pedantics: Tools to facilitate power and sensitivity analyses for pedigree-based studies of natural popualitons

M. B. Morrissey

October 28, 2009

Contents

1 Introduction

pedantix contains three types of functions. The first are functions specifically designed to aid power and sensitivity analyses for quantitative genetic studies, particulary with thought to accommodating the proglems and data structures that arise in data from natural populations. Of these rpederr, fpederr, phensim, and microsim were described, albeit not as R functions, in [Morrissey et al., 2007] and [Morrissey and Wilson, in press]. Two functions for this class of use have since been added, rpederrBird, genomeSim. The second class of functions in pedantix are utilities for manipulating pedigrees. Some, such as makePedigreeNumeric and makePedigreeFactor are intended primarily to be used by other pedantics functions. More importantly, fixPedigree, is intended as a general function to manipulate pedigrees into a form that is required for many pedantics modules and other softwares often used in quantitative genetic analyses, such as asreml Gilmour et al. [2002] or the R package MCMCglmm [Hadfield and Kruuk, 2009]. The remaining functions, pedigreeStats and drawPedigree, are exploratory tools for summarizing and visualizing pedigrees. First we will explore the utility functions in pedantics, and then we will work through some sample power and sensitivity analyses, and finally I will provide a few pointers on setting up more advanced analyses.

All of the functions in pedantics include more advanced features than are described here. All features are described in the help files, accessable in the standard R manner using the function help().

2 Fixing pedigrees

The first thing to do before we can use pedantics is to load it:

```
library(pedantics)
```

Pedigrees of natural populations are commonly represented in forms that are not well handled by computer programs. The two most common problems are (1) that not all individuals in the pedigree are represented by their own records, and (2) very often we construct pedigrees in such a way that records of parents do not necessarily preceed records of offspring. Rectification of these problems is automated in fixPedigree(). Consider the pedigree:

```
pedigree<-as.data.frame(matrix(c(
  10,1,2,
  11,1,2,
  12,1,3,
  13,1,3,
  14,4,5,
  15,6,7,
  4,NA,NA,
  5,NA,NA,
  6,NA,NA,
  7,NA,NA),10,3,byrow=TRUE))
names(pedigree)<-c("id","dam","sire")
```

This is a form in which we might find a pedigree that has been constructed from observational or molecular data. While this is a valid pedigree structure, most computer programs will require that the maternal records preceed their offspring's records and that the paternal records get included in the pedigree, again before the first point at which those males appear as sires. We can accomplish this by:

```
> pedigree
   id dam sire
1  10   1    2
2  11   1    2
3  12   1    3
4  13   1    3
5  14   4    5
6  15   6    7
7   4  NA   NA
8   5  NA   NA
9   6  NA   NA
10  7  NA   NA
>
> fixedPedigree<-fixPedigree(Ped=pedigree)
>
> fixedPedigree
   id  dam sire
7   4 <NA> <NA>
8   5 <NA> <NA>
9   6 <NA> <NA>
10  7 <NA> <NA>
31  1 <NA> <NA>
61  2 <NA> <NA>
63  3 <NA> <NA>
1  10    1    2
2  11    1    2
3  12    1    3
4  13    1    3
5  14    4    5
6  15    6    7
>
```

This has resulted in our pedigree having a different length and order. Some phenotypic data may need to be used in conjunction with the new pedigree order. While this would not be a large problem in general in `R`, i.e., the function `match()` can be handy, any data that is optionally supplied to `fixPedigree()` will be returned in the new order, for example:

```
data<-as.data.frame(matrix(c(
  10,1,2.45,
  11,2,2.87,
  12,2,2.98,
  13,2,2.40,
  14,1,3.10,
  15,2,3.60,
  4,0,1.34,
  5,0,2.48,
  6,0,1.72,
  7,0,2.56),10,3,byrow=TRUE))
names(data)<-c("id","phenotype1","phenotype2")

> data
   id phenotype1 phenotype2
1  10          1       2.45
2  11          2       2.87
3  12          2       2.98
4  13          2       2.40
5  14          1       3.10
6  15          2       3.60
7   4          0       1.34
8   5          0       2.48
9   6          0       1.72
10  7          0       2.56
>
> fixedPedigree<-fixPedigree(Ped=pedigree, dat=data)
>
> fixedPedigree
   id  dam sire V4   V5
7   4 <NA> <NA>  0 1.34
8   5 <NA> <NA>  0 2.48
9   6 <NA> <NA>  0 1.72
10  7 <NA> <NA>  0 2.56
31  1 <NA> <NA> NA   NA
61  2 <NA> <NA> NA   NA
63  3 <NA> <NA> NA   NA
1  10    1    2  1 2.45
2  11    1    2  2 2.87
3  12    1    3  2 2.98
4  13    1    3  2 2.40
5  14    4    5  1 3.10
6  15    6    7  2 3.60
```

```
>
```

