Using *MultiPhen*: and example with PLINK format data

Lachlan Coin and Federico Calboli

July 6, 2015

1 Introduction

This tutorial aims to show the use of the R package *MultiPhen* to analyse genetic data, which has been produced elsewhere and has been transformed in PLINK binary format, one of the most common formats used to store large genetic data. As mentioned in the vignette detailing the basic usage of *MultiPhen*, this tutorial will show the basic steps in the use of the functions provided by the package in addition to showing how to use *MultiPhen* for data that has been stored in PLINK format. While previous verions of the *MultiPhen* package already had the ability to read in and to analyse genetic data in PLINK format, with version 2.0.0 we have include many enhancements, and we recommend all users, even users who had used *MultiPhen* 1.x.x beofre to read this vignette to get a better understanding of the new functions. We have tried to maintain back compatibility as much as possible, but we feel that the future of the package will see the most active development to take place for the new fucntions, thus we encourage all users to prefer using them, and to change any legacy script they might have.
The test data is supplied with the package and is in the same directory where this vignette is.

1.1 Installation

Before going further, we will make sure that *MultiPhen* and all the packages it depends from are installed on the computer. Current version of the package is 2.0.0. Make sure you have a recent version of R ($\geq$ 3.0.2) by typing:

```
> R.version.string
```

```
[1] "R version 3.2.1 (2015-06-18)"
```

Assuming that is the case (ideally the version of R should be the latest stable version), we can install *MultiPhen* with all its dependencies typing:

```
> install.packages("MultiPhen", dependecies = TRUE)
```

