BlueSNP Tutorial

Robert J. Prill, Sandeep Tata, Hailiang Huang

November 1, 2012

Contents

1 Preliminaries

1.1 Software Version

```
BlueSNP 0.1.0
```

1.2 Installation

- This document picks-up where the installation instructions in the BlueSNP manual leave-off.
- **Before proceeding you must install the BlueSNP package and it's dependencies.**
- See the **BlueSNP Manual** for installation instructions.

1.3 Download the Tutorial sample data

Log into the Rhipe VM running on port 2222, user:username, pass:password

```
$ ssh -p 2222 localhost -l username
```

Download the tutorial materials using wget

```
$ wget https://github.com/downloads/ibm-bioinformatics/BlueSNP/tutorial_data_v1.tgz
```

Unzip

```
$ tar xvzf tutorial_data.tgz
```

1.4 Copy the tutorial files to the Hadoop filesystem

Verify that the HDFS directory /user/username exists. (This should have been created at the time of BlueSNP package installation and testing.)

```
$ hadoop fs -ls /user/username
```

If if does not exist, create it.

```
$ hadoop fs -mkdir /user/username
```

Copy the tutorial directory from the local filesystem to the HDFS.

```
$ hadoop fs -copyFromLocal tutorial /user/username
```

Verify that the -copyFromLocal command worked.

```
$ hadoop fs -ls .  # note dot at end
$ hadoop fs -ls tutorial
```

Note that a dot (period) is a Hadoop shorthand for the "current" HDFS directory, /user/username.

Change to the directory containing the R code.

```
$ cd tutorial/R
```

Using an additional login window, you may open the tutorial code in a text editor to follow along, or simply cut and paste from this document.

```
$ pico tutorial.R
```

Start R

```
$ R
```

2 Tour of BlueSNP commands

2.1 Basics

The main BlueSNP function is

- gwas()

Additionally, there are two similar functions that implement different types of data permutations to estimate empirical p-values

- gwas.maxT.perm()
- gwas.adaptive.perm()

These functions have three required parameters used to specify input and output HDFS paths

- genotype.hdfs.path
- phenotype.hdfs.path
- output.hdfs.path

2.2 Analyzing quantitative phenotypes

First we analyze a quantitative phenotype using linear regression as the association test. Genotypes are represented using the minor allele count 0, 1, 2 (additive genetic model). Quantitative phenotypes are decimal values. All data was simulated with PLINK.

You should have already copied the tutorial data to the HDFS in one of the first steps of this Tutorial. The quantitative phenotype data is at the HDFS path

- /user/username/tutorial/qt/data/

2.3 Parse PLINK data files

Load the BlueSNP package.

```
> library(BlueSNP)
```

Parse PLINK tped file(s) into SNP records. Since the tutorial data set is unreasonably small, we force multiple output files with the parameter mapred.reduce.tasks=5.

```
> read.plink.tped(
    tped.hdfs.path="tutorial/qt/data/simulated_qt.tped",
    output.hdfs.path="tutorial/qt/snps",
    mapred.reduce.tasks=5
  )
```

Parse PLINK tfam file into a phenotype data matrix.

```
> read.plink.tfam(
    "tutorial/qt/data/simulated_qt.tfam",
    "tutorial/qt/pheno.RData"
  )
```

2.4 Analyze one trait

Run the default association test (linear regression)

```
> gwas(
    "tutorial/qt/snps",
    "tutorial/qt/pheno.RData",
    "tutorial/qt/results",
    pvalue.report.cutoff=.1
  )
```

Load output from gwas() into R workspace

```
> results = gwas.results("tutorial/qt/results")
```

```
> head(results)

           type     rsid chr  bp           V5
1            R2 null_114   1 125 4.610443e-03
2          beta null_114   1 125 1.090830e-01
3 n.individuals null_114   1 125 1.000000e+03
4       p.value null_114   1 125 3.179432e-02
5            se null_114   1 125 5.073609e-02
6   t.statistic null_114   1 125 2.150008e+00
```

Since there's only one phenotype, we can reshape results into a more intuitive summary table.