3 Visualizing pedigrees

Here we will begin to use the fictional example dataset that is distributed with `pedantics`, data on gryphons collected in the 12th century.

```
> data(gryphons)
> head(gryphons)
   id dam sire sex cohort fatherErrorProb fatherSampledProb firstReproCohort
1 204  NA   NA   1   1147               1               0.5             1148
2 205  NA   NA   1   1147               1               0.5             1148
3 206  NA   NA   0   1147               1               0.5             1148
4 207  NA   NA   0   1147               1               0.5             1148
5 208  NA   NA   1   1147               1               0.5             1148
6 209  NA   NA   1   1147               1               0.5             1148
>
```

The simplest visual interpretation of this pedigee can be provided as follows:

```
gryphPed<-as.data.frame(cbind(gryphons$id,gryphons$dam,
                                             gryphons$sire))
names(gryphPed)<-c("id","dam","sire")
drawPedigree(gryphPed)
```

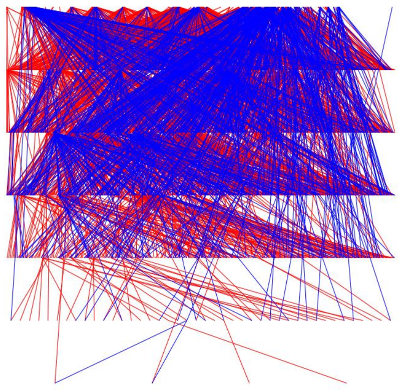
Here maternal links are in red and paternal links are in blue (Figure 1a). A more useful visual description of the pedigree can be provided by plotting by cohort (Figure 1b):

```
drawPedigree(gryphPed,cohorts=gryphons$cohort)
```

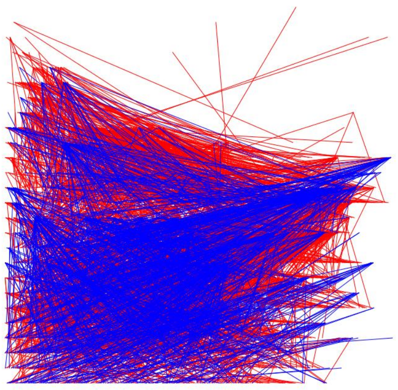
4 Pedigree summary statistics

A range of summary statistics for the gryphon pedigree can be generated as follows:

```
> pedStatSummary(pedigreeStats(gryphPed,graphicalReport='n'))
                  records             maternities
             4.918000e+03            2.435000e+03
              paternities               full sibs
             1.156000e+03            8.600000e+01
            maternal sibs      maternal half sibs
             6.957000e+03            6.871000e+03
            paternal sibs      paternal half sibs
             6.138000e+03            6.052000e+03
```



(a)



(b)

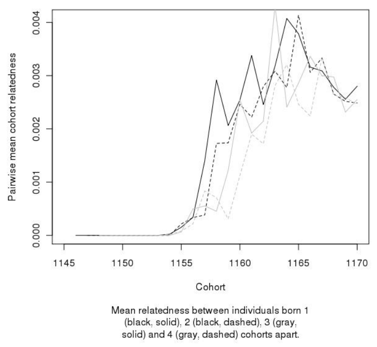
Figure 1: Sample pedigree images using `drawPedigree()`.

```
     maternal grandmothers       maternal grandfathers
              1.503000e+03                8.890000e+02
     paternal grandmothers       paternal grandfathers
              5.070000e+02                3.540000e+02
    maximum pedigree depth                    founders
              6.000000e+00                2.483000e+03
 mean maternal sibsip size   mean paternal sibsip size
              4.183849e+00                3.853333e+00
                non-zero F                   F > 0.125
              2.300000e+01                7.000000e+00
 mean pairwise relatedness pairwise relatedness>=0.125
              8.580691e-02                3.309669e-01
pairwise relatedness>=0.25   pairwise relatedness>=0.5
              1.272125e-01                2.350472e-02
```

