GWAlpha

Genome-Wide estimation of additive effects (Alpha) based on trait quantile distribution from pool-sequencing experiments.

Content

Introduction

Consider a population of individuals measured for a given trait *Y* and binned into *k* pools based on their trait values. *Y* is inverse quantile-normalised into *Y'* so that for $Y \in [\min Y; \max Y]$, $Y' \in [0; 1]$. For each pool *i* encompassing all observations with trait values in $[y'_{i-1}; y'_i]$, we observe $Q_k = \{q_1, ..., q_k\}$ the fractions for a specific allele in each pool which sum to 1. From these fractions GWAlpha estimates the parameters of the distribution for this focal allele across the bins and compares it to all alternative allele states combined for the locus by measuring the difference in the means of these two distributions. The distribution of the allele is modeled using a Beta distribution so that for the *i*-th pool, the fraction of an allele is:

$$Pr(Allele|\Theta) = cdf_{Beta}(y'_i, \Theta) - cdf_{Beta}(y'_{i-1}, \Theta)$$

The parameters Θ of the distribution of the allele can be alternatively estimated by least-squares estimation solving:

$$\underset{\Theta}{argmin} \sum_{i=1}^{k} (q_i - Pr(Allele|\Theta))^2$$

or maximising the likelihood:

$$L(\Theta|Q_k) \propto \prod_{i=1}^{k} f(q_i|\Theta)$$

The distribution of the alternative allele states is modeled identically. Finally, the test statistic is obtained as: $\hat{\alpha} = W(\frac{\hat{\mu}_{Allele} - \hat{\mu}_{Alternative}}{\sigma_Y})$ which compares how the mean distribution for the allele deviates from the mean distribution of the alternative states, and where *W* is a default penalisation for low allele frequency $W = 2\sqrt{p_{Allele}(1 - p_{Allele})}$ which can be set to 1 (no penalisation).The distribution of the α's was modeled using a normal distribution with parameters estimated through maximum-likelihood; the cumulative density function of this

normal distribution is used to calculate empirical p-values.

Requirements

- Linux or MacOS operating system (Linux is recommended)
- Python 2.7.5 or above: https://www.python.org/downloads/
- Numpy 1.8.2 or above: https://www.scipy.org/scipylib/download.html
- Scipy 0.15.20 or above: https://www.scipy.org/scipylib/download.html
- R (only required for generating the Manhattan plots): https://cran.r-project.org/
- parallel (only required to run GWAlpha.sh with multiple parallel threads): http://www.gnu.org/software/parallel/

Download

All scripts and example files can be obtain at https://github.com/aflevel/GWAlpha.

Experimental Design and Input Files

GWAlpha requires to cover the entire distribution of the trait in the study sample. The results obtained from reanalysis of GWAS dataset suggest GWAlpha is best suited when trying to identify the genetic basis of quantitative traits within population. Simulations indicates that 1/ the number of individuals measured should be as big as possible and 2/ five to six pools of even size maximise the detection power.
To run a GWAlpha analysis, the genotype data need to be formatted in synchronised genotype format (*.sync) that can be generated by the java application mpileup2sync.jar from the popoolation2 package (https://sourceforge.net/projects/popoolation2) once the sequence from the different pools have been preprocessed and merged as mpileup files.