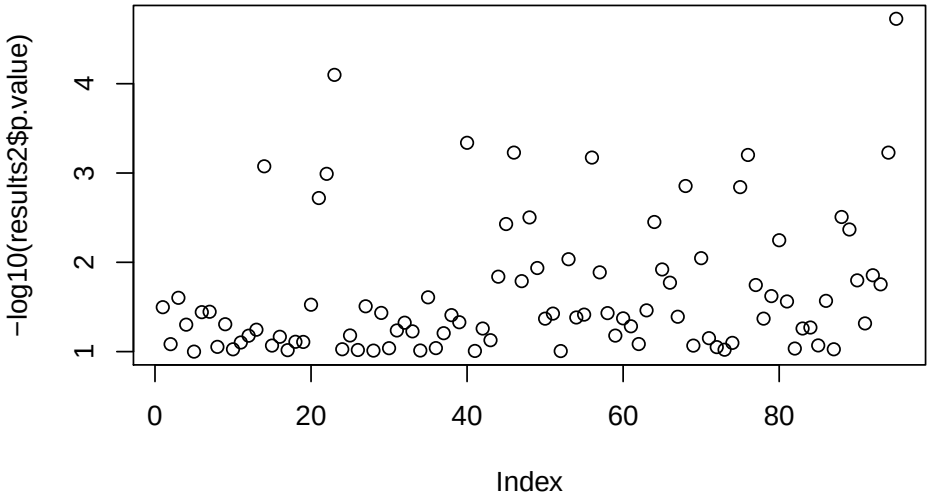
```
> results2 = gwas.results.reshape(results)

> head(results2)

      rsid chr  bp          R2        beta n.individuals    p.value         se
1 null_114   1 125 0.004610443  0.10908301          1000 0.03179432 0.05073609
2 null_169   1 180 0.003018695  0.07965155          1000 0.08246170 0.04582082
3 null_213   1 224 0.005026125 -0.26990585          1000 0.02496658 0.12020879
4 null_231   1 242 0.003846701  0.10681890          1000 0.04990985 0.05441288
5 null_268   1 279 0.002709615 -0.10891234          1000 0.09993940 0.06614071
6 null_282   1 293 0.004391820  0.09581326          1000 0.03613993 0.04566488
  t.statistic
1    2.150008
2    1.738327
3   -2.245309
4    1.963118
5   -1.646676
6    2.098183
```

We can plot the negative log of the p-values.

```
> plot(-log10(results2$p.value))
```



The combination of small sample size and low magnitude effect size conspire to create association statistics that are not genome-wides significant at typical levels.

2.5 Analyze multiple traits

The following function, included in tutorial data package, makes some additional (fake) phenotypes. It writes tutorial/pheno10.RData to the HDFS, a matrix of 10 columns corresponding to 10 phenotypes.

```
> source("~/tutorial/R/generate_more_phenotypes.R")  # local file system
> generate.more.phenotypes(
    "tutorial/qt/pheno.RData",
    "tutorial/qt/pheno10.RData", 10
  )
```

Now we perform a GWAS on three of the phenotypes, specified by column name or column number of the pheno10.RData matrix using the parameter, phenotype.cols.

```
> gwas(
    "tutorial/qt/snps",
    "tutorial/qt/pheno10.RData",
    "tutorial/qt/results-multi",
    pvalue.report.cutoff=.001,
    phenotype.cols=1:3
  )
```

Load the results into the workspace.

```
> results = gwas.results("tutorial/qt/results-multi")

> head(results)

           type     rsid chr  bp            V5  V6            V7
1            R2 null_572   1 583  1.111520e-02 NaN  1.111520e-02
2          beta null_572   1 583 -2.090643e-01 NaN -2.090643e-01
3 n.individuals null_572   1 583  1.000000e+03 NaN  1.000000e+03
4       p.value null_572   1 583  8.405417e-04 NaN  8.405417e-04
5            se null_572   1 583  6.242071e-02 NaN  6.242071e-02
6   t.statistic null_572   1 583 -3.349278e+00 NaN -3.349278e+00
```

Restrict the results to p-values (omit other information such as regression coefficient, number of samples, etc.).

