Family Based Association Tests Using the fbat package

Weiliang Qiu
email: stwxq@channing.harvard.edu
Ross Lazarus
email: ross.lazarus@channing.harvard.edu
Gregory Warnes
email: warnes@bst.rochester.edu
Nitin Jain
email: nitin.jain@pfizer.com

April 22, 2010

Contents

1 Introduction

The `R` package `fbat` can be used to test the following null hypotheses for each marker based on family pedigrees:

H_{01} : the marker has no association and no linkage with the trait;

H_{02} : the marker has no association with the trait in the presence of linkage.

We assume that

- the families are **nuclear** families
- there are no missing genotypes and phenotypes for children
- markers are bi-allelic.

A more general software `FBAT` is available as a stand-alone executable with documentation and example files from `http://www.biostat.harvard.edu/~fbat/fbat.htm`. While this `R` package has some important limitations as present, these will be addressed in further versions.

2 Pedigree data file format

All fields are separated by whitespace (e.g. one or more spaces).

First line : names of all markers in the sequence of the genotype data. For example, marker_1, marker_2, ..., marker_m.

Remaining lines: The remaining lines contain only non-negative integers and have the same format:

family	pid	father	mother	sex	affection	$\text{marker}_{1.1}$	$\text{marker}_{1.2}$	$\cdots$	$\text{marker}_{m.1}$	$\text{marker}_{m.2}$

where

family: family id

pid: patient id

father: father id.

Use 0 (zero) for founders or marry-ins (parents not specified) in a pedigree. A **founder** in a pedigree is an individual who is not a child of any individuals in the pedigree.

mother: mother id.

Use 0 (zero) for founders or marry-ins (parents not specified) in a pedigree. A **founder** in a pedigree is an individual who is not a child of any individuals in the pedigree.

sex: 1 – male; 2 – female;

affection: affection status (i.e., trait)

2 – affected; 1 – unaffected; 0 – unknown

marker$_{i.j}$**:** allele j of marker i, $j = 1, 2$; $i = 1, 2, \ldots, m$.

non-missing Alleles are represented by positive integers. Missing alleles are represented by zero (0).

3 Data quality control

The `R` package fbat also provides some basic QC functions.

The function `missGFreq` checks the completeness of genotypes. This function outputs counts of missing genotypes per marker and per subject.

The function `pedHardyWeinberg` checks the assumption of the Hardy-Weinberg equilibrium for markers.

The function `checkMendelian` checks the following possible Mendelian-related errors:

1. father id = subject id;
2. mother id = subject id;
3. could not determine if an individual is a parent or a child in a family;
4. inconsistent parental sex in a family;
5. parental genotypes are not compatible with childrens' genotypes in a family;
6. all childrens' genotypes are missing in a family;
7. inconsistent sib genotypes in a family.

4 Examples

To call the functions in the `R` package fbat, we first need to load it into `R`:

To read the pedigree file `CAMP.ped` into `R`, we use the function `readGenes` in the `R` package GeneticsBase:

```
gSet<-readGenes(gfile="CAMP.ped", gformat="fbat")
```

The function `readGenes.ped` returns back an object of the `R` class *geneSet*.

Before we apply family based association tests, it would be good practice to check Hardy-Weinberg equilibrium for each marker based on parental data. We can use the function `pedHardyWeinberg` to do this.

```
> data(CAMP)

Reading 8 markers and 2011 subjects from ` CAMP.ped ' ...
generating 'geneSet' object...

Successfully read the pedigree file ` CAMP.ped '.

Number of Markers:  8
Number of Subjects: 2011
Number of Families: 651


Reading 12 vars from `CAMPZ.phe' ... Done.

Number of Phenotype Variables: 12
Number of Observations       : 2011

> ch <- pedHardyWeinberg(CAMP)

converting geneSet object to numerical matrix...
HWE test...
     nInfoInd nGenotype nHET nHOM nAllele nMissing  chi2 df p-value
m709     1269         3    4 1265       2       34 0.003  1   0.955
m654     1259         3  558  701       2       44 1.080  1   0.299
m47      1244         3  582  662       2       59 0.005  1   0.944
p46      1253         3  596  657       2       50 0.029  1   0.864
p79      1244         3  580  664       2       59 0.030  1   0.862
p252     1184         3  410  774       2      119 0.685  1   0.408
p491     1263         3   27 1236       2       40 0.147  1   0.701
p523     1271         3  405  866       2       32 0.082  1   0.775
```

The column `nInfoInd` means the number of informative individuals, i.e. individuals whose genotypes contain no missing alleles for the specified marker; the column `nGenotype` means number of possible genotypes; the column `nHET` means number of heterozygous genotypes; the column `nHOM` means number of homozygous genotypes; the column `nAllele` means number of alleles; the column `nMissing` means number of missing alleles; the column `chi2` means chi square test statistic; the column means `df` means degree of freedom of the chi square test statistic under the null hypothesis that Hardy-Weinberg condition holds; and the column `p-value` means pvalue of the test.