We can now load *MultiPhen* by typing:

```
> library(MultiPhen)
```

2 Usage

We start by explaining the way that MultiPhen uses options to customise the analysis run. MultiPhen has several options, which all have the form "mPhen.xxx", which are grouped in the following categories: "regresssion", "geno.input", "pheno.input", "plot","meta.analysis", "misc". The following snippet prints out the description of all the options, prints just the 'misc' options, and then sets the mPhen.logp option. Note that if you use 'Rscript' on the command line to run R, you can also set options in the command line (e.g. Rscript MultiPhen.plink.R –mPhen.logp=FALSE ...). This option specifies whether the MultiPhen p-values should be calculated and reported in log (base 10).

```
> library(MultiPhen)
> options(width = 60) # for printing this vignette is a reasonably acceptable way
> print(mPhen.options("all",descr = TRUE))
```

```
$mPhen.inverseRegress
[1] "whether to use genotypes as outcome"

$mPhen.JointModel
[1] "whether to use a multivariate model"

$mPhen.scoreTest
[1] "whether to use a score test rather than likelihood test, for ordinal inverse regression only"

$mPhen.calcHWE
[1] "whether to calculate HWE values"

$mPhen.exactMethod
[1] "if non-null, then which exact method to use. Can be wald,midp,fisher,small, see ?oddsratio()"

$mPhen.rescale
[1] "when using imputed values and ordinal regression, how to scale probabilities to get integer values"

$mPhen.variable.selection
[1] "whether backward selection should be used to reduce number of associated variables"

$mPhen.link.geno
[1] "the link function when using genotype as outcome, can be ordinal or gaussian"

$mPhen.adjustSinglePv
[1] "whether to use a Nyoldt-Sidak correction on single-trait pvalue associations based on effective number of phenotypes tested"

$mPhen.useExpected
[1] "whether to converted imputed data to dosage data. Should set link.geno to gaussian in this case"

$mPhen.link.pheno
[1] "link function when using phenotpe as outcome.  If it is NULL, then automatically determined based on number of unique levels i

$mPhen.ccarep
[1] "Number of repetitions when using mPhen.cca"

$mPhen.thresh.diff
[1] "Threshold on euclidian distances (L2) between genotype weights from one iteration to the next, and also between phenotype weig

$mPhen.orthPheno
[1] "Whether to orthogonalise phenotypes prior to association. Can be 'none', 'PCA', or 'GramSchmidt'"

$mPhen.sigThresh
[1] "only plot pvalues more significant than this"

$mPhen.noPhenosPerPlot
[1] "maximum number of phenotypes to include per plot"

$mPhen.limitP
[1] "the smallest pvalue to plot in manhattan (smaller values will be set to this value)"

$mPhen.title
[1] "base title for all plots"

$mPhen.minlogpv
[1] "the smallest pvalue to show on qqplot"

$mPhen.colourByChroms
[1] "whether to colour different chroms on manhattan by colour, or to colour different phenotypes by different colours.  Phenotypes

$mPhen.cex
[1] "scaling for x-axis in plots"

$mPhen.pool
[1] "in drawing qq plot, whether to pool pvalues across phenotypes before drawing plot"

$mPhen.Colv
[1] "whether to use a dendrogram in heatmap on phenotypes.  In some cases making the dendrogram will throw an error, in which case s

$mPhen.indexMatch
[1] "this is the co-ordinates of a locus you wish to highlight on heatmap"

$mPhen.onlyShowJoint
[1] "whether to only show Joint model pvalues whenever a joint model is used.  If false, then joint model not plotted, if TRUE, onl

$mPhen.nacolor
[1] "what colour to use on heatmap for NA"

$mPhen.batch
[1] "the number of snp genotypes to read into memory"

$mPhen.baseValue
[1] "Default genotype, use 0 for SNP genotypes, and 2 for CN genotypes"

$mPhen.format
[1] "Which format field to read in from VCF file"

$mPhen.starting
[1] "Defines start position on chromosome for analysis"

$mPhen.ending
[1] "Defines end position on chromosome for analysis"
```

```
$mPhen.thin
[1] "Allows taking every 10th snp, default is 1"

$mPhen.rsidToDo
[1] "Which rs identifiers to run the analysis on"

$mPhen.matchPosition
[1] "whether to match mPhen.rsidToDo to position rather than id"

$mPhen.numGenoPCs
[1] "How many geno pcs to calculate.  Note: the relatedness matrix is updated if this value is greater than zero, but the PCs (eige

$mPhen.onlyCommon
[1] "whether to restrict to only positions in common when merging multiple cohorts"

$mPhen.sep.pheno
[1] "separator between cols in phenotype file"

$mPhen.onlyCommonPheno
[1] "whether to include only common phenotypes"

$mPhen.numHeaderRows.pheno
[1] "how many header rows in phenotype file"

$mPhen.naRowThresh
[1] "exclude samples with more than this fraction of missing genotypes"

$mPhen.naColThresh
[1] "exclude phenotypes with more than this fraction of missing values"

$mPhen.fillMissingPhens
[1] "whether to impute missing phenotype values when reading in a phenotype file"

$mPhen.quantileThresh
[1] "If  excludeFile has a second column, indicating quality, then include the top percentage only in analysis"

$mPhen.sep.pheno
[1] "separator between cols in phenotype file"

$mPhen.onlyCommonPheno
[1] "whether to include only common phenotypes"

$mPhen.numHeaderRows.pheno
[1] "how many header rows in phenotype file"

$mPhen.naRowThresh
[1] "exclude samples with more than this fraction of missing genotypes"

$mPhen.naColThresh
[1] "exclude phenotypes with more than this fraction of missing values"

$mPhen.fillMissingPhens
[1] "whether to impute missing phenotype values when reading in a phenotype file"

$mPhen.quantileThresh
[1] "If  excludeFile has a second column, indicating quality, then include the top percentage only in analysis"

$mPhen.log10p
[1] "whether to calculate probabilities in log10 space"

$mPhen.defaultBackwardFrac
[1] "what proportion of variables to remove at each iteration"

$mPhen.defaultBackwardThresh
[1] "threshold for backward selection significance"

$mPhen.defaultUseBF
[1] "whether to use bayes factor for backward selection"

$mPhen.resultsDir
[1] "output directory"

$mPhen.metaMinCohortNum
[1] "the minimum number of cohorts which have a beta/se in order to report a meta-analysis result"

$mPhen.useFisherIfAllBetaNA
[1] "if beta is not available, whether to revert to Fishers method for combining pvalues"


> print(mPhen.options("misc",descr = TRUE))


$mPhen.log10p
[1] "whether to calculate probabilities in log10 space"

$mPhen.defaultBackwardFrac
[1] "what proportion of variables to remove at each iteration"

$mPhen.defaultBackwardThresh
[1] "threshold for backward selection significance"
```

```
$mPhen.defaultUseBF
[1] "whether to use bayes factor for backward selection"

$mPhen.resultsDir
[1] "output directory"

$mPhen.metaMinCohortNum
[1] "the minimum number of cohorts which have a beta/se in order to report a meta-analysis result"

$mPhen.useFisherIfAllBetaNA
[1] "if beta is not available, whether to revert to Fishers method for combining pvalues"


> options("mPhen.log10p"=FALSE)
```