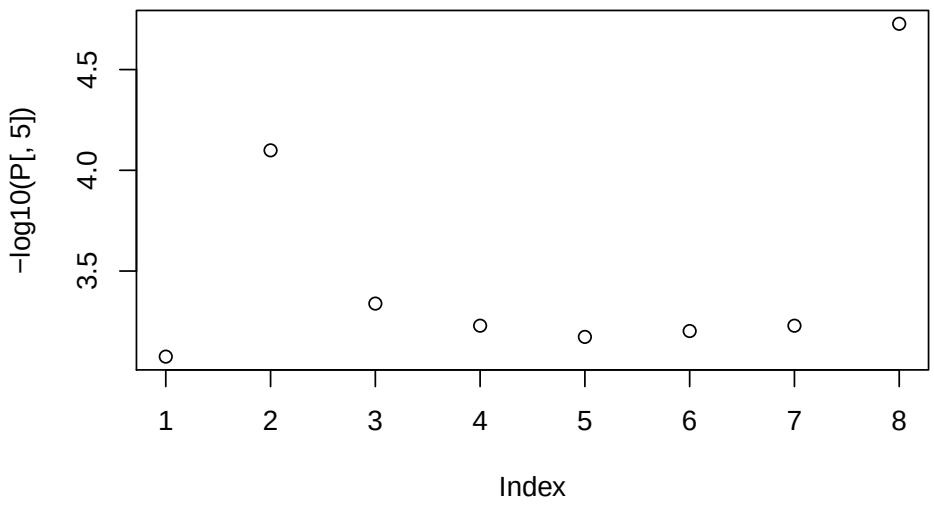
```
> P = gwas.results("tutorial/qt/results-multi", type="p.value")

> head(P)

     type     rsid chr  bp           V5  V6           V7
1 p.value null_572   1 583 0.0008405417 NaN 0.0008405417
2 p.value    qtl_6   1   7 0.0000795760 NaN 0.0000795760
3 p.value null_812   1 823 0.0004585407 NaN 0.0004585407
4 p.value    qtl_7   1   8 0.0005903245 NaN 0.0005903245
5 p.value null_482   1 493 0.0006711194 NaN 0.0006711194
6 p.value    qtl_4   1   5 0.0006274548 NaN 0.0006274548
```

Cols 1-4 of P contain genome mapping info, col 5 holds the results for pheno1, col 6 holds the results for pheno2, and so on. NA entries in P occur if the pvalue.report.cutoff option of gwas() is not satisfied for a particular phenotype.

```
> plot(-log10(P[,5]))
```



We can reshape results of one phenotype (col 5) into a more intuitive summary table.

```
> results2 = gwas.results.reshape(results[,1:5])

> head(results2)

      rsid chr  bp         R2       beta n.individuals      p.value         se
1 null_572   1 583 0.01111520 -0.2090643          1000 0.0008405417 0.06242071
2    qtl_6   1   7 0.01548618 -0.3208153          1000 0.0000795760 0.08097082
3 null_812   1 823 0.01223247 -0.2928078          1000 0.0004585407 0.08328901
4    qtl_7   1   8 0.01176606  0.1565992          1000 0.0005903245 0.04542958
5 null_482   1 493 0.01152958 -0.1960678          1000 0.0006711194 0.05746668
6    qtl_4   1   5 0.01165357  0.1592590          1000 0.0006274548 0.04642627
  t.statistic
1   -3.349278
2   -3.962110
3   -3.515564
4    3.447076
5   -3.411851
6    3.430364
```

