CCPM Biobank GWAS Pipeline

Release 0.1.0

Tonya Brunetti

Jan 29, 2021

Contents

CHAPTER 1

Installation

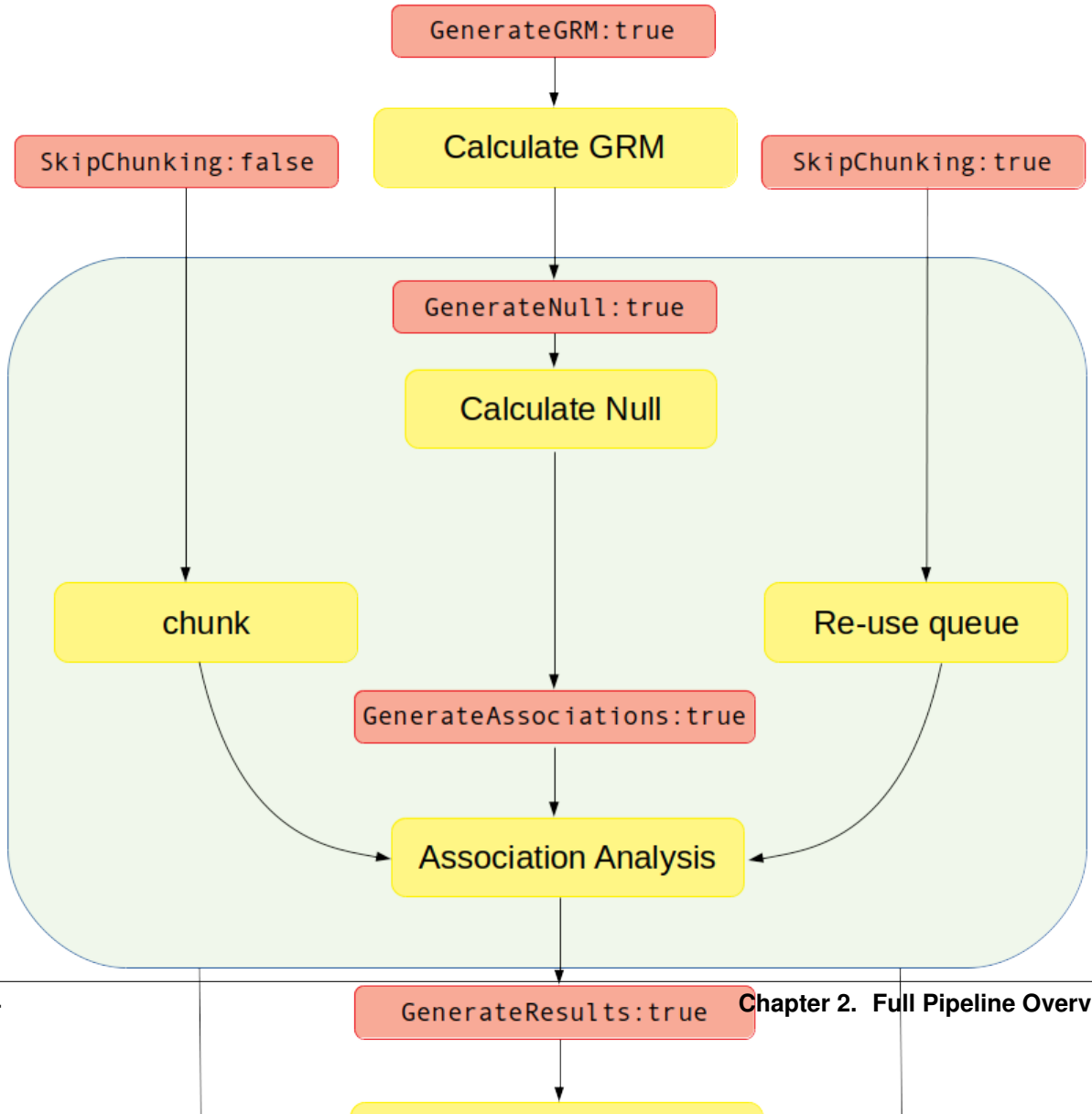
Good news! There is no installation required to use this pipeline and it should be OS agnostic. There are only 2 true system dependencies and most HPCs and shared resources already have these dependencies installed:

- Singularity version >= 3.0
- Golang version >= go1.13.5 (has been tested on versions go1.13.5 and go1.15.2 on a Linux OS).

If not, you can install Singularity and install Golang using the links on your local computer. Both should be available across OS platforms for Windows, MacOS, and Linux.

CHAPTER 2

Full Pipeline Overview



CHAPTER 3

Getting Started

3.1 Deciphering the config file

The config file is a text file that contains key-value mappings to all input parameters and pipeline logic. The file can be broken down into three main parts: the pipeline logic, environment setup, and the user data input options.

3.1.1 Pipeline Logic

There are five keywords that control the logic of the pipeline and all five accept only boolean arguments (true/false):

```
GenerateGRM:true
GenerateNull:true
GenerateAssociations:true
GenerateResults:true
SkipChunking:false
```

3.1.2 Environment Setup

There are three keywords that control the environment setup of the pipeline. It is a requirement to use the container provided as an LSF object on github. This ensures all software is properly versioned and it reduces the complexity of the pipeline by using predefining software paths as well as reducing installation issues that arise with many softawre dependencies.

```
BindPoint:/path/to/bind/container
BindPointTemp:/path/to/tmp/
Container:/path/to/SAIGE_v0.39_CCPM_biobank_singularity_recipe_file_11162020.simg
```

`BindPoint` and `BindPointTemp` are directories to where you want the container to be mounted. `BindPointTemp` allows you to give the container a secondary binding point where all temp files and calculations will be performed, and once finished will move the final files to the `BindPoint`. The files generated in `BindPointTemp` will be deleted after the run completes.

Note: If you do not want to use or do not have a temp directory, please set this parameter to be the same as `BindPoint`.

Warning: **IMPORTANT PLEASE READ!** The container follows the same rules of inheritance as Singularity specifies. This means the `BindPoint` and `BindPointTemp` become the highest point in your directory tree. Thus, you can only access paths, directories, and files if you are able to traverse below the tree from these starting points but not above these entry points from your host system. Therefore, be sure all the paths, directories, and files specified in the config file are contained within the scope of at least one of these two entry points.