To view the statistics for individual markers, we can use the function `viewHW`. For example,

```
> viewHW(ch, "p79")

number of possible genotypes for marker  p79  >>
[1] 3
genotype frequency >>
     p79.1 p79.2 freq
[1,]     1     1  488
[2,]     1     2  580
[3,]     2     2  176
allele frequency >>
    1     2
0.625 0.375
 nInfoInd nGenotype      nHET      nHOM   nAllele  nMissing      chi2        df
 1244.000     3.000   580.000   664.000     2.000    59.000     0.030     1.000
  p-value
    0.862
```

To check Mendelian-realted errors, we can use the function `checkMendelian`. For example,

```
> tmp <- checkMendelian(CAMP, quiet = TRUE)
> cat("For each marker, how many families contains mendelian errors?\n")

For each marker, how many families contains mendelian errors?

> print(tmp$nMerrMarker)

m709 m654  m47  p46  p79 p252 p491 p523
  19  129  128  131  134  140   25  101

> cat("For each family, how many markers contains mendelian errors?\n")

For each family, how many markers contains mendelian errors?

> cat("tmp$nMerrFamily[1:10]>>\n")

tmp$nMerrFamily[1:10]>>

> print(tmp$nMerrFamily[1:10])

 family1  family2  family3  family4  family5  family6  family7  family8
       0        1        0        0        1        0        0        0
 family9 family10
       0        0
```

```
> cat("For each family, how many times\n")

For each family, how many times

> cat("'father id = subject id' or 'mother id = subejct id'?\n")

'father id = subject id' or 'mother id = subejct id'?

> cat("tmp$nErrFamilySample[1:10]>>\n")

tmp$nErrFamilySample[1:10]>>

> print(tmp$nErrFamilySample[1:10])

 family1  family2  family3  family4  family5  family6  family7  family8
       0        0        0        0        0        0        0        0
 family9 family10
       0        0
```

To count the number of missing genotypes for a marker or for a subject, we can use the function `missGFreq`. For example,

```
> res <- missGFreq(CAMP, founderOnly = FALSE, quiet = TRUE)
> cat("The number of missing genotypes for markers>>")

The number of missing genotypes for markers>>

> print(res$nMissMarkers)

      00 0* *0
m709  55  0  0
m654  60  0  0
m47   89  0  0
p46   68  0  0
p79   90  0  0
p252 188  0  0
p491  57  0  0
p523  53  0  0

> cat("The number of missing genotypes for the first 10 subjects>>")

The number of missing genotypes for the first 10 subjects>>

> print(res$nMissSubjects[1:10, ])
```

```
          00 0* *0
subject1   0  0  0
subject2   0  0  0
subject3   0  0  0
subject4   0  0  0
subject5   0  0  0
subject6   1  0  0
subject7   0  0  0
subject8   0  0  0
subject9   0  0  0
subject10  0  0  0
```

To get the family based association test statistics, we use the function `fbat`:

```
> res <- fbat(CAMP)
```

The usage of the function `fbat` is

```
fbat(geneSetObject, model="a", traitMethod=3, traitOffset=0, quiet=TRUE)
```

The function argument `model` specifies the genotype codings.

By default, we use the additive model (`model`="a"). Other available models include dominant (`model`="d"), recessive (`model`="r"), and genotype (`model`="g") models.

The function argument `traitMethod` indicates the trait coding method. If `traitMethod` is equal to 1, then the trait is represented by `trait-offset` where `trait` is the sixth column (i.e., `affection` status) of the pedigree matrix and the value of `offset` is provided by the argument `traitOffset`. If the argument `traitMethod` takes value other than 1, then the trait is set to be 1 if the sixth column of the pedigree matrix takes value 2 and the trait is set to be 0 if the sixth column of the pedigree matrix takes value 1.

The function `fbat` returns a list. To summarize the values, degrees of freedom, and p-values of the test statistics for the markers, we can use the function `summaryPvalue`:

```
> summaryPvalue(res)
```

```
****************************
          chisq rank       pvalue
m709  1.0000000    1 3.173105e-01
m654  1.3677298    1 2.422023e-01
m47  16.5161290    1 4.823799e-05
p46   0.1130742    1 7.366710e-01
p79  11.1838235    1 8.251356e-04
p252 37.7790698    1 7.922726e-10
p491 18.2413793    1 1.946047e-05
p523 45.7821782    1 1.321609e-11
****************************
```