First, we read the phenotypes from a phenotype file, for which the first column specifies sample ids, and the first row specifies phenotype names. This requires a set of default options specifying how to read the phenotype input, for example the delimiter between columns, which is tab by default. Note that a plink style '.fam' file can also be read as a phenotype file, however, a space delimiter should be specified.

```
> pheno.opts = mPhen.options("pheno.input")
> pheno = mPhen.readPhenoFiles("pheno.txt", opts = pheno.opts)
> pheno.opts$mPhen.sep.pheno=" "
> pheno.opts$mPhen.numHeaderRows.pheno = 0
> pheno.hapmap = mPhen.readPhenoFiles("hapmap2.fam", opts = pheno.opts)
```

Note that *pheno* is a list with two elements - the first element is a matrix of phenotype values, and the second is a list which specifies which phenotypes to use as for association, and which to use as covariates, residuals, stratification or exclusion indices. This second list can be modified as desired.

```
> print(names(pheno))


[1] "pheno" "limit"


> print(pheno$limit)


$phenotypes
[1] "pheno1" "pheno2" "pheno3" "pheno4" "pheno5" "pheno6"

$covariates
NULL

$resids
NULL

$strats
NULL

$excls
NULL


> pheno.hapmap$limit$covariates = pheno.hapmap$limit$phenotypes[1]
> pheno.hapmap$limit$phenotypes = pheno.hapmap$limit$phenotypes[-1]
```

The next step is to prepare the phenotype data for association analysis. This includes carrying out any phenotype transformations, (including potentially orthogonalising the phenotype data) as well as pre-calculating covariance matrices, residual offsets and stratification indices. This step uses options specified in the 'regression' options category. The *pheno$limit* is used here to define which are the outcome variables, and which are used as covariates, etc.

```
> opts = mPhen.options("regression")
> phenoObject = mPhen.preparePheno(pheno,opts = opts)


[1] "excluding 0 samples based on exclusion criteria"


> phenoObject.hapmap = mPhen.preparePheno(pheno.hapmap,opts = opts)


[1] "excluding 0 samples based on exclusion criteria"
```

```
> numPhenos = length(phenoObject$phenN)
> numPhenos.hapmap = length(phenoObject.hapmap$phenN)
```

Next, we get the default options for reading in genotypes, and for regression. We modify the mPhen.batch option so that we only read in 100 genotypes at a time. We also specify the input file, which can be in multiple formats. The mPhen.format option is used for multi-format input files, such as a vcf or a zip file. In this case we will specify the 'GT' or genotype format, but we can also specify a 'GL' or genotype-likelihood format.

```
> geno.opts = mPhen.options("geno.input")
> opts = mPhen.options("regression")
> geno.opts$mPhen.batch = 100
> geno.opts$mPhen.format = "GT"
```

Next, we read genotypes from the file. Note also that if the data is read in as imputed (supported for .vcf and .zip formats, but not plink format), then the genotype matrix will be 3 dimensional, with the third dimension specifying the probability. Note that in this example we specify the 'indiv' variable when reading in the genotype file. This will ensure that the genotypes are re-ordered to be in the same order as in the phenotype file. Note that as plink input comprises multiple files with the same root name, we just specify the root name. across possible genotypes. Note that this superseeds the "read.plink" function, which is still in the package for backward compatibility.

```
> file = "ALL.chr21.integrated.phase1.v3.20101123.snps.indels.svs.genotypes.extract.zip"
> genoConnection <-mPhen.readGenotypes(file, opts = geno.opts,
+                                      indiv = rownames(pheno$pheno))


[1] "first last 9412126 9414592"


> geno <-genoConnection$genoData
> dimg = dim(geno)
> file = "hapmap2"
> genoConnection.hapmap <-mPhen.readGenotypes("hapmap2", opts = geno.opts,
+                                             indiv = rownames(pheno.hapmap$pheno))
> geno.hapmap <-genoConnection.hapmap$genoData
```

3 Association

. We first run an inverse regression using the default MultiPhen model using *mPhen.assoc*. The default options for regression can be viewed by printing the opts object obtained in the previous step. In particular, note that opts$mPhen.inverseRegress = TRUE; opts$mPhen.JointModel = TRUE and opts$mPhen.link.geno = "ordinal".

```
> print(opts)


$mPhen.inverseRegress
[1] TRUE

$mPhen.JointModel
[1] TRUE

$mPhen.scoreTest
[1] FALSE

$mPhen.calcHWE
[1] FALSE

$mPhen.exactMethod
NULL

$mPhen.rescale
[1] 100

$mPhen.variable.selection
[1] FALSE

$mPhen.link.geno
[1] "ordinal"

$mPhen.adjustSinglePv
[1] FALSE

$mPhen.useExpected
[1] FALSE

$mPhen.link.pheno
```

```
NULL

$mPhen.ccarep
[1] 5

$mPhen.thresh.diff
[1] 0.001

$mPhen.orthPheno
[1] "none"
```

```
> resultsJoint = mPhen.assoc(geno, phenoObject,  opts = opts)
```