3.1.3 User Data Input

The remainder of the keywords are parameters offered to the user and the user can specify paths and options for all or some of these keywords:

```
ChromosomeLengthFile:/path/to/chromosomeLengths.txt
Build:hg38
Chromosomes:1-22
ImputeSuffix:_rsq70_merged_renamed.vcf.gz
ImputeDir:/path/to/imputed/data/directory/
OutDir:/path/to/output/final/results
OutPrefix:myGWAS
PhenoFile:/path/to/phenotype/covariate/file
Plink:/path/to/LDpruned/plink/file/prefix
Trait:binary
Pheno:myPhenotype
InvNorm:FALSE
Covars:PC1,PC2,PC3,PC4,PC5,age,sex
SampleID:SampleIDcol
NThreads:
SparseKin:True
Markers:30
Rel:0.0625
Loco:TRUE
CovTransform:True
VcfField:DS
MAF:0.05
MAC:10
IsDropMissingDosages:FALSE
InfoFile:/path/to/info/file
SaveChunks:false
GrmMAF:0.01
ChunkVariants:1000000
SaveAsTar:false
ImputationFileList:/path/to/list/of/chunked/chromosomes/file.txt
SparseGRM:/path/to/grm/file.mtx
SampleIDFile:/path/to/grm/sample/id/file.mtx.SampleID.txt
NullModelFile:/path/to/null/model/file.rda
VarianceRatioFile:/path/to/variance/ratio/file.txt
AssociationFile:/path/to/concatenated/association/results/file.txt
```

User Data Input: Parameters that are paths to files

See also:

parameters under the section: Full List of User Input Data Parameters. This will prodvide keyword descriptions and types. For file and name formatting of keyword values see *fileFormats*.

3.2 Parameters

This explains in more detail how and when to use each keyword in the config file. For readability, the minimum required parameters are listed for each step. If more than one step will be run (>1 out of 5 possible pipeline logic keywords is set to true), the minimum parameters required is the sum of the minimum required parameters for all true keywords.

3.2.1 Minimum Required Parameters for Each Step

Minimum Requirements to Run Genetic Relatedness Matrix

To calculate the GRM, `GenerateGRM:true` needs to be specified in the config file. If it is set to `false`, the pipeline assumes one of two things:

1. *GRM is not needed because one is provided in the config file from from a previous calculation*
2. *GRM is not needed at all for the scope of the pipeline logic*

Minimum Requirements to Run Null Model

To calculate the null model, `GenerateNull:true` needs to be specified in the config file. If it is set to `false`, the pipeline assumes one of two things:

1. *The null model files are not needed because one is provided in the config file from from a previous calculation*
2. *The null model files are not needed at all for the scope of the pipeline logic*

Minimum Requirements to Run Association Analysis

To calculate the association analyis, `GenerateAssociations:true` needs to be specified in the config file. If it is set to `false`, the pipeline assumes one of two things:

1. *The association file results are not needed because one is provided in the config file from from a previous calculation*
2. *The association analysis files are not needed at all for the pipeline*

Minimum Requirements to Run Results

To calculate the association analyis, `GenerateResults:true` needs to be specified in the config file. If it is set to `false`, the pipeline assumes that no data clean up, data merges, and graphical summaries are needed.

3.2.2 Full List of User Input Data Parameters

Table 1: Definitions and Use

Parameter	Type	Default	Description	Relevance
ChromosomeLengthFile	string		This is a tab-delimited text file that contains the same names as the chromosomes in your dataset followed by the chromosome length. This file can generally be pulled down from the NCBI website under the build you are using (hg19 or hg38). This is used for chunking the imputed data.	if `skipChunking:false`
Build	option: hg19 or hg38	hg38	This determines whether the software needs to be searching for a "chr" before the chomosome name or not	
Chromosomes	string	1-22	This lets the software know which chromosomes you want to use for association analysis. It must be a range. If you want to only run analysis on a single chromsome, the start and end will be the same value. For example: running chromome 2 only will look like 2-2.	if `GenerateAssociation`
ImputeSuffix	string		The full suffix for the software to determine which files in the directory are impuation files. Important, it is assuming the prefix is either a chromosome number (hg19) or the string chr followed by the chromosome number (hg38)	if `GenerateAssociation` and `skipChunking:false`
ImputeDir	string		Full path to directory where imputed results are located	if `GenerateAssociation`
OutDir	string		Full path to directory where final results should be transferred	all
OutPrefix	string		string (no whitespace or special characters) to prefix to the output files generated	all
PhenoFile	string		Full path to tab-delimted phenotype file containing sample IDs, phenotypes, and covariates, with whatever string of headers you choose. NO WHITESPACES in header names.	If `:GenerateNull:true`
Plink	string		Full path to the directory and plink file prefix (dropping the suffix .bed,.bim,.fam) to an LD-pruned set of data to be used to generate GRM relatedness and to select random markers from for the variance ratio value	If `GenerateGRM:true` or `GenerateNull:true`

continues on next page

Table 1 – continued from previous page

Parameter	Type	Default	Description	Relevance
Trait	option: binary or quantitative		Based upon your association phenotype. If binary, all values will be 0/1/NA, if quantitivate all phenotype traits to be analyzed will be continuous or have numeric quantitative meaning	if `GenerateNull:true` or `GenerateResults:tru`
Pheno	string		The exact name (case-sensitive) of the phenotype to be analyzed in your PhenoFile. Must be present in PhenoFile.	if `GenerateNull:true` or `GenerateResults:tru`
InvNorm	boolean	FALSE	This applies to the phenotype of interest to be analzyed and whether to perform an inverse normalization. For binary traits, this should be set to FALSE and for quantitative traits, set this to TRUE.	if `GenerateNull:true`
Covars	comma-separated list		A comma-separated list (no whitespaces) of all the covariate names to regress out in the model. These variables need to be in your PhenoFile.	if `GenerateNull:true` or `GenerateResults:tru`
SampleID	string		A string (no whitespaces) that is contained in the header of your PhenoFile. This is the sampleID names and they must be the same names as listed in the PhenoFile, Imputation Files, and Plink Files.	if `GenerateNull:true`
Nthreads	int		Strongly Recommned to leave this blank! If left blank, it will auto-decect available resources and scale steps automatically on the back-end. By specifiying the threads it tells the program to use max Nthreads for parallelization and concurrency.	all
SparseKin	boolean	TRUE	If set to true, takes advantage of the sparsity of the GRM, otherwise will not use the sparsity to make assessments	if `GenerateNull:true`
Markers	int	30	The number of random markers selected from the LD-pruned plink file to estimate the variance ratio component in the null model. Warning! This number increases time linearly	if `GenerateNull:true`
Rel	float	0.0625	A float between 0.0-1.0. This is the threshold in kinship estimate to consider someone related. Anything below this value will be treated as an unrelated individual in the pairwise comparison and calculation for the sparse GRM.	if `GenerateGRM:true`
Loco	boolean	TRUE	Leave-One-Chromosome-Out method. Warning – Setting this to true, increases the time complexity of the algorithm.	if `GenerateNull:true`a :code:`GenerateAsso
CovTransform	boolean	TRUE	Recommended to set to true. It is a QR decomposition that aids in the covergence of the null model.	if `GenerateNull:true`