2.6 maxT permutation

The maxT procedure for estimating family-wise p-values is available.

```
> gwas.maxT.perm(
    "tutorial/qt/snps",
```

```
    "tutorial/qt/pheno.RData",
    "tutorial/qt/results-maxT",
    n.permutations=100
  )
> results = gwas.results.perm("tutorial/qt/results-maxT")

> head(results)

  phenotype.name     rsid chromosome position    p.value p.value.adjusted
1              1 null_812          1      823 0.00990099             0.34
2              1    qtl_3          1        4 0.00990099             0.99
3              1    qtl_7          1        8 0.00990099             0.48
4              1 null_284          1      295 0.00990099             1.00
5              1 null_482          1      493 0.00990099             0.50
6              1 null_167          1      178 0.00990099             0.95
  statistic.real hits tries is.finished
1      -3.515564    0   100       FALSE
2       2.907339    0   100       FALSE
3       3.447076    0   100       FALSE
4      -2.608503    0   100       FALSE
5      -3.411851    0   100       FALSE
6      -2.960724    0   100       FALSE
```

2.7 Adaptive permutation

The adaptive permutation procedure drops SNPs from further consideration when the current p-value estimate exceeds a threshold. A SNP is dropped when there are N or more occurrences of a random test statistic greater than the actual test statistic. The default value of N is 5. This default supports the estimation of p-values with $p \leq 5x10^{-N}$.

```
> gwas.adaptive.perm(
    "tutorial/qt/snps",
    "tutorial/qt/pheno.RData",
    "tutorial/qt/results-adaptive",
    n.permutations=2e5
  )
> results = gwas.results.perm("tutorial/qt/results-adaptive")

> head(results)

  phenotype.name     rsid chromosome position      p.value p.value.adjusted
1              1    qtl_9          1       10 1.749996e-05             NULL
2              1    qtl_6          1        7 8.173655e-05             NULL
3              1    qtl_4          1        5 4.041337e-04             NULL
```

```
4              1 null_482          1      493 5.891760e-04             NULL
5              1    qtl_7          1        8 5.926679e-04             NULL
6              1    qtl_5          1        6 6.039168e-04             NULL
  statistic.real hits  tries is.finished
1      -4.300247    6 400000        TRUE
2      -3.962110    6  85640        TRUE
3       3.430364    6  17320        TRUE
4      -3.411851    6  11880        TRUE
5       3.447076    6  11810        TRUE
6       3.446974    6  11590        TRUE
```

The p.value.adjusted column only pertains to maxT permutation, not adaptive permutation, so the values are NULL.

2.8 Analyzing case-control phenotypes

Simulated case-control phenotypes are also provided in the tutorial data package. Categorical phenotypes should be encoded using either control=1, case=2 or control=0, case=1. All data was simulated with PLINK which uses the 1, 2 representation.

You should have already copied the tutorial data to the HDFS in one of the first steps of this Tutorial. The case-control phenotype data is at the HDFS path

- /user/username/tutorial/cc/data/

```
> read.plink.tped(
    "tutorial/cc/data/simulated_cc.tped",
    "tutorial/cc/snps",
    mapred.reduce.tasks=5
  )
> read.plink.tfam(
    "tutorial/cc/data/simulated_cc.tfam",
    "tutorial/cc/pheno.RData"
  )
```

The following function, included in the tutorial data package, makes some additional (fake) phenotypes. It writes tutorial/pheno10.RData to the HDFS, a matrix of 10 columns corresponding to 10 phenotypes.

```
> source("~/tutorial/R/generate_more_phenotypes.R")
> generate.more.phenotypes("tutorial/cc/pheno.RData",
    "tutorial/cc/pheno10.RData", 10)
```