Here is an example of a typical workflow starting from raw reads in fastq format:

```
#Align the reads against a reference genome
[home]$ bwa mem -M -R "@RG\tID:Pop1\tSM:Pool1" ReferenceGenome.fa Seq_Pop1_Pool1_R1.fastq.gz
Seq_Pop1_Pool1__R2.fastq.gz > Seq_Pop1_Pool1.sam 2> /dev/null
#Compress the raw SAM aligned reads into binary BAM
[home]$ samtools view -bT ReferenceGenome.fa Seq_Pop1_Pool1.sam > Seq_Pop1_Pool1.bam
#Sort the reads
[home]$ samtools sort Seq_Pop1_Pool1.bam -o bam > Seq_Pop1_Pool1.sorted
#Rename the sorted BAM
[home]$ mv Seq_Pop1_Pool1.sorted Seq_Pop1_Pool1.bam
#Index the sorted bam
[home]$ samtools index Pop1_Pool1.bam > Pop1_Pool1.bam.bai
#... repeat n times for the n pools …
```

```
#Once the n BAM files are created
#create a *.pile file containing the name of all BAM files for a population, ordered from minimum to maximum trait value.
[home]$ ls Seq_Pop1_*.bam > Pop1.pile
#pileup the n sorted and indexed bam files into a sinple *.mpileup file
[home]$ samtools mpileup -b Pop1.pile > Pop1.mpileup
#create a synchronised geneotype file (*.sync) from the mpileup
[home]$ java -ea -Xmx10g -jar mpileup2sync.jar --input Pop1.mpileup --output Pop1.sync --fastq-type sanger --min-qual 30
```

Installation

Clone the GWAlpha directory from https://github.com/aflevel/GWAlpha

```
[home]$ git clone https://github.com/aflevel/GWAlpha.git
```

Add the GWAlpha directory to the PATH variable

```
[home]$ PATH=$PATH:<path-to-parent-directory>/GWAlpha
```

move to the GWAlpha directory

```
[home]$ cd GWAlpha
```

make sure the GWAlpha scripts are executable

```
[home]$ chmod +x GWAlpha.py GWAlpha_GPS.py GWAlpha_RDC.py GWAlphaPlot.r GWAlpha.sh
```

Scripts and Output Files

GWAlpha.py

Runs the GWAlpha method either using least-square or maximum-likelihood estimation, with or without penalisation. The input files are:

1) a *.sync file formatted as:
   - column 1: reference contig
   - column 2: position in the reference contig
   - column 3: reference allele
   - column >3: allele frequencies in each pool (one column per pool) in the form A-count:T-count:C-count:G-count:N-count:deletion-count.

Example:

```
2L  73   N  1:15:4:269:0:0    0:25:3:406:0:0    0:16:2:320:0:0    0:20:5:393:0:0    1:21:3:409:0:0
2L  119  N  17:257:1:0:0:0    7:369:2:0:0:0     8:251:1:0:0:0     19:353:1:0:0:0    16:344:0:1:0:0
2L  125  N  12:231:0:0:0:0    9:324:0:0:0:0     13:232:0:1:0:1    10:326:0:0:0:0    12:324:0:0:0:0
2L  126  N  14:219:0:0:0:0    12:321:0:0:0:0    10:220:0:0:0:0    12:310:0:0:0:0    11:323:0:0:0:0
2L  135  N  23:0:203:1:0:0    13:0:271:1:0:0    8:1:188:0:0:0     20:0:289:2:0:0    19:2:280:2:0:0
2L  143  N  85:1:1:171:0:0    140:2:0:206:0:0   97:5:1:145:0:0    158:4:0:211:0:0   146:5:0:197:0:0
2L  145  N  7:2:68:172:0:0    7:1:104:208:0:0   1:1:71:157:0:0    3:4:128:207:0:0   4:6:101:196:0:0
2L  149  N  2:69:1:185:0:1    2:110:0:226:0:0   1:73:0:155:0:0    2:121:1:225:0:0   4:102:0:220:0:0
2L  153  N  223:0:40:0:0:13   277:0:57:1:0:19   203:1:40:1:0:13   268:0:56:0:0:34   268:0:60:2:0:18
2L  154  N  217:1:1:41:0:13   273:1:0:57:0:19   213:0:0:42:0:13   265:3:1:57:0:34   280:1:0:59:0:18
```

2) a *_pheno.py file containing the design of the experiment with:

- the name of the trait, Pheno_name
- the standard deviation of the trait, sig
- the minimum value for the trait, MIN
- the maximum value for the trait, MAX
- the cutoff percentiles defining the pools in a [0;1] interval, perc
- the corresponding trait value for the cutoff percentiles, q

Example:

```
Pheno_name= "NQ1-20C";
sig= 48.2944721195801;
MIN= 16.3333333333333;
MAX= 331;
perc=[ 0.064, 0.256, 0.744, 0.936];
Q=[ 64.4166666666667,101,163.5,210.096 ];
```

PLEASE NOTE: The *sync file and the *_pheno.py file need to share the same name (as in the example files: NQ1-20C.sync and NQ1-20C_pheno.py).

Usage:

```
python GWAlpha.py [INPUT_SYNC_FILE] [options]

        INPUT_SYNC_FILE: a *.sync file
        options: ML: perform estimation using maximum likelihood
                 LS: perform estimation using least-square
                 NoP: remove penalisation
                 -MAF [value]: minimum allele frequency threshold, default is 0.03
```

Output:
a GWAlpha_*_out.csv file, example:

```
# Chromosome,Position,Mutation,Alpha,MAF
2L,73,T,0.00498198099211,0.05
2L,119,A,0.00327686076444,0.035
2L,125,A,0.00722404256546,0.042
2L,126,A,0.0154871058116,0.041
2L,135,A,0.0118718624754,0.051
2L,143,A,0.0255044798615,0.397
2L,145,C,0.021811689634,0.323
2L,149,T,0.0151587948777,0.322
2L,153,A,0.016710894771,0.779
2L,153,C,0.00826397992158,0.157
```

GWAlpha_GPS.py

Runs the GWAlpha Genotype-to-Phenotype Simulator. A phenotype is simulated under the additive influence of a predefined set of genetic effects, generating a *simul.sync file and the corresponding *simul_pheno.py file containing all the information needed to run GWAlpha together with the metadata about the simulation parameters. Depending the user-defined options, a *fet.txt and a *glm.txt file can be generated, reporting for each SNP the p-value obtained using a Fisher exact test and a general linear model, respectively.

Usage:

```
GWAlpha_GPS.py [options]...

        options: -h2 : value for the heritability of the trait, continuous ranging from 0 to 1, default is 0.5
                 -nSNP [value]: number of SNP to be simulated, integer, default is 100
                 -nInd [value]: number of individuals in the population, integer, default is 150
                 -nFX [value]: number of genetic effects (ie SNP affecting the phenotype so that their cumulated effect sums to 1), integer, default is 5
                 -lambdaCoverage [value]: lambda parameter of the poisson distribution of the sequencing coverage, continuous positive, default is 40
                 -minFreq [value]: minimun allele frequency for the SNP, continuous from 0 to 1, default is 0.05
                 -nBins [value]: number of bins the phenotypes were pooled into, this option assumes the bins were of even sizes, integer, default is 5
                 -speBins [value]: custom specified cut-offs percentage used to define the bins, this option overwrites the -nBins option, set of n-1 continous to represent the n bins, default (in %) is 6.4 23.5 76.5 93.6
                 -nSim [value]: number of simulation to be carried, integer, default is 1
                 -glm : runs a general linear model between the phenotype and the SNP data, returning a *_glm.txt file with minor allele frequencies, p-values and ranking of associations
                 -fet : runs a Fisher exact test between the two extreme bins of the data, returning a *_fet.txt file with minor allele frequencies, p-values and ranking of test statistics
                 -cmh : runs a Cochran-Mantel-Haenszel test between the two extreme bins of the data, returning a *_cmh.txt file with minor allele frequencies, p-values and ranking of test statistics. Requires -nRep.
                 -nRep : number of replicated experiments to be simulated for the CMH test
```

GWAlpha_RDC.py

Runs the GWAlpha Real Data Converter to compare the performance of different setup of GWAlpha compared to conventional GWAS. Lines/accessions genotypes from a GWAS are pooled based on their trait values, generating a *simul.sync file and the corresponding *simul_pheno.py file containing all the information needed to run GWAlpha. The input files are:

1- a phenotype file in *.csv format formatted as:

- column 1: line/accession identity
- column 2: trait value

Example:

```
accession_id,BLUP_G2P
1,6431.56686182869
100000,1191.81004857314
1026,2844.55901154381
104,2150.87300796283
106,5760.67943441505
1061,11179.5039428838
1062,683.859567401787
1063,9393.41111040432
1064,7362.3588999735
```

2- a genotype file in *.csv format containing the design of the experiment with:

- column 1: Chromosome name
- column 2: position in bp
- column 3 to last: the SNP genotypes coded as 0, homozygous for the reference allele; 1 heterozygous; 2, homozygous for the derived allele.

Example:

```
CHR,POS,1,100000,1026,104,106,1061,1062,1063,1064
1,657,0,2,0,2,2,2,0,2,2,2,0
1,3102,0,2,0,2,2,2,0,2,2,2,0
1,4648,0,2,0,2,2,2,0,2,2,2,0
1,4880,0,0,0,0,0,0,0,0,0,0,0
1,5975,0,0,0,0,0,0,0,0,0,0,0
1,6063,2,0,2,0,0,0,2,0,0,0,2
1,6449,0,0,0,0,0,0,0,0,0,0,0
1,6514,0,2,0,2,2,2,0,2,2,2,0
1,6603,0,2,0,2,2,2,0,2,2,2,0
```

Usage:

```
GWAlpha_RDC.py [INPUT_PHENOTYPE] [INPUT_GENOTYPE] [options]...

        INPUT_PHENOTYPE: a *.csv file with the line/accession identity and trait value
        INPUT_GENOTYPE: a *.csv file with the line/accession identity and genotype values
        options: -nBins : number of bins the phenotypes were pooled into, this option assumes the bins were of even sizes, integer, default is 5
                 -speBins : custom specified cut-offs percentage used to define the bins, this option overwrites the -nBins option, set of n-1 continous to represent the n bins, default (in %) is 6.4 23.5 76.5 93.6
```

GWAlphaPlot.r

Calculates empirical p-values and generates a Manhattan plot of the GWAlpha output. The data can be visualised either as the raw GWAlpha test statistic or as p-values can obtained from the genome-wide empirical distribution of the GWAlpha test statistic.

Usage:

```
GWAlphaPlot.r [GWAlpha_OUPUT] [options]...

        GWAlpha_OUTPUT: Output file of the GWAlpha method in *.csv format
        options: pval: compute the empirical p-values and sets a p-value<0.001 threshold based on the distribution of the GWAlpha test statistic. Default is null.
                        qqplot: draw the quantile-quantile plot between expected and observed p-values (requires the pval option).
```

GWAlpha.sh

Wrapper running the GWAlpha method in parallel across all processors available. It requires the `parallel` library to be installed. The input *.sync file is split across multiple *tmp files, GWAlpha runs on the *tmp files either using least-square or maximum-likelihood estimation and with or without penalisation. The input file are the same as those of the **GWAlpha.py** script. Finally all *tmp outputs are recompiled into a single output, the empirical p-values computed and a Manhattan plot is produced.

Usage:

```
GWAlpha.sh [INPUT_SYNC_FILE] [options]

        INPUT_SYNC_FILE: a *.sync file
        options: ML: perform estimation using maximum likelihood OR
                        LS: perform estimation using least-square
                        NoP: remove penalisation
                        -MAF [value]: minimum allele frequency threshold, default is 0.03
```

Reference:

Alexandre Fournier-Level, Charles Robin, David J. Balding; GWAlpha: genome-wide estimation of additive effects (alpha) based on trait quantile distribution from pool-sequencing experiments. *Bioinformatics* 2016 btw805. doi: 10.1093/bioinformatics/btw805