continues on next page

Table 1 – continued from previous page

Parameter	Type	Default	Description	Relevance
VcfField	option: DS or GT	DS	This determines what metric to base association upon. DS = dosages and GT = genotypes. If you have genotypes only, i.e. chip data withouth dosage calculations, DS cannot be used!	if `GenerateAssociation`
MAF	float	0.05	Float between 0.0-0.50 that specifies the cutoff to be considered a common snp or a rare snp. For example, keeping this to the default of 0.05 will assume common snps are defined as those with a minor allele frequency >5% and that rare snps are defined as those with a minor allele frequency 5%. THIS IS NOT A FILTER!	if `GenerateResults:tru`
MAC	int	10	A filter applied to the cleaned association results to remove snps that have low minor allele counts. Default recommendation is to set this to 10.	if `GenerateResults:tru`
IsDropMissingDosages	boolean	FALSE		if `GenerateAssociation`
InfoFile	string		Path to the info file. This file contains snps information pertaining to chromosome, positions, genotype/imputation status, R2, ER2 values. For formatting of this file please refer to <----------→	if `GenerateResults:tru`
SaveChunks	boolean	TRUE	Specifies whether to save the chunked files and the queue list for future use.	if `GenerateAssociation` and `skipChunking:false`
GrmMAF	float	0.01	The minor allele frequency threshold for a snp to be included in the GRM calculation based on the LD-pruned plink file. For example, if set to 0.01 this means any snp with a MAF > 0.01 wil be used to calculate relatedness in the GRM.	if `GenerateGRM:true`
ChunkVariants	int	1000000	The window of base pairs to chunk imputation files. It is recommended to keep this at the default of 1000000.	if `GenerateNull:true` and `SkipChunking:false`
SaveAsTar	boolean	FALSE		all
ImputationFileList	string		Ends in _chunkedImputationQueue.txt	if `GenerateAssociation` and `skipChunking:true`
SparseGRM	string		Ends in .sparseGRM.mtx	if `GenerateGRM:false` and `GenerateNull:true`
SampleIDFile	string		Ends in sparseGRM.mtx.sampleIDs.txt	if `GenerateGRM:false` and `GenerateNull:true`

continues on next page

Table 1 – continued from previous page

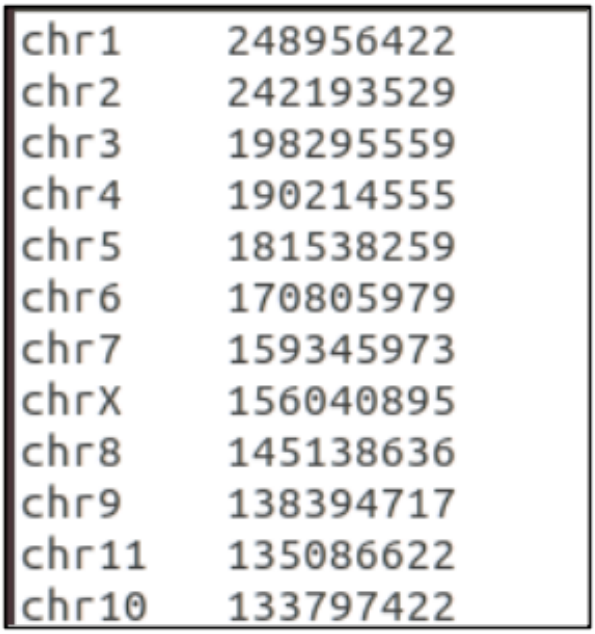
Parameter	Type	Default	Description	Relevance
NullModelFile	string		Ends in .rda	if `GenerateNull:false` and `GenerateAssociation`
VarianceRatioFile	string		Ends in .varianceRatio.txt	if `GenerateNull:false` and `GenerateAssociation`
AssociationFile	string		Ends in _SNPassociationAnalysis.txt	If `GenerateAssociation` and `GenerateResults:tru`

3.3 Formatting the Required Files

The pipeline does require some formatting for file names and contents within files to be present. This section explains the file format expectations, as well as file name expectations.

3.3.1 Parameter: ChromosomeLengthFile

The `ChromsomeLengthFile` parameter is a file that can be downloaded and modified from sources such as NCBI and UCSC. It should be a tab-delimted file with chromosomes 1-22 listed in the first column and the length of the chromosome in the second column. This file contains **no header**. The build you select should be based upon the build used for your imputation file. If only using genotyping file, base this file on the genotype build.



Example snippet for hg38

- 1st column is chromosome
- 2nd column is length

Warning: hg19 vs hg38 formatting Depending on the build you choose you must format your file accordingly.

- *hg19:* omit "chr" and should strictly be integer values between 1-22 for the first column.
- *hg38:* include the preceding "chr" string (no spaces) for integer values between 1-22 for the first column.

3.3.2 Parameter: ImputeSuffix

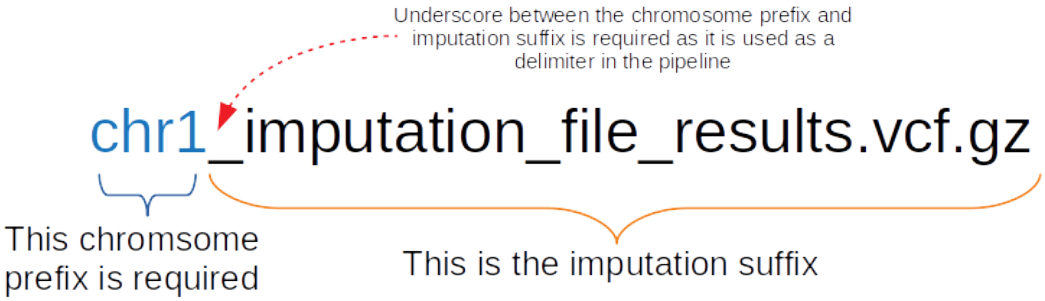
This string is the suffix of your imputation file. The imputation files require to be named and split in a particular way.

First, all imputation files follow these rules:

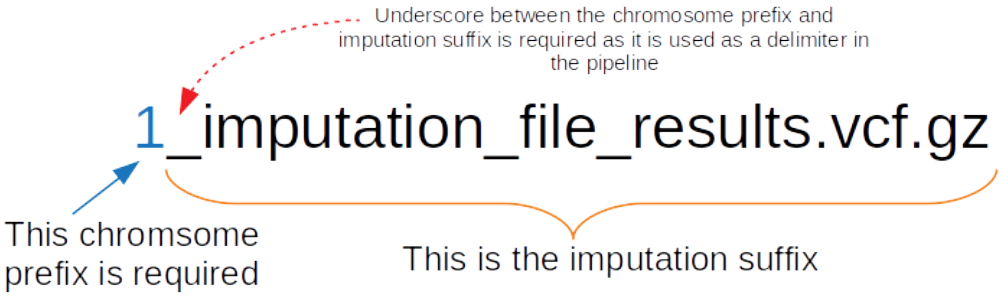
- split by chromosome
- sample names are the samples names used in `GenerateGRM` and `GenerateNull` (order doesn't need to be maintained)
- gzipped vcf files, and
- have the matching tabix index ending in the imputation file name followed by the .tbi string