The results include both the calculated minor allele frequency (results$maf) as well a 4 dimensional array containing the association results (results$Results). The first dimension is strata (e.g. male/female/both), the second dimension is snpid, the third dimension is phenotype and the fourth dimension is statistic (i.e. Beta, p-value and Nobs). Note that since the JointModel was specified, the size of the phenotype dimension is equal to numPhenos +1, with the extra column storing the results for the overall model. Also note that if JointModel is specified, then the single-trait pvalues given are those calculated from t-test within the regression model, and reflect the significance of that variable in the context of all the other variables included in the model. With many correlated phenotypes, these are often much less significant than single-trait association pvalues. So, we can identify which snp indices have nominally significant JointModel associations, and print to screen the pvalues of association at those genotypes

```
> sigInds = which(resultsJoint$Res[,,numPhenos+1,2]<0.05)
> print(resultsJoint$Res[,sigInds,,2])
```

```
              pheno1    pheno2    pheno3    pheno4
21_9412126 0.07050237 0.1147210 0.7234937 0.8712047
21_9414592 0.12719936 0.2703818 0.4726575 0.9049065
              pheno5    pheno6   JointModel
21_9412126 0.5265230 0.5995167 7.958833e-24
21_9414592 0.9908531 0.6242443 1.804836e-06
```

Next, we might wish to carry out a variable selection procedure to identify which are the primary phenotypes associated. Note that if the number of snps in geno is large, we might prefer to restrict the analysis to geno[,sigInds,drop=F]. Note the use of drop=F to preserve the dimensions of geno. We also set opts$mPhen.link.geno="gaussian" to improve speed of variable selection, but it is also possible to keep the 'ordinal' link.

```
> opts$mPhen.variable.selection=TRUE
> opts$mPhen.link.geno="gaussian"
> resultsBackwardSelection =mPhen.assoc(geno,
+                                       phenoObject,  opts=opts)
> #print p-values
> print(resultsBackwardSelection$Res[,,,2])
```

```
                pheno1      pheno2 pheno3 pheno4 pheno5
21_9412126 1.101641e-26 3.65775e-15      1      1      1
21_9414592 3.430628e-09 8.91976e-04      1      1      1
           pheno6   JointModel
21_9412126      1 3.709352e-27
21_9414592      1 2.412490e-08
```

```
> #print betas
> print(resultsBackwardSelection$Res[,,,1])
```

```
              pheno1      pheno2 pheno3 pheno4 pheno5
21_9412126 -0.2926753 -0.15470925      0      0      0
21_9414592 -0.1366723 -0.05558485      0      0      0
           pheno6 JointModel
21_9412126      0         NA
21_9414592      0         NA
```

Note that of the pvalues reported here, many are now set to 1.0 (and correspondingly the betas are 0), which indicates that phenotype is not included in the variable selection model. The overall model p-value represents the model p-value only using those variables which are selected (i.e. have single-trait pvalues less than 1.0), and is often more significant that the original model p-value. However, the process of variable selection does lead to inflation of this smaller model p-value, so it should be treated with a little caution.
Next, we calculate standard single-trait association p-values , by setting JointModel = FALSE and inverseRegress = FALSE

```
>    opts$mPhen.variable.selection=FALSE
>    opts$mPhen.JointModel=FALSE
>    opts$mPhen.inverseRegress=FALSE
>    resultsSingle = mPhen.assoc(geno, phenoObject,  opts)
```

Next, we calculate a joint genotype model (with separate phenotypes) by setting inverseRegress = FALSE, and JointModel=TRUE. We calculate association statistics

```
>   opts$mPhen.variable.selection=FALSE
>   opts$mPhen.JointModel=TRUE
>   opts$mPhen.inverseRegress = FALSE
>   resultsJointG = mPhen.assoc(geno, phenoObject,opts)
```

Next we calculate a model which is iteratively Multiple phenotypes and then Multiple genotypes, each time updating a linear combination of either phenotypes or genotypes. This is similar to a canonical correlation analysis. We use a gaussian link for speed, but an ordinal link would also work.

```
> opts$mPhen.link.geno="gaussian"
> resultsCCA =  mPhen.cca(geno,
+                         phenoObject,opts=opts)


[1] "delta 9.68465288192687 1.26059739814204"
[1] "delta 0.00484369752650435 0.00131756439557709"
[1] "delta 1.21599811471657e-05 3.25492396519234e-06"
```