An appropriate case-control test such as the allelic test (cc.allelic) or logistic regression test (cc.logistic) is required. See the parameter method=.

```
> gwas(
    "tutorial/cc/snps",
    "tutorial/cc/pheno10.RData",
    "tutorial/cc/results-allelic",
    method="cc.allelic",
    pvalue.report.cutoff=0.001,
    phenotype.cols=1:3
  )
> results = gwas.results("tutorial/cc/results-allelic")

> head(results)

           type      rsid chr bp           V5  V6           V7
1        chi.sq disease_3   1  4 2.902815e+01 NaN 2.902815e+01
2 n.individuals disease_3   1  4 1.000000e+03 NaN 1.000000e+03
3    odds.ratio disease_3   1  4 6.042833e-01 NaN 6.042833e-01
4       p.value disease_3   1  4 7.133407e-08 NaN 7.133407e-08
5        chi.sq disease_7   1  8 3.591037e+01 NaN 3.591037e+01
6 n.individuals disease_7   1  8 1.000000e+03 NaN 1.000000e+03
```

Inspect results for one phenotype (column 5).

```
> results2 = gwas.results.reshape(results[,1:5])

> head(results2)

      rsid chr  bp   chi.sq n.individuals odds.ratio      p.value
1 disease_3   1   4 29.02815          1000  0.6042833 7.133407e-08
2 disease_7   1   8 35.91037          1000  0.5782209 2.066059e-09
3  null_125   1 136 14.79490          1000  1.4118774 1.198592e-04
4 disease_4   1   5 25.86207          1000  0.5798319 3.667047e-07
5 disease_8   1   9 30.12867          1000  1.6572641 4.043087e-08
6 disease_2   1   3 28.39912          1000  2.1200942 9.871023e-08
```

Inspect the p-values for all phenotypes.

```
> P = gwas.results("tutorial/cc/results-allelic", type="p.value")

> head(P)

    type      rsid chr  bp           V5  V6           V7
1 p.value disease_3   1   4 7.133407e-08 NaN 7.133407e-08
2 p.value disease_7   1   8 2.066059e-09 NaN 2.066059e-09
3 p.value  null_125   1 136 1.198592e-04 NaN 1.198592e-04
4 p.value disease_4   1   5 3.667047e-07 NaN 3.667047e-07
5 p.value disease_8   1   9 4.043087e-08 NaN 4.043087e-08
6 p.value disease_2   1   3 9.871023e-08 NaN 9.871023e-08
```

3 User-defined association tests

3.1 From scratch

Functions that perform an association test follow conventions illustrated in the following example.

```
> # my_custom_test.R
> my.custom.test <- function(y, x) {
    # y is phenotype vector {0,1} = {control,case} or {1,2} = {control,case}
    # x is genotype {0,1,2} = number of copies minor allele

    # REQUIRED CONVENTION
    # Return output var names when function is called with no params
    if (nargs()==0) {  # called with no params
      y = sample(0:1, 100, replace=T)  # dummy data
      x = sample(0:2, 100, replace=T)
      return(names(my.custom.test(y, x)))
    }

    # select elements with values
    is = !is.na(x) & !is.na(y)
    x = x[is]
    y = y[is]

    # number of individuals
    N = as.numeric(sum(is))

    # our novel test statistic
    stat = cor(y, x)^2 * (N - 2)

    # REQUIRED CONVENTION
    # return a list of named entries
    list(n.individuals=N, stat=stat)
  }
```

Note the two conventions: the return value is a list of named elements, and the function called without arguments returns the names of the returned list elements. Also note that the return list elements in this example don't include an element called "p.value". Therefore this function is suitable for estimating empirical p-values using gwas.adaptive.perm() or gwas.maxT.perm() but this function can not be used by gwas() which requires a "p.value" return value. To use this function with gwas() we must return a dummy p-value, preferably one that will alert the user that the p-value is not real.