This is how each imputation file needs to be named:

for hg38



for hg19



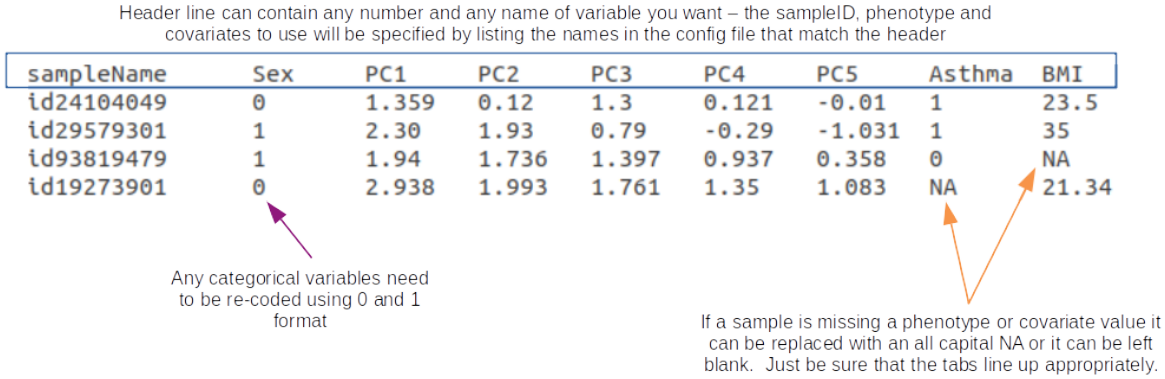
Warning: hg19 vs hg38 formatting Depending on the build you choose you must format your file name accordingly.

- *hg19:* omit "chr" and strictly be integer values between 1-22 as the prefix, followed by a required underscore, following by remaining string.
- *hg38:* include the preceding "chr" string (no spaces) for integer values between 1-22 as the prefix, followed by a required underscore, following by remaining string.

3.3.3 Parameter: PhenoFile

This is tab-delimited txt file that contains all the sample IDs (must be the same IDs used in the plink file, imputation file, GRM, and null model -- order agnostic). In addition to the sample IDs, it also contains any phenotype(s) you may want to run and any covariates you may want to use, although the user is not required to use everything listed in the header/file.

Below is an example of a tab-delimited `PhenoFile`. Again, none of these header names are required, however, there needs to be a header variable at minimum that denotes the sample ID and a phenotype to analyze.



Note: It is worthwhile to generate a single PhenoFile that contains many phenotypes and covariates you may want to analyze for the sample set. Within the config file, the user can specify which single phenotype to run and which covariates to run. That way, the user can run several jobs in parallel using the same PhenoFile but just changing the trait, phenotype, covariates, and invNorm parameters without having to change anything else.

3.3.4 Parameter: Plink

3.3.5 Parameter: Pheno

3.3.6 Parameter: Covars

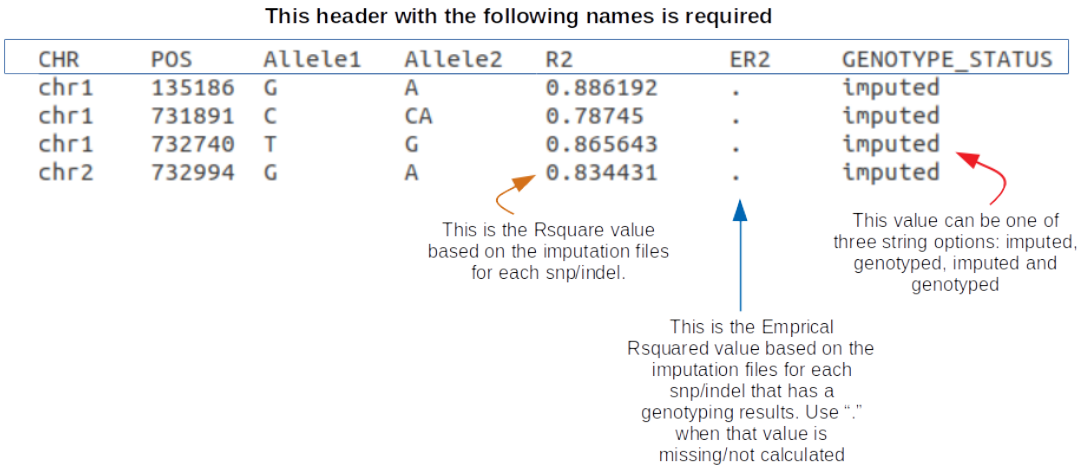
3.3.7 Parameter: SampleID

3.3.8 Parameter: InfoFile

This is a tab-delimited file that contains information about all snps present in the genotyping chip and imputation files. This file is really only necessary if `GenerateResults:` is set to `true`.

The following headers must be present:

- CHR
- POS
- Allele1
- Allele2
- R2
- ER2
- GENOTYPE_STATUS



3.3.9 Parameter: ImputationFileList

3.3.10 Parameter: SparseGRM

3.3.11 Parameter: SampleIDFile

3.3.12 Parameter: NullModelFile

3.3.13 Parameter: VarianceRatioFile

3.3.14 Parameter: AssociationFile

3.4 Output Generated

3.5 Parsing Through StdErr and StdOut

When using the pipeline with the container and go binary file, all standard out messages and errors are reported in the generated log file. It is important to note, that since the pipeline automatically runs steps concurrently when possible, the order of the log files may show some steps running as dependencies become available and based on the threads available. Also, this standard out is used by SAIGE as well, so anything that does not follow the format below is a results of the software calculations generated by SAIGE, and not the pipeline.

The following shows how messages are formatted in the log output:

3.5.1 Logging Levels

There are 4 different levels or types of debugging and error handling. Some are meant to inform, while other are meant to stop the pipeline if an error occurs that cannot be resolved.

Level	Interpretation
UPDATE!	To inform the user of the status of the pipeline.
CONFIRMED!	To let user know a check or validation has been passed.
WARNING!	There may be a problem but it continues through the pipeline and skips the issue.
ERROR!	There is definitely a problem and forces the pipeline to prematurely exit.

By using the levels and function names, one can `grep` through the log file to find specific messages related to functions or error types.