To adjust multiple comparisons, we can use the function `p.adjust` in the R package base to adjust the p-values. For example,

```
> pvals <- res$statPvalue[, 3]
> p.adjust.M <- p.adjust.methods
> p.adj <- sapply(p.adjust.M, function(meth) p.adjust(pvals, meth))
> noquote(apply(p.adj, 2, format.pval, digits = 3))
```

```
     holm     hochberg hommel   bonferroni BH       BY       fdr      none
[1,] 0.726607 0.634621 0.634621 1.000000   0.36264  0.985605 0.36264  0.317311
[2,] 0.726607 0.634621 0.484405 1.000000   0.32294  0.877695 0.32294  0.242202
[3,] 0.000241 0.000241 0.000241 0.000386   9.65e-05 0.000262 9.65e-05 4.82e-05
[4,] 0.736671 0.736671 0.736671 1.000000   0.73667  1.000000 0.73667  0.736671
[5,] 0.003301 0.003301 0.003301 0.006601   0.00132  0.003588 0.00132  0.000825
[6,] 5.55e-09 5.55e-09 5.55e-09 6.34e-09   3.17e-09 8.61e-09 3.17e-09 7.92e-10
[7,] 0.000117 0.000117 0.000117 0.000156   5.19e-05 0.000141 5.19e-05 1.95e-05
[8,] 1.06e-10 1.06e-10 1.06e-10 1.06e-10   1.06e-10 2.87e-10 1.06e-10 1.32e-11
```

To view summary statistics of individual marker, we can use the function `viewstat`. For example,

```
> viewstat(res, "p79")
```

```
****************************************
651  pedigree  2011  persons
359  informative families at marker  p79
The alleles of marker  p79  >>
[1] 1 2
Score for marker  p79  >>
[1] 471 301
Expected score for marker  p79  >>
[1] 432 340
Covariance matrix of the score for marker  p79  >>
     [,1] [,2]
[1,]  136 -136
[2,] -136  136
Moore-Penrose generalized inverse of covariance matrix
             [,1]         [,2]
[1,]  0.001838235 -0.001838235
[2,] -0.001838235  0.001838235
test statistics for marker  p79  >>
       chisq         rank       pvalue
1.118382e+01 1.000000e+00 8.251356e-04
****************************************
```

Note that if the covariance matrix of the S score vector is singular, the Moore-Penrose generalized inverse is used.

Sometimes the user might want to know if a genotype a homozygous or heterozygous. The function pedFlagHomo can provide those information. For example,

```
> res.f <- pedFlagHomo(CAMP)

converting geneSet object to numerical matrix...
flag homozygotes and heterozygotes...
dim(flagHomoMat)= 1303 8
length(ped[,2])= 1303
numHomo -- number of homozygous genotypes
numHetero -- number of homozygous genotypes
numMiss1 -- number of genotypes containing one missing allele
numMiss2 -- number of genotypes containing two missing alleles
counts>>>
     numHomo numHetero numMiss1 numMiss2
m709    1265         4        0       34
m654     701       558        0       44
m47      662       582        0       59
p46      657       596        0       50
p79      664       580        0       59
p252     774       410        0      119
p491    1236        27        0       40
p523     866       405        0       32
```

The function pedGFreq gets genotype frequencies and percentages. For example,

```
> res <- pedGFreq(CAMP)

converting geneSet object to numerical matrix...
counting genotype frequencies...
genotype counts>>>
      1/1 1/2 2/2
m709 1265   4   0
m654  536 558 165
m47   171 582 491
p46   197 596 460
p79   488 580 176
p252   68 410 706
p491 1236  27   0
p523  813 405  53
```

The function pedAFreq gets allele frequencies and percentages. For example,

```
> res <- pedAFreq(CAMP)

converting geneSet object to numerical matrix...
count allele frequencies...
allele frequencies and percentages>>>
        1    2     1     2
m709 2534    4 0.998 0.002
m654 1630  888 0.647 0.353
m47   924 1564 0.371 0.629
p46   990 1516 0.395 0.605
p79  1556  932 0.625 0.375
p252  546 1822 0.231 0.769
p491 2499   27 0.989 0.011
p523 2031  511 0.799 0.201
```

The functions `fbat`, `pedHardyWeinberg`, `pedFlagHomo`, `pedGFreq`, and `pedAFreq` have default forms (`fbat.default`, `pedHardyWeinberg.default`, `pedFlagHomo.default`, `pedGFreq.default`, and `pedAFreq.default`) that use a pedigree matrix as input.