...

```
  # returns dummy p-value for compatability with gwas()
  list(n.individuals=N, stat=stat, p.value=2)
}
```

The function my.custom.test() is included in tutorial/R/my_custom_test.R and was already copied to the HDFS in one of the first steps of this Tutorial. Otherwise, at the UNIX command prompt, copy the text file from the local filesystem to the HDFS.

```
$ hadoop fs -copyFromLocal ~/tutorial/R/my_custom_test.R tutorial/R
```

Note that my.custom.test() does not return a p-value, only a test statistic. We estimate empirical p-values using gwas.maxT.perm() or gwas.adaptive.perm().

```
> gwas.maxT.perm(
    "tutorial/cc/snps",
    "tutorial/cc/pheno.RData",
    "tutorial/cc/results-custom",
    n.permutations=200,
    user.code="tutorial/R/my_custom_test.R",
    method="my.custom.test",
    statistic.name="stat"
  )
```

Note that we supplied parameters specifying the HDFS path of the text file containing the function definition of the user-defined code, the name of the user-defined function implementing the statistical test, and the name of the list element holding the test statistic returned by user-defined function.

```
> results = gwas.results.perm("tutorial/cc/results-custom")
```

```
> head(results)
```

```
  phenotype.name      rsid chromosome position     p.value p.value.adjusted
1              1 disease_2          1        3 0.004975124            0.000
2              1 disease_6          1        7 0.004975124            0.000
3              1  null_373          1      384 0.004975124            0.825
4              1 disease_4          1        5 0.004975124            0.000
5              1 disease_8          1        9 0.004975124            0.000
6              1  null_338          1      349 0.004975124            0.985
  statistic.real hits tries is.finished
1      26.576756    0   200       FALSE
2      36.753993    0   200       FALSE
3       9.953149    0   200       FALSE
4      26.376434    0   200       FALSE
5      31.248532    0   200       FALSE
6       8.209132    0   200       FALSE
```

Inspect SNPs with adjusted p-values below a threshold.

```
> subset(results, p.value.adjusted<.01)
```

```
   phenotype.name      rsid chromosome position     p.value p.value.adjusted
1               1 disease_2          1        3 0.004975124                0
2               1 disease_6          1        7 0.004975124                0
4               1 disease_4          1        5 0.004975124                0
5               1 disease_8          1        9 0.004975124                0
7               1 disease_3          1        4 0.004975124                0
10              1 disease_7          1        8 0.004975124                0
   statistic.real hits tries is.finished
1        26.57676    0   200       FALSE
2        36.75399    0   200       FALSE
4        26.37643    0   200       FALSE
5        31.24853    0   200       FALSE
7        29.76965    0   200       FALSE
10       35.31395    0   200       FALSE
```

The p.value.adjusted column is the family-wise p-value from the maxT procedure.

3.2 Using an association test defined in another package

A user-defined association test can make use of another R package so long as the user-defined function loads the package using the library(packagename). As an example, we demonstrate how to use the EMMA package for efficient mixed-model association [1].

3.2.1 Efficient Mixed-Model Association (EMMA)

First, download and install the EMMA R package. You must obtain emma from the author's website (http://mouse.cs.ucla.edu/emma/). WARNING: The CRAN package called emma is unrelated to genetics!

Download and install EMMA.

```
$ wget http://mouse.cs.ucla.edu/emma/emma_1.1.2.tar.gz
$ sudo R CMD INSTALL emma_1.1.2.tar.gz
```

We use the Tutorial case-control data set on the local filesystem at (/tutorial/cc/data) and on the HDFS at (/user/username/tutorial/cc/data).

In addition to genotype and phenotype data, mixed-model association requires a kinship matrix specifying relations among the individuals, such as the strain. Emma provides a helper function for computing this directly from the SNP data. This is a computationally intensive step that is done one time and saved for later.