3.5.2 Exit Status Codes

Although I am not a huge fan of relying on exit status codes, for those that use Gnu Parallel or other software that records exit status to determine if a job was successful or not, I have coded up a few exit codes so that if an error does occur the number can tell the user which function is responsible for the error. This make troubleshooting and errors easier to identify.

functions	Exit Status
func(main)	return 42
func(chunk)	return 10
func(smallerChunk)	None
func(processing)	None
func(nullModel)	None
func(associationAnalysis)	None
func(checkInput)	return 5
func(saveResults)	return 99
func(saveQueue)	None
func(usePrevChunks)	return 17
func(findElement)	return 20
func(parser)	return 3

CHAPTER 4

Quick Start and Examples

4.1 Tutorials and Examples

4.1.1 Quick Start Command

In order to run the pipeline, open a shell or bash prompt (or batch script for a job-scheduler) and type:

```
$ ./CCPM_GWAS_pipeline myConfigFile.txt
```

4.1.2 Tutorials

Tutorial: Full Pipeline with Binary Trait

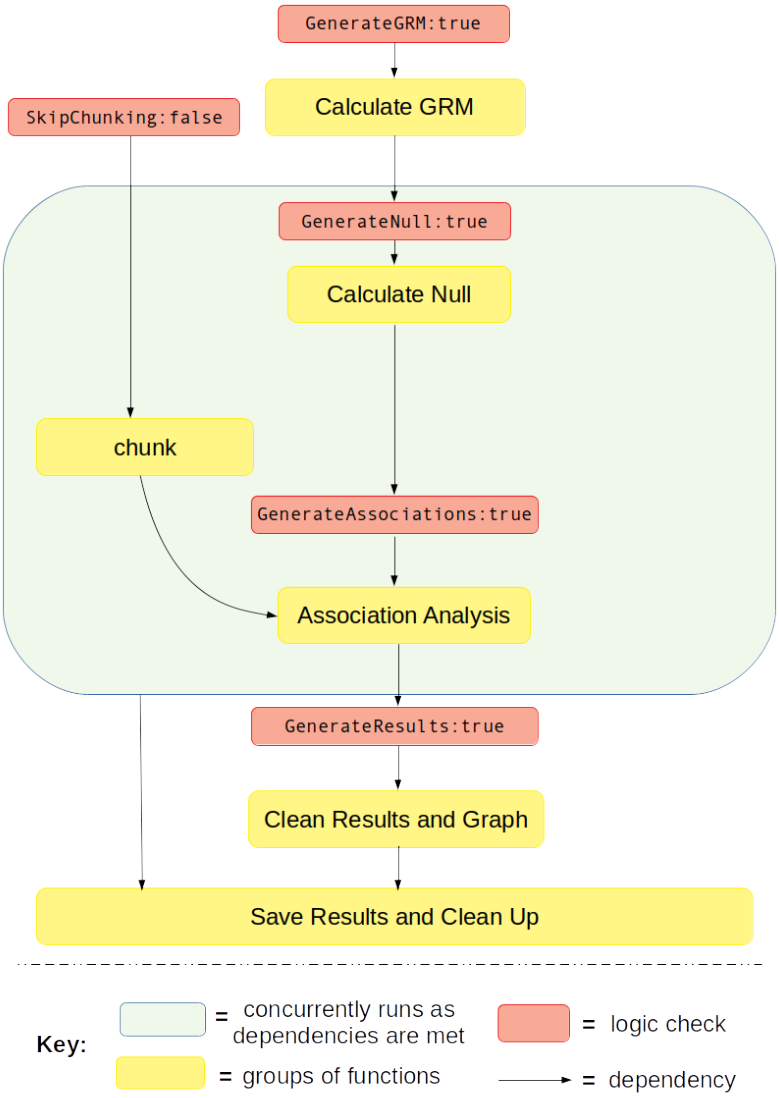
This example will walk you through how to run the full pipeline when the phenotype for association analysis if for a binary trait. It will guide you through how to properly set the logic, remind you to set the environment, list all the additional files you need, and finally which user parameters need to be set.

Section: Logic and Overview

Full pipline means you want to run every component of the pipeline from beginning to end in one go, without re-using any previously calculated data from the pipeline. This is analagous to setting the pipeline logic kewords to the following:

```
GenerateGRM:true
GenerateNull:true
GenerateAssociations:true
GenerateResults:true
SkipChunking:false
```

If the pipeline is set to the above logic, the following workflow will be executed:



Section: Step-by-Step Tutorial

STEP 1: Set the logic

As stated about above, open your config file (.txt) and make sure the logic is set to the following:

```
GenerateGRM:true
GenerateNull:true
GenerateAssociations:true
GenerateResults:true
SkipChunking:false
```

STEP 2: Set the environment

Open your config file (.txt) and make sure you set the path to where the bind point, temp bind point, and container image are located. I suggest you set the `BindPoint` keyword to the same path as where the container is located to avoid any confusion. If you have a tmp directory you want to use as scratch space, set that path as well. If this doesn't exist or you choose not to use it, set the keyword `BindPointTemp` to be the same as the path listed in the keyword `BindPoint`.

```
BindPoint:/path/to/bind/container
BindPointTemp:/path/to/tmp/
Container:/path/to/SAIGE_v0.39_CCPM_biobank_singularity_recipe_file_11162020.simg
```

STEP 3: Ensure you have all the files required

For running the full pipeline, including chunking the imputation files, you will need access to the following files:

1. **LD-pruned plink file**
    - used for when logic parameters `GenerateGRM` is set to true and/or `GenerateNull` is set to true.
    - fulfills parameter `Plink`
    - see *Parameter: Plink* for formatting
2. **phenotype and covariates file**
    - used for when logic parameter `GenerateNull` is set to true
    - fulfills parameter `PhenoFile`
    - see *Parameter: PhenoFile* for formatting
3. **chromosome lengths file**
    - used for when logic parameter, `SkipChunking` is set to true.
    - fulfills parameter `ChromosomeLengthFile`
    - see *Parameter: ChromosomeLengthFile* for formatting
4. **imputation files properly named and formatted or genotype files formatted in same way as imputation files**
    - used for when logic paramters `SkipChunking` is set to true and/or `GenerateAssociations` is set to true.
    - fulfills parameter `ImputeSuffix`
    - see *Parameter: ImputeSuffix* for formatting
5. **SNP information file**
    - use for when logic parameter `GenerateResults` is set to true
    - fulfills parameter :code: *InfoFile*
    - see *Parameter: InfoFile* for formatting

See also:

For a complete list of files and name formatting of keyword values listed in the config file see *Formatting the Required Files*.