4 Output

We will now save the results of our analyses to different files to have a convenient reference of our work thus far. We can do this using the *mPhen.writeOutput* function. This function writes results in multiple formats, and also generates multiple plots. First we need a list of the formats we want to write the results in, as well as a list of the plots to generate, and then we can use this to generate the desired output. Note that

```
>  resDir = "resultsDir"
>  towrite = list(long.txt = TRUE,   wide.txt = TRUE)
>  toplot = list(.manh = TRUE,.qq = TRUE,.heatm = TRUE,
+                .fprint = TRUE)
>  plotopts = mPhen.options("plot")
>  output=mPhen.writeOutput(resultsJoint,output=resDir,
+                           geno = geno, towrite = towrite,
+                           toplot = toplot, opts = plotopts)
```

The output of *mPhen.writeOutput* is an object which can be then used to write subsequent results. This is particularly useful if you are running the genetic analyses in multiple batches, in which case the results will be appended to the same file. Also note that the plots are not produced until the final batch has been read (which is indicated in attr(geno,"closeConnection")).

```
>  output=mPhen.writeOutput(resultsBackwardSelection,output=output,
+                           geno = geno, towrite = towrite,
+                           toplot = toplot,     opts = plotopts)
>  output=mPhen.writeOutput(resultsSingle,output=resDir, geno = geno,
+                           towrite = towrite, toplot = toplot,
+                           opts = plotopts)
>  output=mPhen.writeOutput(resultsJointG,output=output,
+                           geno = geno, towrite = towrite,
+                           toplot = toplot,     opts = plotopts)
```

Note that each object *resultsSingle, resultsJoint, ...* produces separate results files, which have as a prefix the name of each object respectively.

5 Simulation

This section describes how you can use MultiPhen simulate phenotype data with a given correlation structure and a genetic association in a particular direction in phenotype space. The first step is to calculate a covariance matrix. This is specified by specifying desired orthogonality between phenotypes. We specify orthogonality within and between phenotype 'blocks'. The idea is that phenotypes within blocks will be more correlated than phenotypes within different blocks. We specify how many phenotypes are within each block, using *blockSize*, which must be an integer divisor *numPhenos*. We then specify how orthogonal phenotypes should be between and within blocks respectively.

```
> blockSize = 2 ## size of each block for partitioning correlation
> orthogAll = c(0.9,0.5) ## parameters controlling
> ## how 'orthogonal' phenotypes are to each other.
> ## First entry is orthogonality between blocks, and
> ## second is orthoganility within blocks
> #parameters must in interval (0,1), with closer to 1 indicating more orthogonal
>  covar = mPhen.sampleCovar(numPhenos,blockSize,orthogAll = orthogAll)
```

We use this covariance structure to simulate phenotypes. We have to also specify the effect direction (effDir) in phenotype space, the proportion of phenotypic variance explained in this direction; and also the genetic effect, which we calculate based on specifying which snps from the genotype matrix are associated and the relative size of their effects. Lastly, we can visualise that this gives the desired correlation and association structure using the *mPhen.plotCorrelation* function. Note, that this simulation does not work for imputed data (i.e. if (length(dim(geno))==3)

```
>   effDir = c(1,1,0,0,0,0)
>   total.variance.explained = 0.1  ## total variance explained by all snps.
>   snpIndices <- c(1,2)
>   betag = c(1,0.5)
>   genoEffect = geno[,snpIndices,drop=F] %*% (betag/(betag %*% betag))
>   pheno.sim = mPhen.simulate(genoEffect,dimnames(geno)[[1]], covar,effDir,
+                              total.variance.explained, inverse=FALSE,
+           effDirInReverseEigenspace = FALSE)
>   mPhen.plotCorrelation(pheno.sim$pheno[,which(pheno.sim$effDir!=0),
+                         drop=F],geno[,snpIndices,drop=F],cex=0.5)


[1] "pheno1" "pheno2"
[1] "pheno1" "pheno2"
```

This phenotype can then be used in place of *pheno* above in the association analyses.

```
> phenoObjectSim = mPhen.preparePheno(pheno.sim,opts = opts)


[1] "excluding 0 samples based on exclusion criteria"


> opts = mPhen.options("regression")
> resultsJointSim = mPhen.assoc(geno, phenoObjectSim,  opts = opts)
```