Load a small subset of the SNP data into the R workspace using the BlueSNP helper function peek() which is intended to be used for debugging, but as a side effect is also useful for loading a small subset of records into the R workspace.

```
> library(emma)
> library(BlueSNP)
> peek("tutorial/cc/snps", 20)

[BlueSNP] Reusing existing Rhipe connection
[1] "chromosome" "rsid"       "distance"   "position"   "snp.vector"
```

We fetched 20 SNP records. We now have two lists in the R workspace, keys and values.

```
> head(keys)

     [,1]
[1,] "null_0"
[2,] "null_5"
[3,] "null_10"
[4,] "null_15"
[5,] "null_24"
[6,] "null_29"
```

This R one-liner builds a genotype matrix from the list, values.

```
> X = do.call("cbind", lapply(values, "[[", "snp.vector"))
```

The emma.kinship() function expects the transpose of X and SNP values in 0, .5, 1 instead of 0, 1, 2, thus we take the transpose with t() and divide by 2. (For the purpose of this tutorial, it is not important to understand the details of the EMMA data format.)

```
> K = emma.kinship(t(X/2))
```

We need to save the kinship matrix K to an appropriate place. Currently the phenotype matrix (Y) is located in /tutorial/cc/pheno.RData. This is an ideal place to save K. Fetch phenotype.RData from the HDFS to the local FS.

```
> rhget("tutorial/cc/pheno.RData", ".")
```

Load it into the R workspace.

```
> load("pheno.RData")
```

Re-save Y and K to a new RData file.

```
> save(file="pheno_and_kinship.RData", list=c("Y", "K"))
```

And copy it to the HDFS

```
> rhput("pheno_and_kinship.RData", "tutorial/cc")
```

Let's fit the emma model to one SNP.

```
> y = Y[,1]
> x = X[,1]
> emma.MLE(y, cbind(1, x/2), K)

$ML
[1] -725.7552

$delta
[1] 22026.47

$ve
[1] 0.2499707

$vg
[1] 1.134865e-05
```

Now, we write a function to perform this test using BlueSNP. In a text editor create the file my_emma_test.R.

```
# my_emma_test.R
linear.mixed.model <- function(y, x) {
  # y is phenotype vector {0,1} or {1,2} = {control, case}
  # x is genotype vector {0,1,2}

  # REQUIRED CONVENTION
  # return output names when function is called with no args
  if (nargs()==0) {  # called with no params
    return(c("ML", "delta", "ve", "vg", "n.individuals", "p.value"))
  }

  # select elements with values
  is = !is.na(x) & !is.na(y)
  x = x[is]
  y = y[is]

  N = as.numeric(sum(is))

  # allow {1,2} instead of {0,1} labels
  if (max(y) > 1) {
    if (max(y)==2) {
      y = y - 1
    } else {
      stop("case-control phenotype must be encoded as {1,0} or {2,1}")
```

```
    }
  }

  library(emma)  # require() gives errors with Rhipe

  # due to lexical scoping, K is available in the calling environment
  results = emma.MLE(y, cbind(1, x/2), K)  # returns a list

  # emma.MLE does not return a p.value but gwas()
  # requires a p.value for filtering on p.value
  # so we fudge this with a p.value of 2
  # to provide a clue that it's not real.
  c(results, n.individuals=N, p.value=2)
}
```

my_emma_test.R is already on the HDFS at /tutorial/cc/R/my_emma_test.R. You can overwrite by copying from the local filesystem to the HDFS

```
rhput("my_emma_test.R", "tutorial/cc/R")
```

Finally, run the association tests.

```
> library(BlueSNP)
> gwas(
    genotype.hdfs.path="tutorial/cc/snps",
    phenotype.hdfs.path="tutorial/cc/pheno_and_kinship.RData",
    output.hdfs.path="tutorial/cc/results-emma",
    user.code="tutorial/R/my_emma_test.R",
    method="linear.mixed.model",
    pvalue.report.cutoff=3
  )
```

References

[1] Hyun Min Kang, Noah A Zaitlen, Claire M Wade, Andrew Kirby, David Heckerman, Mark J Daly, and Eleazar Eskin. Efficient control of population structure in model organism association mapping. *Genetics*, 178(3):1709–1723, Mar 2008.