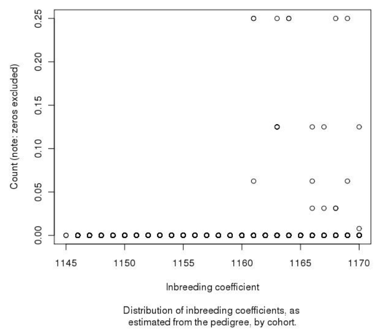
In many pedigree-based studies of natural populations, attributes of pedigrees vary in important ways over time. Providing cohort affinities to `pedigreeStats` allows it to provide more informative data, especially in graphical form:

```
stats2<-pedigreeStats(gryphPed,cohorts=gryphons$cohort)
```

The above code generates a number of graphical representations of different aspects of the pedigree, including those in Figure 2.



(a)



(b)

Figure 2: Samples of the graphical output of pedigree statistics generated by `pedigreeStats()`.

5 Generating 'plausible true' and assumed pedigrees

When pedigrees are uncertain, an important step in power and sensitivity analyses can be the generation of pairs of pedigrees. One pedigree of the pair mimics a true pedigree based on the true breeding system of the organism in question and the other pedigree differs from the first in a way that mimics patterns of pedigree uncertainty in any given study. `pedantics` contains several functions to generate these pairs of pedigrees. The function `rpederr` is most useful for the gryphon pedigree, because with a lot of pedigree data have already been collected, so generation of the plausible true pedigree can be done based on the uncertain pedigree, thus avoiding the need to simulate an entire pedigree from scratch:

```
gryphons<-gryphons[order(gryphons$cohort),]
gryphPed<-as.data.frame(cbind(gryphons$id,gryphons$dam,
                                               gryphons$sire))
names(gryphPed)<-c("id","dam","sire")

fullPedigree<-rpederr(gryphPed,founders=(is.na(gryphons$dam))+0,
             sex=gryphons$sex,sireE=gryphons$fatherErrorProb,
             cohort=gryphons$cohort,first=gryphons$firstReproCohort,
             last=gryphons$lastReproCohort)$truePedigree
```

This particular use of `rpederr` would only have filled in some of the missing paternal links, since all individuals without maternal links were designated as founders (many other more realistic assumptions are possible with `rpederr`). We can check that it worked right be making sure that some paternal links did get filled in:

```
> table(is.na(gryphPed$sire))

FALSE  TRUE
 1156  3762
>
> table(is.na(fullPedigree$sire))

FALSE  TRUE
 2435  2483
>
```

6 Generating simulated phenotypic data

The next step in work flows for many power and sensitivity analyses will be the generation of data based on the plausible true pedigree. Here we can simulate data for a single trait with a heritability of 0.5:

```
simPhen<-phensim(fullPedigree,randomA=0.5,randomE=0.5)$phenotypes
```

We can check that this worked out by regressing offspring phenotypes on their parents. In this case, we'll calculate the mother-offspring covariance, which should equal to half the additive genetic variance [Lynch and Walsh, 1998]:

```
simPhen<-phensim(fullPedigree,randomA=0.5,randomE=0.5)$phenotypes
simPhen$mum<-gryphons$dam[match(simPhen$id,gryphons$id)]
simPhen$mumPhenotype<-simPhen$trait_1[
                match(simPhen$mum,as.numeric(as.character(simPhen$id)))]
>
> cov(simPhen$mumPhenotype,simPhen$trait_1,use="pairwise.complete.obs")
[1] 0.2314591
```

