
MVtest API Documentation

Release 1.0.0

Todd Edwards, Chun Li and Eric Torstenson

March 03, 2016

1	MVtest API Libraries	1
1.1	meanvar package	1
1.1.1	meanvar.mv_esteq module	1
1.1.2	meanvar.mvresult module	2
1.1.3	meanvar.mvstandardizer module	3
1.1.4	meanvar.simple_timer module	3
1.1.5	Module contents	4
1.2	pygwas package	4
1.2.1	pygwas.bed_parser module	4
1.2.2	pygwas.boundary module	5
1.2.3	pygwas.data_parser module	7
1.2.4	pygwas.exceptions module	8
1.2.5	pygwas.impute_parser module	9
1.2.6	pygwas.locus module	10
1.2.7	pygwas.mach_parser module	11
1.2.8	pygwas.parsed_locus module	13
1.2.9	pygwas.pedigree_parser module	13
1.2.10	pygwas.pheno_covar module	14
1.2.11	pygwas.snp_boundary_check module	16
1.2.12	pygwas.standardizer module	16
1.2.13	pygwas.transposed_pedigree_parser module	18
1.2.14	Module contents	18
	Python Module Index	21
	Index	23

MVTEST API LIBRARIES

The following represents the API functionality associated with the meanvar application which includes a single interface for extracting data from each of the supported file types (pygwas). The contents below are only of interest for those who wish to extend MVtest or utilize PyGWAS in their own GWAS analysis programs.

MVtest is released under the Gnu Public license version 3 (<http://www.gnu.org/licenses/gpl-3.0.en.html>).

1.1 meanvar package

The meanvar library provides the base functionality for the MVtest method. It depends on PyGWAS functionality.

1.1.1 meanvar.mv_esteq module

`meanvar.mv_esteq.MeanVarEstEQ(y, x, covariates, tol=1e-08)`

Perform the mean var calculation using estimated equations

Parameters

- *y* -- Outcomes
- *x* -- [genotypes, cov1, ..., covN]
- *tol* -- convergence criterion

`meanvar.mv_esteq.RunAnalysis(dataset, pheno_covar)`

Run the actual analysis on all valid loci for each phenotype

Parameters

- *dataset* -- GWAS parser object
- *pheno_covar* -- holds all of the variables

This acts as a standard iterator, returning a single MVResult for each locus/phenotype combination.

Missing is evaluated as anything missing in any of the phenotype, covariate(s) or genotype

`meanvar.mv_esteq.RunMeanVar(pheno, geno, covar=//)`

Setup and execute the mean var calculation.

Parameters

- *pheno* -- Phenotype data (one phenotype at a time)
- *geno* -- SNP data (might be genotypes, or dosages, etc)

- covar -- List of covariate data

It is possible that the optimization will fail to converge. Such cases are stripped of data, but are still reported to alert the user that there were problems with the data.

1.1.2 meanvar.mvresult module

```
class meanvar.mvresult.MVResult(chr, pos, rsid, maj, min, eff_alcount, non_miss_count, p_mvtest,
                                ph_label, beta_values, pvalues, stderrors, maf, covar_labels=[],
                                lm=-1, runtime=-1)
```

Bases: object

Result associated with a single locus/phenotype execution

beta_pvalues = None
list of beta pvalues

beta_stderr = None
list of std errors

betas = None
list of beta values

chr = None
Chromosome

covar_labels = None
Covariate labels used for analysis

eff_alcount = None
Total count of effect alleles

lmpv = None
LM

maf = None
minor allele frequency

maj_allele = None
Major allele (A,C,G,T, etc)

min_allele = None
Minor allele

non_miss = None
non missing count

p_mvtest = None
mvtest's pvalue

p_variance

ph_label = None
current phenotype label

pos = None
BP position

print_header(f=<open file '<stdout>', mode 'w'>, verbose=False)
Prints header to f (will write header based on verbose)

Parameters

- `f` -- stream to print output
- `verbose` -- print all data or only the most important parts?

`print_result(f=<open file '<stdout>', mode 'w'>, verbose=False)`
 Print result to `f`

Parameters

- `f` -- stream to print output
- `verbose` -- print all data or only the most important parts?

`rsid = None`
 RSID

`runtime = None`
 number of seconds analysis took to complete

`stringify(value)`

1.1.3 meanvar.mvstandardizer module

`class meanvar.mvstandardizer.Standardizer(pc)`
 Bases: `pygwas.standardizer.StandardizedVariable`

Optional plugin object that can be used to standardize covariate and phenotype data.

Many algorithms require that input be standardized in some way in order to work properly, however, rescaling the results is algorithm specific. In order to facilitate this situation, application authors can write up application specific Standardization objects for use with the data parsers.