Appendix

A Notation

For a given marker,

- Y_{ij} — Observed trait of the j-th offspring in family i.
- T_{ij} — A function of Y_{ij}.
$$T_{ij} = T(Y_{ij}).$$
For example
$$T_{ij} = T(Y_{ij}) = Y_{ij} - \mu_{ij},$$
where μ_{ij} is an offset.
- g_{ij} — Genotype of the j-th offspring in family i;
- X_{ij} — A function of g_{ij}.
$$X_{ij} = X(g_{ij}).$$
- S score:
$$S = \sum_{ij} T_{ij} X_{ij} = \sum_{ij} T(Y_{ij}) X(g_{ij}).$$

- test statistic:

$$U = S - \mathrm{E}[S|H_0, \mathcal{C}],$$

where $\mathcal{C}$ is a condition set. When parental genotypes are complete, the condition set $\mathcal{C} = \mathcal{T} \cup \mathcal{G}$, where $\mathcal{T}$ is the observed traits in all family members and $\mathcal{G}$ is the parental genotypes. When parental genotypes are incomplete, the condition set $\mathcal{C} = \mathcal{T} \cup \mathcal{G}^* \cup \mathcal{G}_{\text{offspring}}$, $\mathcal{G}^*$ is the partially observed parental genotypes and $\mathcal{G}_{\text{offspring}}$ is the set of offspring genotypes (i.e., the offspring genotype configuration).

- V – variance or covariance matrix of U under the null hypothesis H_0. I.e.,

$$V = \mathrm{Cov}(U|H_0, \mathcal{C}) = \mathrm{Cov}\,(S|H_0, \mathcal{C}).$$

- For the univariate case,

$$Z = \frac{U}{\sqrt{V}}\Bigg| H_0, \mathcal{C} \dot{\rightarrow} \mathrm{N}\,(0, 1).$$

- For the multivariate case,

$$\chi^2 = U'V^{-1}U\Big| H_0, \mathcal{C} \dot{\rightarrow} \chi^2_r,$$

where $r = \mathrm{rank}\,(V)$.

B Genotype coding methods

Denote K as the number of all possible different alleles for the locus and X as the vector of genotype coding.

GEN X is a vector with length equal to the number of genotypes that are possible given the parental genotypes in the sample, a maximum of $K(K+1)/2$ genotypes, and with elements equal to 1 or 0 to indicate which of the possible genotypes is equal to the genotype g.

GDOM codes the jth element of the vector X as $x_j = 1$ if genotype g has one or two alleles of type j, otherwise $x_j = 0$. X is a vector of length K.

GREC codes the jth element of the vector X as $x_j = 1$ if genotype g has two alleles of type j, otherwise $x_j = 0$. X is a vector of length K.

GTDT scores the number of alleles of a particular type by coding x_j equal to the number of alleles of type j in the genotype g (i.e., $x_j = 0, 1$, or 2 if g has 0, 1 or 2 alleles of type j). X is a vector of length K.

2-allele case

Example of different marker codings for a marker with $K = 2$ alleles, see Schaid (1996)

genotype	$X(g)$			
g	GEN	GDOM	GREC	GTDT
		(A, a)	(A, a)	(A, a)
AA	(0,0,0)	(1,0)	(1,0)	(2,0)
Aa	(1,0,0)	(1,1)	(0,0)	(1,1)
aa	(0,1,0)	(0,1)	(0,1)	(0,2)

3-allele case

Example of different marker codings for a marker with $K = 3$ alleles, see Schaid (1996) (This table is Table 4 of Horvath et al.'s report for FBAT software)

genotype	$X(g)$			
g	GEN	GDOM	GREC	GTDT
		(A, B, C)	(A, B, C)	(A, B, C)
AA	(0,0,0,0,0)	(1,0,0)	(1,0,0)	(2,0,0)
AB	(1,0,0,0,0)	(1,1,0)	(0,0,0)	(1,1,0)
AC	(0,1,0,0,0)	(1,0,1)	(0,0,0)	(1,0,1)
BB	(0,0,1,0,0)	(0,1,0)	(0,1,0)	(0,2,0)
BC	(0,0,0,1,0)	(0,1,1)	(0,0,0)	(0,1,1)
CC	(0,0,0,0,1)	(0,0,1)	(0,0,1)	(0,0,2)

C Trait coding methods

Denote Y_{ij} as the trait of the j-th child of the ith nuclear family. Y_{ij} can be dichotomous, measured (i.e., continuous?), time-to-onset (i.e., censored?)

The trait coding methods ($T_{ij} = T(Y_{ij})$) are listed below:

- $T_{ij} = 1$ if the jth child is affected; $T_{ij} = 0$ otherwise.
- $T_{ij} = Y_{ij} - \mu_{ij}$, where μ_{ij} is an offset.
- $T_{ij} = Y_{ij} - \mu_{ij}(\boldsymbol{x}'\boldsymbol{\beta})$, where $\mathrm{E}(Y_{ij}|\boldsymbol{x}) = \mu_{ij}(\boldsymbol{x}'\boldsymbol{\beta})$, and $\boldsymbol{x}$ are design matrix of covariates, $\boldsymbol{\beta}$ are unknown parameters.