7 A power analysis

Normally, analysis of a single simulated dataset, as above, will not be particularly informative. Typically we need to repeat analyses of each of a number of scenarios a large number of times before we can begin to make inferences on issues of power and sensitivity. An analysis of the power to detect segregating genetic variation using mother-offspring regression in the gryphon data set might proceed as follows:

```
## we'll do plenty of replicates of each scenario:
n<-1000
```

```
## here is a decent range of heritabilities to simulate:
heritabilities<-seq(0,0.2,by=0.02)

## we will need to keep track of the p-values resulting
## from each replicate of each scenario, and for each
## scenario, we'll want to keep track of the proportion
## of significant tests:
p_values<-matrix(NA,n,length(heritabilities))
power<-array(dim=length(heritabilities))

## here we'll loop through all the scenarios and replicates,
## keeping track of the p-value at the end of each iteration
## through the work flow, and stopping to calculate the
## proportion of significant tests each time we get through
## all the replicates for each heritability
for(x in 1:length(heritabilities)) {
  for(y in 1:n) {
    simPhen<-phensim(fullPedigree,randomA=heritabilities[x],
                          randomE=1-heritabilities[x])$phenotypes
    simPhen$mum<-gryphons$dam[match(simPhen$id,gryphons$id)]
    simPhen$mumPhenotype<-simPhen$trait_1[match(simPhen$mum,
                            as.numeric(as.character(simPhen$id)))]
    p_values[y,x]<-anova(lm(simPhen$trait_1~simPhen$mumPhenotype))[1,5]
  }
  power[x]<-table(p_values[,x]<0.05)["TRUE"]/n
}
```

And we can view the power curve in tabular form:

```
> power_results<-cbind(heritabilities,power)
> power_results
      heritabilities power
 [1,]           0.00 0.039
 [2,]           0.02 0.069
 [3,]           0.04 0.172
 [4,]           0.06 0.313
 [5,]           0.08 0.497
 [6,]           0.10 0.664
 [7,]           0.12 0.828
 [8,]           0.14 0.900
 [9,]           0.16 0.949
[10,]           0.18 0.986
[11,]           0.20 0.996
>
```

8 Making `pedantics` talk to non-R software

One of the difficulties of setting up an analysis with `pedantics` is that several steps are required. Often, not all of these steps can yet be conduced directly in `R`. The above sensitivity analysis demonstrates a typical work flow, but often the step where simulated data are analysed will require a non-`R` software. In quantitative genetic analyses, this might involve using ASREML (Gilmour et al. 2002; note that an `R` version is available, but lacks significant functionality relative to the stand alone version) or WOMBAT Meyer [2006].

Use of an external program in an `R`-based `pedantics` analysis is possible, and requires three operations: (1) simulated data must be made available to the external program, (2) the external program must be called, and (3) output from the external program must be read back into `R`.

The first step is easily accomplished using `write.table()`. For example, we could write the pedigree and data files required by ASREML, for the sensitivity analysis we conducted by mother-offspring regression as follows:

```
## write a pedigree file
write.table(fullPedigree,"pedigree_for_asreml.ped")
## write a data file file
write.table(simPhen,"phens_for_asreml.ped")
```

If `R` had been invoked from the `../asreml/bin` folder, these files would now be in place to conduct an asreml analysis. We can do this without having to leave the `R` environment using the `system()` call function. Assuming an appropriate ASREML program file (and post-processing file - the commands for the second step of most ASREML analyses where variance components and their standard errors are produced) existed in the `../asreml/bin` folder, it could be run as follows:

```
system("./asreml gryphTest.as")
system("./asreml gryphTest.pin")
```

This would generate an ouput file `gryphTest.pvc` that might look like the following:

```
   3 phenvar        1.023      0.0213
   2 addvar         0.566      0.1224
     h2           = addvar    /phenvar    =         0.550  0.145
```