`destandardize(estimated, se, **kwargs)`
 Revert the betas and variance components back to the original scale.

`standardize()`
 Standardize the variables within a range [-1.0 and 1.0]

This replaces the local copies of this data. When it's time to scale back, use `destandardize` from the datasource for that.

1.1.4 meanvar.simple_timer module

`class meanvar.simple_timer.SimpleTimer`
 Simple abstraction to allow for basic timing.

`report(msg, do_reset=False, file=<open file '<stdout>', mode 'w'>)`
 Print to stdout msg followed by the runtime.

When true, `do_reset` will result in a reset of start time.

`reset()`
 Reset start time

`result(msg, do_reset=False)`
 Return log message containing ellapsed time as a string.

When true, `do_reset` will result in a reset of start time.

`runtime()`
 Return ellapsed time and reset start.

1.1.5 Module contents

1.2 pygwas package

PyGWAS provides a singular interface for using several GWAS data formats such as Pedigree (ped), Translated Pedigree (tped), Binary Pedigree (bed) and two common imputed formats, IMPUTE and MACH as well as each of the accompanying files such as marker or family data. Support for plink style phenotype and covariate formatted files are also provided.

1.2.1 pygwas.bed_parser module

```
class pygwas.bed_parser.Parser(fam, bim, bed)
```

Bases: *pygwas.transposed_pedigree_parser.Parser*

`ReportConfiguration(file)`
Report configuration for logging purposes.

Parameters file -- Destination for report details

Returns None

alleles = None
Alleles for each locus

bed_file = None
Filename associated with the binary allele information (in variant major format only)

bim_file = None
filename for marker info in PLINK .bim format

`extract_genotypes(bytes)`
Extracts encoded genotype data from binary formatted file.

Parameters bytes -- array of bytes pulled from the .bed file

Returns standard python list containing the genotype data

Only ind_count genotypes will be returned (even if there are a handful of extra pairs present).

fam_file = None
Filename associated with the pedigree data (first 6 columns from standard pedigree: fid, iid, fid, mid, sex, pheno)

families = None
Pedigree information for reporting

`filter_missing()`
Filter out individuals and SNPs that have too many missing to be considered

Returns None

This must be run prior to actually parsing the genotypes because it initializes the following instance members:

- ind_mask
- total_locus_count
- locus_count
- data_parser.boundary (adds loci with too much missingness)

geno_conversions = None
Genotype conversion

genotype_file = None
Actual pedigree file being parsed (file object)

ind_count = None
Number of valid individuals

ind_mask = None
Mask indicating valid samples

init_genotype_file()
Resets the bed file and preps it for starting at the start of the genotype data
Returns to beginning of file and reads the version so that it points to first marker's info

Returns None

load_bim(*map3=False*)
Basic marker details loading.
(chr, rsid, gen. dist, pos, allele1, allele2)

Parameters *map3* -- When true, ignore the genetic distance column

Returns None

load_fam(*pheno_covar*)
Load contents from the .fam file, updating the *pheno_covar* with family ids found.

Parameters *pheno_covar* -- Phenotype/covariate object

Returns None

load_genotypes()
Prepares the file for genotype parsing.

Returns None

markers = None
Valid loci to be used for analysis

populate_iteration(*iteration*)
Parse genotypes from the file and iteration with relevant marker details.

Parameters *iteration* -- ParseLocus object which is returned per iteration

Returns True indicates current locus is valid.

StopIteration is thrown if the marker reaches the end of the file or the valid genomic region for analysis.

1.2.2 pygwas.boundary module

class pygwas.boundary.BoundaryCheck(*bp=(None, None), kb=(None, None), mb=(None, None)*)
Bases: object

Record boundary specifications from user to control traversal.

Default boundaries are specified in numerical positions along a single chromosome. Users are permitted to provide boundaries in 3 forms: Bases, Kilobases and Megabases. All are recorded as single base offsets from the beginning of the chromosome (starting at 1).

The valid setting doesn't mean the boundary object is invalid, only that no actual boundary ranges have been provided. This is done to allow the user interface code to be a little simpler (i.e. if the user didn't provide bounds using numerical boundaries, it can try instantiating a `SnBoundary` and pass the relevant arguments to that object. If none are valid, then either can be used, at which point both act as chromosome boundaries or simple SNP filters)

If `chrom` is specified, all SNPs and boundaries are expected to reside on that chromosome.

`LoadExclusions(snps)`

Load locus exclusions.

Parameters `snps` -- Can either be a list of rsids or a file containing rsids.