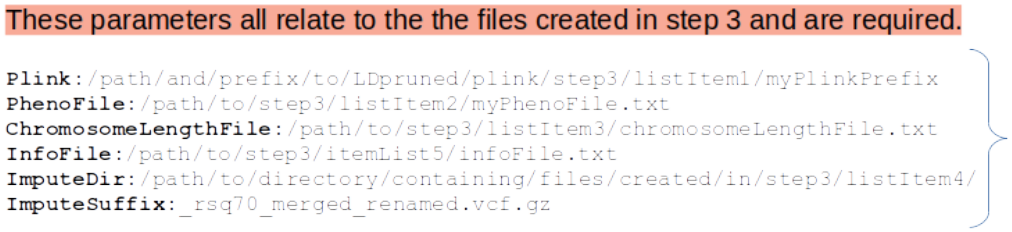
STEP 4: Set the path and values to all the required input parameters

Now that you have all the required files, it is time to set the values and locations within your config file using the keywords expected. Here are the required keywords and how to specify them:

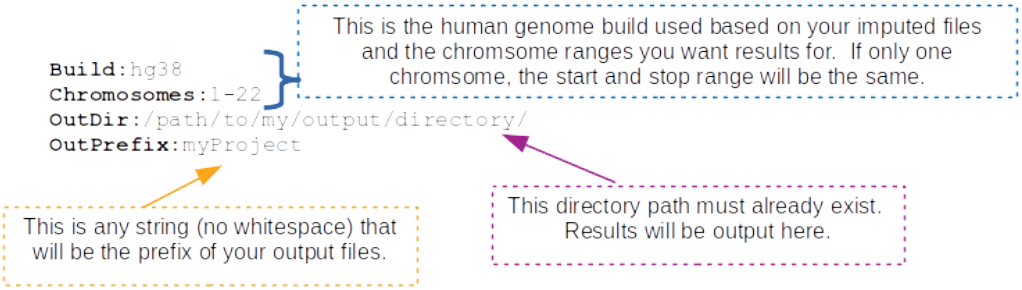
1. This `RUNTYPE` parameter need to just be here for placeholder purposes, however it is required. It has no impact on the pipeline, except as a header to check that it exists.

```
RUNTYPE:FULL
```

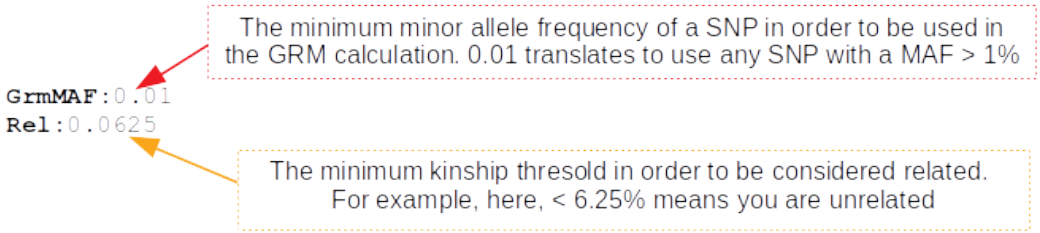
2. The next set of parameters are the keywords that relate to file inputs:



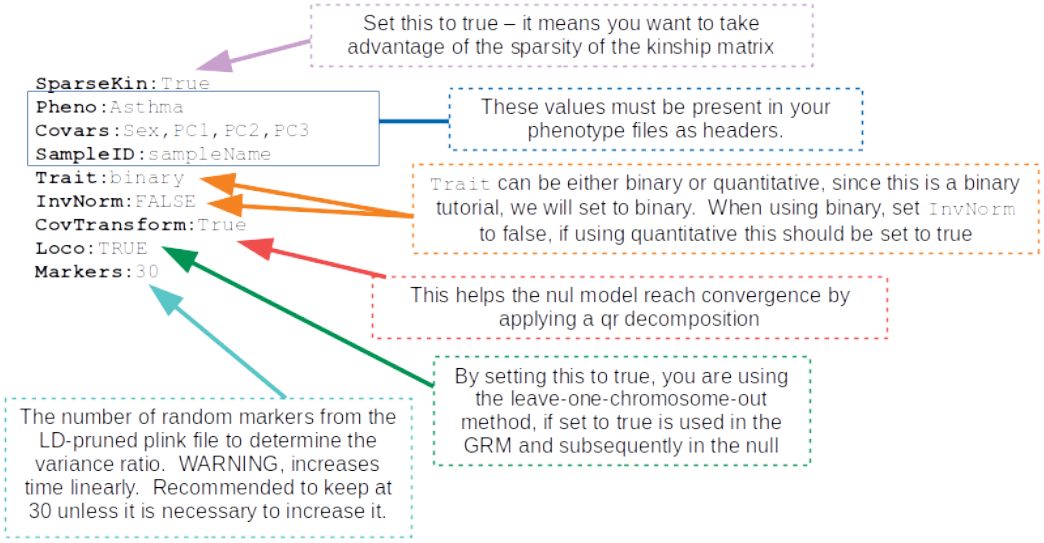
3. Here are some required general keyword parameters that need to be set:



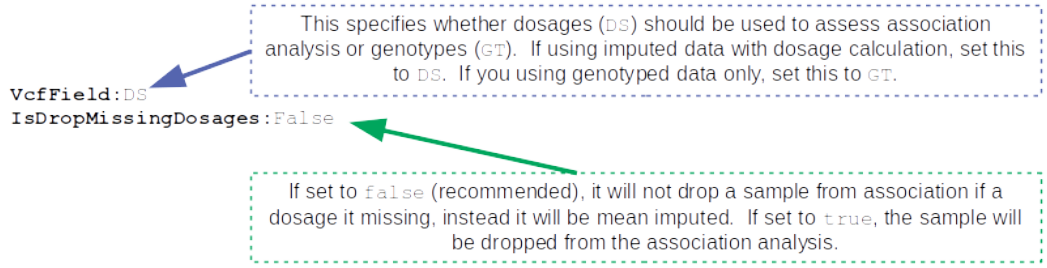
4. The following two sets of keyword parameters affect the GRM step, i.e. `GenerateGRM:true` :



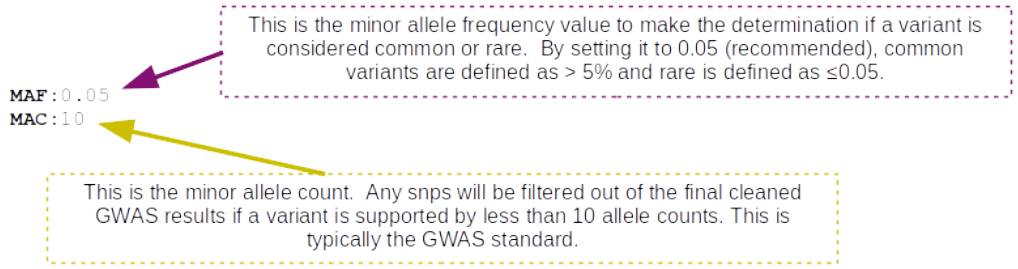
5. The following sets of keyword parameters affect the null model step, i.e. `GenerateNull:true` :