The trickiest step is extraction of this information from the datafile, and getting it back into `R`. This is particularly difficult because the file is not arranged as a table. For the ASREML results above, the variance components and their standard errors can be recovered into `R` as follows:

```
inp<-readLines("./gryphTest.pvc")
get_vals<-function(lineData){
```

```
## this function reduces text and numbers separated by
## variable numbers of spaces into a vector of just
## the numeric values:
  suppressWarnings(subset(as.numeric(unlist(
                strsplitlineData[1]," "))),is.na(as.numeric(
                unlist(strsplit(lineData[1]," "))))==FALSE))
}
phenVar<-get_vals(inp[1])[2]
phenVarSE<-get_vals(inp[1])[3]
additiveVar<-get_vals(inp[2])[2]
additiveVarSE<-get_vals(inp[2])[3]
heritability<-get_vals(inp[3])[1]
heritabilitySE<-get_vals(inp[3])[2]
```

We can check that we recovered the right values:

```
> phenVar
[1] 1.023
> phenVarSE
[1] 0.0213
>
> additiveVar
[1] 0.566
> additiveVarSE
[1] 0.1224
>
> heritability
[1] 0.055
> heritabilitySE
[1] 0.145
>
```

This set of text handling `R` functions is described in Crawley [2007], p. 105. Note that some adjustement will be required for the particular locations where different results will occur in ASREML `.pvc` files generated for different kinds of analyses. Note that the positions of important data in the decomposed string from each line of the `.pvc` can change: in the above example, the genetic and phenotypic variances and their standard errors are the second and third numeric values on their respective lines, while the estimates and standard errors of the heritability occur in the first and second numeric positions of the heritability line.

9 Further processing of simulated data

In the following sections, I provide a few pointers on how to take advantage of the flexibility provided by the implementation of `pedantics` in `R` to simulate complexities in phenotypic data, beyond those that are immediately provided by the modules in `pedantics`.

9.1 Fixed effects

Suppose some issue in the analysis of the gryphon data required a simulation study where birth year effects on mean phenotype were important. Year effects taken from a normal distribution with variance 0.5 units could be added to the phenotypic data we simulated in section 6 as follows

```
## simulate some effects associated with each cohort
cohort_effects<-as.data.frame(cbind(as.numeric(
                names(table(gryphons$cohort))),
                rnorm(length(table(gryphons$cohort)),
                0,sqrt(0.5))))

## associate cohort effects with individuals
simPhen$cohort_effect<-cohort_effects[match(
                gryphons$cohort,cohort_effects[,1]),2]

## re-calculate phenotype
simPhen$measured_value<-simPhen$trait_1+simPhen$cohort_effect
```

9.2 Repeated records

Multiple records are quite common. Given the new implementation of `pednatics` in `R`, the simulation of a variety of scenarios of repeated records data, including random-regression is possible. One scenario might include a trait measured several times, say anually, with declining data availability (due to mortality). A trait measured up to four times, for individuals with an annual survival rate of 0.5 might be simulated as follows. We'll use the trait above into which we incorporated a cohort effect to simulate these data. Additionally, we will simulate an additional environmental variance of 0.2 for between-year, within-individual variation:

```
## simulate with no missing data
## simulate with no missing data
simPhen$year1<-simPhen$measured_value+
              rnorm(length(simPhen[,1]),0,sqrt(0.2))
simPhen$year2<-simPhen$measured_value+
              rnorm(length(simPhen[,1]),0,sqrt(0.2))
simPhen$year3<-simPhen$measured_value+
              rnorm(length(simPhen[,1]),0,sqrt(0.2))
simPhen$year4<-simPhen$measured_value+
              rnorm(length(simPhen[,1]),0,sqrt(0.2))
```

```
## simulate missing data, remembering that a dead
## individual can't be measured in subsequent years
simPhen$year1<-simPhen$year1*
              rbinom(length(simPhen[,1]),1,0.5)
simPhen$year2<-simPhen$year2*
              rbinom(length(simPhen[,1]),1,0.5)*
              ((simPhen$year1!=0)+0)
simPhen$year3<-simPhen$year3*
               rbinom(length(simPhen[,1]),1,0.5)*
              ((simPhen$year2!=0)+0)
simPhen$year4<-simPhen$year4*
              rbinom(length(simPhen[,1]),1,0.5)*
              ((simPhen$year3!=0)+0)

## change the zeros that we now have for missing data to NAs
simPhen$year1[simPhen$year1 %in% 0]<-NA
simPhen$year2[simPhen$year2 %in% 0]<-NA
simPhen$year3[simPhen$year3 %in% 0]<-NA
simPhen$year4[simPhen$year4 %in% 0]<-NA
```