Returns `None`

If `snps` is a file, the file must only contain RSIDs separated by whitespace (tabs, spaces and return characters).

`LoadSNPs(snps=//)`

Define the SNP inclusions (by RSID). This overrides true boundary definition.

Parameters `snps` -- array of RSIDs

Returns `None`

This doesn't define RSID ranges, so it throws `InvalidBoundarySpec` if it encounters what appears to be a range (SNP contains a "-")

`NoExclusions()`

Determine that there are no exclusion criterion in play

Returns `True` if there is no real boundary specification of any kind.

Simple method allowing parsers to short circuit the determination of missingness, which can be moderately compute intensive.

`ReportConfiguration(f)`

Report the boundary configuration details

Parameters `f` -- File (or standard out/err)

Returns `None`

`TestBoundary(chr, pos, rsid)`

Test if locus is within the boundaries and not to be ignored.

Parameters

- `chr` -- Chromosome of locus
- `pos` -- BP position of locus
- `rsid` -- RSID (used to check for exclusions)

Returns `True` if locus isn't to be ignored

`beyond_upper_bound = None`

Is set once the upper limit has been exceeded

`bounds = None`

Actual boundary details in BP

`chrom = -1`

`dropped_snps = None`

Indices of loci that are to be dropped {`chr=>`[`pos1`, `pos2`, ..., `posN`]}

ignored_rs = None
List of RS Numbers to be ignored

target_rs = None
List of RS Numbers to be targeted (ignores all but those listed)

valid = None
True if boundary conditions remain true

1.2.3 pygwas.data_parser module

```
class pygwas.data_parser.DataParser
    Bases: object

    Abstract representation of all dataset parsers

    boundary = <pygwas.boundary.BoundaryCheck object>
        Boundary object specifying valid region for analysis

    compressed_pedigree = False
        When true, assume that standard pedigree and transposed pedigree are compressed with gzip

    get_effa_freq(genotypes)

    get_loci()

    has_fid = True
        When false, pedigree header expects no family id column

    has_liability = False
        When false, pedigree header expects no liability column

    has_parents = True
        When false, pedigree header expects no parents columns

    has_pheno = True
        When false, pedigree header expects no phenotype column

    has_sex = True
        When false, pedigree header expects no sex column

    ind_exclusions = []
        Filter out specific individuals by individual ID

    ind_inclusions = []
        Filter in specific individuals by individual ID

    ind_miss_tol = 1.0
        Filter individuals with too many missing

    max_maf = 1.0
        filter out if a minor allele frequency exceeds this value

    min_maf = 0.0
        this can be used to filter out loci with too few minor alleles

    missing_representation = '0'
        External representation of missingness

    missing_storage = -1

    snp_miss_tol = 1.0
        Filter SNPs with too many missing
```

```
static valid_indid(indid)
```

```
pygwas.data_parser.check_inclusions(item, included=[], excluded=[])
```

Everything passes if both are empty, otherwise, we have to check if empty or is present.

1.2.4 pygwas.exceptions module

```
exception pygwas.exceptions.InvalidBoundarySpec(malformed_boundary)
```

Bases: [*pygwas.exceptions.ReportableException*](#)

Indicate boundary specification was malformed or non-sensical

```
exception pygwas.exceptions.InvalidSelection(msg)
```

Bases: [*pygwas.exceptions.MalformedInputFile*](#)

Indicate that the user provided input that is meaningless.

This is likely a situation where the user provided an invalid name for a phenotype or covariate. Probably a misspelling.

```
exception pygwas.exceptions.InvariantVar(msg='')
```

Bases: [*pygwas.exceptions.ReportableException*](#)

No minor allele found

```
exception pygwas.exceptions.MalformedInputFile(msg)
```

Bases: [*pygwas.exceptions.ReportableException*](#)

Error encountered in data from an input file

```
exception pygwas.exceptions.NanInResult(msg='')
```

Bases: [*pygwas.exceptions.ReportableException*](#)

NaN found in result

```
exception pygwas.exceptions.NoMatchedPhenoCovars(msg='')
```

Bases: [*pygwas.exceptions.ReportableException*](#)

No ids matched between pheno or covar and the family data

```
exception pygwas.exceptions.ReportableException(msg)
```

Bases: [*exceptions.Exception*](#)

Simple exception with message

```
exception pygwas.exceptions.TooFewAlleles(chr=None, rsid=None, pos=None, alleles=None, index=None)
```

Bases: [*pygwas.exceptions.TooManyAlleles*](#)

Indicate fixed allele was found

```
exception pygwas.exceptions.TooManyAlleles(chr=None, rsid=None, pos=None, alleles=None, index=None, prefix='Too many alleles: ')
```