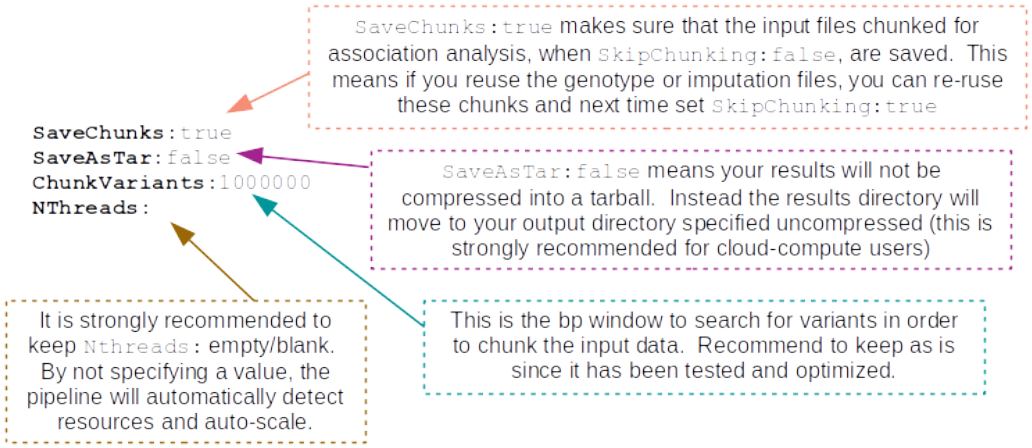
6. The following sets of keyword parameters affect the association analysis step, i.e. `GenerateAssociations:true`:



7. The following sets of keyword parameters affect the results step, i.e. `GenerateResults:true`:



8. These parameters I recommend to keep as is, unless you are familiar with the pipeline and have a reason to change them:



STEP 5: Running the pipeline

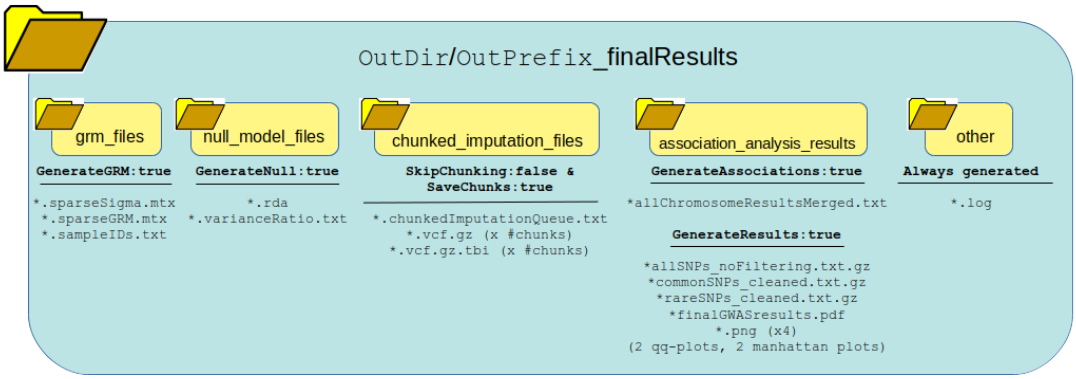
To run the pipeline make sure the files are all accessible to the container relative to the bindpoint.

Once all the files are accessible and the config is ready, the following command will run the pipeline:

```
$ ./CCPM_GWAS_pipeline myConfigFile.txt
```

Section: Generated Output

The following graphic shows how all the data generated from running the logic of this pipeline will be organized and which files are present. One thing to notice is the list of files generated in each directory based on whether the pipeline logic is set to `true` or `false`. Many of these outputs and be re-used under certain circumstances to save time and bypass running certain steps of the pipeline in the next run.

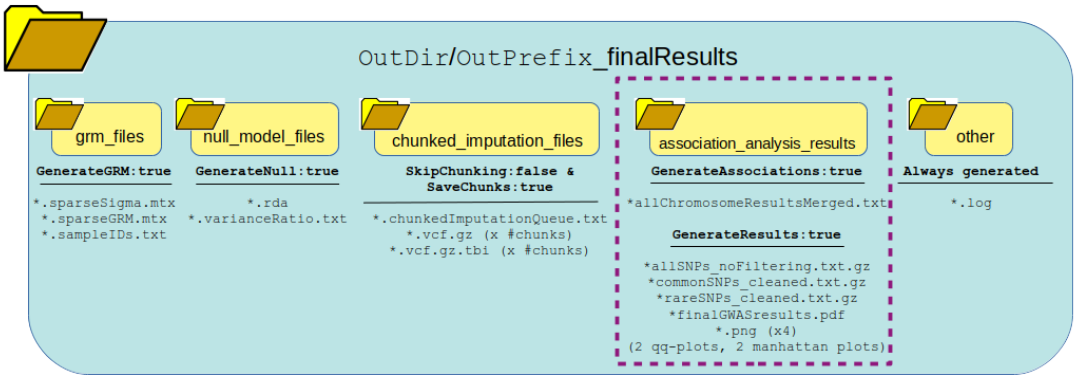


Warning: **IMPORTANT PLEASE READ!** Although the pipeline tries its best to not generate output as critical errors occur, this is not always the case. It is particularly important to parse through the standard error output, as well as the log file produced in the `other` directory of your output directory. The log file can be quite large, therefore, it is recommended to use `grep` to seach for keywords. I would recommend the following: `grep -i "err" other/*.log`, `grep -i "warn" other/*.log`, and `grep -i "exit" other/.*log`. Also, please see the note below, for additional ways to parse the log file.

See also:

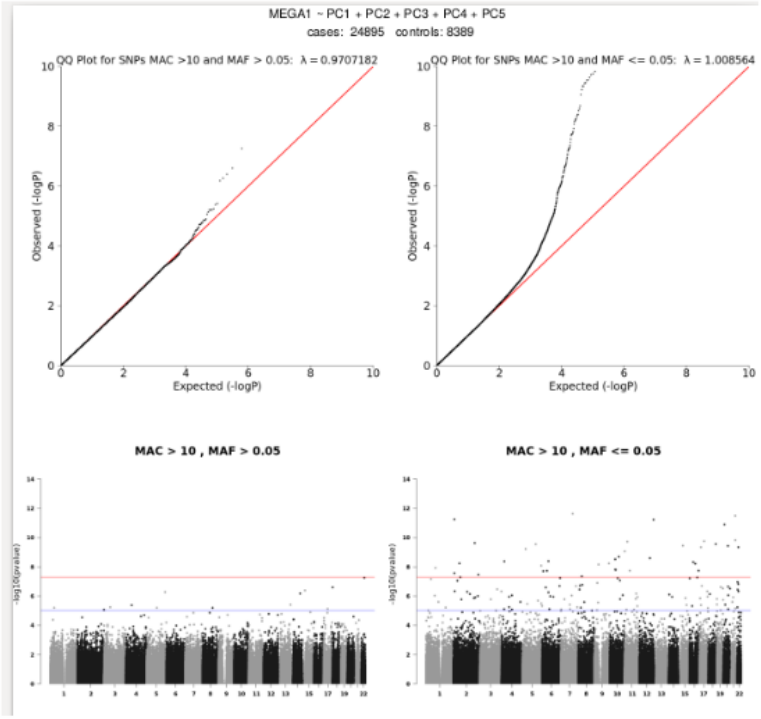
For a interpreting and searching the log files for potential pipeline errors, see *Parsing Through StdErr and StdOut*.