And to check a portion of the data:

```
> simPhen[1:10,c("id","year1","year2","year3","year4")]
    id       year1      year2      year3      year4
1   91          NA         NA         NA         NA
2   92  0.74858933  0.1685677  0.7500114         NA
3   93  0.95772673         NA         NA         NA
4   94          NA         NA         NA         NA
5   95          NA         NA         NA         NA
6   96          NA         NA         NA         NA
7   97 -0.87097848 -0.9690001 -0.6132075 -1.0726285
8   98  0.12817738  0.4200899 -1.0273101         NA
9   99          NA         NA         NA         NA
10 100 -0.03473497 -0.6074088 -0.7519217  0.0485704
>
```

9.3 Non-normal data

Non-normal data distributions (e.g. binary, binomial, exponential, poisson) can be produced by treating the normal deviates generated by `pedantics` as an underlying liability. In such an approach the non-normal data can be generated using a link function. For example, a lot of ecological data are binary, (e.g. reproduced/failed) and such data can be created using the inverse logit function and binomial deviates with one trial:

```
## note that we are leaving the data centered on zero, as
## simulated.  To change the population mean, we would
## need to add a constant to the liability
simPhen$year1<-suppressWarnings(rbinom(length(
              simPhen[,1]),1,inv.logit(simPhen$year1)))
simPhen$year2<-suppressWarnings(rbinom(length(
              simPhen[,1]),1,inv.logit(simPhen$year2)))
simPhen$year3<-suppressWarnings(rbinom(length(
              simPhen[,1]),1,inv.logit(simPhen$year3)))
simPhen$year4<-suppressWarnings(rbinom(length(
              simPhen[,1]),1,inv.logit(simPhen$year4)))
```

And to check a portion of the data:

```
> simPhen[1:10,c("id","year1","year2","year3","year4")]
    id year1 year2 year3 year4
1   91   NaN   NaN   NaN   NaN
2   92     0     1     0   NaN
3   93     1   NaN   NaN   NaN
4   94   NaN   NaN   NaN   NaN
5   95   NaN   NaN   NaN   NaN
6   96   NaN   NaN   NaN   NaN
7   97     0     0     0     0
8   98     1     1     0   NaN
9   99   NaN   NaN   NaN   NaN
10 100     0     0     0     0
>
```

10 Acknowledgements

pedantics has been developed with input from Alastair Wilson, Jarrod Hadfield, Josephine Pemberton, Loeske Kruuk, Moira Ferguson, Denis Reale, and Jane Reid. Jarrod Hadfield helped enormously in preparing pedantics in the form of an R package.

References

M.J. Crawley. *The R book.* Wiley, West Sussex, England, 2007.

A. R. Gilmour, B. J. Gogel, B. R. Cullis, S. J. Welham, and R. Thompson. *ASReml user guide release 1.0.* VSN International Ltd, Hemel Hempstead, United Kingdom, 2002.

J.D. Hadfield and L. E. B. Kruuk. MCMC methods for multi-response generalised linear mixed models: The MCMCglmm R package. submitted, 2009.

M. Lynch and B. Walsh. *Genetics and analysis of quantitative traits.* Sinauer, Sunderland, MA, 1998.

K. Meyer. WOMBAT - digging deep for quantitative genetic analyses by restricted maximum likelihood. In *8th World Congress on Genetics Applied to Livestock Production, August 13-18, Belo Horizonte, Brazil*, 2006.

M. B. Morrissey, A. J. Wilson, J. M. Pemberton, and M. M. Ferguson. A framework for power and sensitivity analyses for quantitative genetic studies of natural populations and case studies in soay sheep (*Ovis aries*). *Journal of Evolutionary Biology*, 20:2309–2321, 2007.

M.B. Morrissey and A.J. Wilson. `pedantics`, an `R` package for pedgiree-based genetic simulation and pedigree manipulation, characterization, and viewing. *Molecular Ecology Resources*, in press.