA
3    1  3      1      2   2       NA
4    1  4      0      0   1       NA
5    1  9      4      3   2        2
6    1 10      0      0   1       NA
7    1 11      4      3   2       NA
8    1 12      0      0   1        2
9    1 19     10      9   2        2
10   1 22     12     11   2        1
11   1 24     12     11   2        1
12   1 25      1      2   1       NA
13   1 26      0      0   2       NA
14   1 27     25     26   1        1
15   1 33     25     26   2        0
```

We will estimate the allele frequencies in this locus by likelihood maximization. We start by creating a list of `es.pedigree` objects.

```
> fam.ids <- unique(fams$fam);
> # creating a list of genotypes corresponding to individuals in fam.ids
> # genotype is NA -> 0, 1 or 2
> genotypes <- lapply( fams$genotype, function(x) if(is.na(x)) 0:2 else x )
> X <- vector("list", length(fam.ids))
> for(i in seq_along(fam.ids))
+ {
+   w <- which(fams$fam == fam.ids[i])
+   X[[i]] <- es.pedigree( id = fams$id[w], father = fams$father[w],
+       mother = fams$mother[w], sex = fams$sex[w], pheno = rep(0, length(w)),
+       geno = genotypes[w], famid = fam.ids[i] )
+ }
```

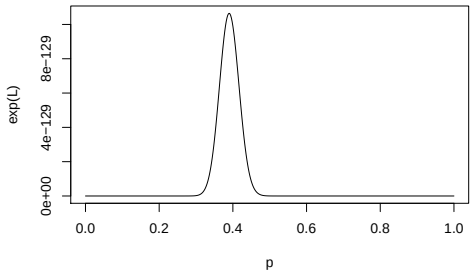
When we use the function `Likelihood`, we don't have to take care of memoization anymore.

```
> # computing the log-likelihood for a single value p
> Likelihood(X, modele.di, theta = list( p=0.5), n.cores=1 )
```

```
[1] -303.006
```

Vectorization is possible! With the code below, the algorithm is ran only once to compute the 501 log-likelihoods.

```
> # computing the likelihood for a vector p
> p <- seq(0,1,length=501)
> L <- Likelihood(X, modele.di, theta = list( p=p ), n.cores=1 )
> plot( p, exp(L), type="l")
```



We can run an optimization algorithm. Here Elston-Stewart is ran several times, on a computation cluster of 2 nodes. The cluster is created at first run, and left open for the next computations.

```
> # running an optimization algorithm
> # Elston-Stewart is ran several times
> # here we run the algorithm with 2 cores
> optimize( function(p) -Likelihood(X, modele.di, theta = list( p=p ), n.cores=2 ) , c(0.35,0.45) )

$minimum
[1] 0.389929

$objective
[1] 294.668
```

If you don't need it any more, close the cluster... (you will lose the memoization, which is specific to each cluster node).

```
> es.stopCluster()

stopping one cluster with 2 nodes
```