Once it is confirmed that the error and log files ran successfully without major errors, the results and files are ready for viewing. The directory of highest interest will be the `association_analysis_results` directory.



When `GenerateAssociations:true`, the pipeline generates raw association analysis data of all SNPs. This set of data does have the allele flips in place, it is uncleaned and unfiltered, unannotated, lacking additional calculations and will not generate any visuals. The file is the `*allChromosomeResultsMerged.txt` files.

Now, when `GenerateResults:true`, it takes that file, `*allChromosomeResultsMerged.txt` and applies allele flips to ensure allele2 is always the minor allele, cleans the data using the `MAC` filter, annotates the data with ER2, R2, and whether the SNP/Indel is imputed/genotyed/both. This will also split your data in to common vs rare variants as defined by `MAF` and generate qqplots and Manhattan plots for each. The plots are put in a pdf report, `*finalGWASresults.pdf`. Each plot is also reported as individual pngs so they can easily be embedded into presentations and documents. Here is an example of one of the pdf reports:



If you open any of the `.txt.gz` files in located in the `association_analysis_results` directory produced

by `GenerateResults:true`, the following headers are listed for all the SNPs/indels, in a tab-delimited file:

Header	Definintion
CHR	chromosome name/ID
POS	position in chromosome, build is based on input imputation file build
majorAllele	major allele based on the allele frequency of your project
minorAllele	minor allele based on the allele frequency of your project
SNPID	snpID/name
BETA	beta value
SE	standard error of the beta
OR	odds ratio
LogOR	log(odds ratio)
Lower95OR	the lower 95% confidence interval of the odds ratio
Upper95OR	the upper 95% confidence interval of the odds ratio
MAF	minor allele frequency
MAC	minor allele count
p.value	pvalue significance of association (note, for GWAS sig p<5e-08)
N	total samples used in this snp analysis
N.Cases	total number of case samples
N.Controls	total number of control samples
casesHomMinor	total number of cases that have homozygous minor alleles
casesHet	total number of cases that are heterozygous
controlHomMinor	total number of controls that have homozygous minor alleles
controlHet	total number of controls that are heterozygous
negLog10pvalue	-log10(p.value)
R2	imputation R2 quality
ER2	empirical R2 quality -- only for genotyped variants
GENTOYPE_STATUS	whether a SNP is genotyped/imputed/both

Tutorial: Generate GRM Only

This example will show you how to generate the GRM step only. It will guide you through how to properly set the logic, remind you to set the environment, list all the additional files you need, and finally which user parameters need to be set.

Section: Logic and Overview

GRM only means you want to run the GRM step. This is analagous to setting the pipeline logic kewords to the following:

Note, `SkipChunking` *can be set to either* `true` *or* `false` *because it is only used if* `GenerateAssociation` *is set to* `true`.

```
GenerateGRM:true
GenerateNull:false
GenerateAssociations:false
GenerateResults:false
SkipChunking:false
```

Section: Step-by-Step Tutorial

Section: Generated Output

See also:

For a interpreting and searching the log files for potential pipeline errors, see *Parsing Through StdErr and StdOut*.

Tutorial: Generate Null Model only

This example will show you how to generate the null model step only. It will guide you through how to properly set the logic, remind you to set the environment, list all the additional files you need, and finally which user parameters need to be set.

Section: Logic and Overview

Null Model only means you want to run the Null Model only. It makes an assumption that you already have the GRM pre-calculated and want to re-use it in this step by setting the keywords `SparseGRM` and `SampleIDFile` located in the config file. These two files are the result of running `GenerateGRM:true`.

Choosing to run just the null model generation step is analagous to setting the pipeline logic kewords to the following:

```
GenerateGRM:false
GenerateNull:true
GenerateAssociations:false
GenerateResults:false
SkipChunking:false
```

`SkipChunking` *can be set to either* `true` *or* `false` *because it is only used if* `GenerateAssociation` *is set to* `true`.

Section: Step-by-Step Tutorial

Section: Generated Output

See also:

For a interpreting and searching the log files for potential pipeline errors, see *Parsing Through StdErr and StdOut*.

Tutorial: Generate Association Analysis Only

This example will show you how to generate the association analysis step only. It will guide you through how to properly set the logic, remind you to set the environment, list all the additional files you need, and finally which user parameters need to be set.

Section: Logic and Overview

Association Analysis only means you only want to run the association anlysis. It makes an assumption that you already have the null model file (.rda) pre-calculated and have a pre-calculate variance ratio file (.varianceRatio.txt) and want to re-use/use it in this step by setting the keywords `NullModelFile` and `VarianceRatioFile` located in the config file. These two files are the result of running `GenerateNull:true`.

Choosing to run just the association analysis step is analagous to setting the pipeline logic kewords to the following:

```
GenerateGRM:false
GenerateNull:false
GenerateAssociations:true
GenerateResults:false
```

When `GenerateAssociations:true`, the `SkipChunking` logic comes into play.

Note: This step produces the raw associaions results concatenated into a file. It does not clean up the data, perform the proper flips, or generate graphs/figures. If you want the raw data in addition to the previously mentioned actions, be sure to also set `GenerateResults:true`.

Section: Step-by-Step Tutorial

Section: Generated Output

See also:

For a interpreting and searching the log files for potential pipeline errors, see *Parsing Through StdErr and StdOut*.

4.1.3 FAQs

CHAPTER 5

FAQs

CHAPTER 6

Acknowledgements

The most important acknowledgment is for the group of developers, investigaters, scientists, and researchers of SAIGE . This pipline is honestly just an automated wrapper to most of their work. They have an excellent github page here at https://github.com/weizhouUMICH/SAIGE -- The entire team is very responsive to github questions. They are tirelessy working on newer and more complex versions of SAIGE, and for that I must thank them. It is an amazing piece of software! Please read their publication:

Zhou, W., Nielsen, J. B., Fritsche, L. G., Dey, R., Gabrielsen, M. E., Wolford, B. N., LeFaive, J., VandeHaar, P., Gagliano, S. A., Gifford, A., Bastarache, L. A., Wei, W. Q., Denny, J. C., Lin, M., Hveem, K., Kang, H. M., Abecasis, G. R., Willer, C. J., & Lee, S. (2018). Efficiently controlling for case-control imbalance and sample relatedness in large-scale genetic association studies. Nature genetics, 50(9), 1335–1341. https://doi.org/10.1038/s41588-018-0184-y

All of this work could not be done without the full support of the Colorado Center for Personalized Medicine (CCPM) under the guidance of their Biobank, and the Translation Informatics Services (TIS) sector, among several input from exerpienced scientists and professor within CCPM whose expertise is in GWAS.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search