Bases: [*pygwas.exceptions.ReportableException*](#)

Indicate locus found with more than 2 alleles

`alleles = None`

Allele 1 and 2

`chr = None`

Chromosome

```

index = None
    Index of the locus within the file

pos = None
    BP Position

rsid = None
    RSID

exception pygwas.exceptions.UnsolvedLocus(msg)
    Bases: pygwas.exceptions.ReportableException

```

1.2.5 pygwas.impute_parser module

```

class pygwas.impute_parser.Encoding
    Bases: object

    Simple enumeration for various model encodings

    Additive = 0
    Dominant = 1
    Genotype = 3
    Raw = 4
    Recessive = 2

class pygwas.impute_parser.Parser(fam_details, archive_list, chroms, info_files=[])
    Bases: pygwas.data_parser.DataParser

    Parse IMPUTE style output.

    ReportConfiguration(file)

        Parameters file -- Destination for report details

        Returns None

archives = None
    This is only the list of files to be processed

chroms = None
    List of chroms to match files listed in archives

current_chrom = None
    This will be used to record the chromosome of the current file

current_file = None
    This will be used to record the opened file used for parsing

current_info = None
    This will be used to record the info file associated with quality of SNPs

fam_details = None
    single file containing the subject details (similar to plink's .fam)

gen_ext = 'gen.gz'
    The genotype file suffix (of not following convention)

get_effa_freq(genotypes)
    Returns the effect allele's frequency

```

`get_next_line()`

If we reach the end of the file, we simply open the next, until we run out of archives to process

`info_ext = 'info'`

the extension associated with the .info files if not using conventions

`info_files = None`

array of .info files

`info_threshold = 0.4`

The threshold associated with the .info info column

`load_family_details(pheno_covar)`

Load family data updating the pheno_covar with family ids found.

Parameters `pheno_covar` -- Phenotype/covariate object

Returns None

`load_genotypes()`

Prepares the files for genotype parsing.

Returns None

`populate_iteration(iteration)`

Parse genotypes from the file and iteration with relevant marker details.

Parameters `iteration` -- ParseLocus object which is returned per iteration

Returns True indicates current locus is valid.

StopIteration is thrown if the marker reaches the end of the file or the valid genomic region for analysis.

`pygwas.impute_parser.SetEncoding(sval)`

Sets the encoding variable according to the text passed

Parameters `sval` -- text specification for the desired model

1.2.6 pygwas.locus module

`class pygwas.locus.Locus(other=None)`

Bases: object

`alleles = None`

List of alleles present

`chr = None`

Chromosome

`exp_hetero_freq`

Returns the estimated frequency of heterozygotes

`flip()`

This will switch major/minor around, regardless of frequency truth.

This is intended for forcing one of two populations to relate correctly to the same genotype definitions. When flipped, Ps and Qs will be backward, and the maf will no longer relate to the “minor” allele frequency. However, it does allow clients to use the same calls for each population without having to perform checks during those calculations.

`hetero_count = None`

total count of heterozygotes observed

hetero_freq
Returns the frequency of observed heterozygotes (not available with all parsers)

maf
Returns the MAF. This is valid for all parsers

maj_allele_count = None
total number of major alleles observed

major_allele
Sets/Returns the encoding for the major allele (A, C, G, T, etc)

min_allele_count = None
total number of minor alleles observed

minor_allele
Sets/Returns the encoding for minor allele

missing_allele_count = None
total number of missing alleles were observed

p
Frequency for first allele

pos = None
BP Position

q
Frequency for second allele

rsid = None
RSID

sample_size
Returns to total sample size

total_allele_count
Returns the total number of alleles

1.2.7 pygwas.mach_parser module

class pygwas.mach_parser.Encoding
Bases: object

Dosage = 0
Currently there is only one way to interpret these values

class pygwas.mach_parser.Parser(*archive_list*, *info_files*=[])
Bases: *pygwas.data_parser.DataParser*

Parse IMPUTE style output.

Due to the nature of the mach data format, we must load the data first into member before we can begin analyzing it. Due to the massive amount of data, SNPs are loaded in in chunks.

ISSUES:

- Currently, we will not be filtering on individuals except by explicit removal
- We are assuming that each gzip archive contains all data associated with the loci contained within (i.e. there won't be separate files with different subjects inside) ((Todd email jan-9-2015))

- There is no place to store RSID from the output that I've seen (Minimac output generated by Ben Zhang). As of Feb 2016, I've made the chr:pos ID requirement optional, and added in a 3rd column (optional for even the option) which can be rsid if present (currently, the data we have puts alleles in that column, but future imputations can be done differently). When using the mach-chrpos flag, users can produce results that appear similar to plink output with chromosome, position and optionally RSIDs in the expected columns. This is, by default, turned off, and the entire contents of that ID column is stored as simply the RSID when reporting results.

`ReportConfiguration(file)`

Report the configuration details for logging purposes.

Parameters `file` -- Destination for report details

Returns None

`chrpos_encoding = False`

`chunk_stride = 50000`

Number of loci to parse at a time (larger stride requires more memory)

`dosage_ext = 'dose.gz'`

Extension for the dosage file

`get_effa_freq(genotypes)`

Returns the frequency of the effect allele

`info_ext = 'info.gz'`

Extension for the info file

`load_family_details(pheno_covar)`

Load contents from the .fam file, updating the pheno_covar with family ids found.

Parameters `pheno_covar` -- Phenotype/covariate object

Returns None

`load_genotypes()`

Actually loads the first chunk of genotype data into memory due to the individual oriented format of MACH data.

Due to the fragmented approach to data loading necessary to avoid running out of RAM, this function will initialize the data structures with the first chunk of loci and prepare it for otherwise normal iteration.

Also, because the parser can be assigned more than one .gen file to read from, it will automatically move to the next file when the first is exhausted.

`min_rsquared = 0.3`

rsquared threshold for analysis (obtained from the mach output itself)

`openfile(filename)`

`parse_genotypes(lb, ub)`

Extracts a fraction of the file (current chunk of loci) loading the genotypes into memory.

Parameters

- `lb` -- Lower bound of the current chunk
- `ub` -- Upper bound of the current chunk

Returns Dosage dosages for current chunk

`populate_iteration(iteration)`

Parse genotypes from the file and iteration with relevant marker details.

Parameters `iteration` -- ParseLocus object which is returned per iteration

Returns True indicates current locus is valid.

StopIteration is thrown if the marker reaches the end of the file or the valid genomic region for analysis.

This function will force a load of the next chunk when necessary.

1.2.8 pygwas.parsed_locus module

`class pygwas.parsed_locus.ParsedLocus(datasource, index=-1)`

Bases: `pygwas.locus.Locus`

Locus data representing current iteration from a dataset

Provide an iterator interface for all dataset types.

`cur_idx = None`

Index within the list of loci being analyzed

`genotype_data = None`

Actual genotype data for this locus

`next()`

Move to the next valid locus.

Will only return valid loci or exit via StopIteration exception

1.2.9 pygwas.pedigree_parser module

`class pygwas.pedigree_parser.Parser(mapfile, datasource)`

Bases: `pygwas.data_parser.DataParser`

Parse standard pedigree dataset.

Data should follow standard format for pedigree data, except alleles be either numerical (1 and 2) or as bases (A, C, T and G). All loci must have 2 alleles to be returned.

Attributes initialized to None are only available after `load_genotypes()` has been called.

Issues:

- Pedigree files are currently loaded in their entirety, but we could load them in according to chunks like we are doing in mach input.
- There are a bunch of legacy lists which should be reduced to a single list of Locus objects.

`ReportConfiguration(file)`

Report configuration for logging purposes.

Parameters `file` -- Destination for report details

Returns None

`alleles = None`

List of both alleles for each valid locus

`datasource = None`

Filename for the actual pedigree information

genotypes = **None**
Matrix of genotype data

get_loci()

individual_mask = **None**
Mask used to remove excluded and filtered calls from the genotype data (each position represents an individual)

invalid_loci = **None**
Loci that are being ignored due to filtration

load_genotypes(*pheno_covar*)
Load all data into memory and propagate valid individuals to *pheno_covar*.

Parameters *pheno_covar* -- Phenotype/covariate object is updated with subject information :return: **None**

load_mapfile(*map3=False*)
Load the marker data

Parameters *map3* -- When true, ignore the gen. distance column

Builds up the marker list according to the boundary configuration

locus_count = **None**
Number of valid loci

mapfile = **None**
Filename for the marker information

markers = **None**
List of valid Locus Objects

markers_maf = **None**
List of MAF at each locus

populate_iteration(*iteration*)
Parse genotypes from the file and iteration with relevant marker details.

Parameters *iteration* -- ParseLocus object which is returned per iteration

Returns True indicates current locus is valid.

StopIteration is thrown if the marker reaches the end of the file or the valid genomic region for analysis.

rsids = **None**
List of all SNP names for valid loci

1.2.10 pygwas.pheno_covar module

class `pygwas.pheno_covar.PhenoCovar`
Bases: `object`

Store both phenotype and covariate data in a single object.

Provide iterable interface to allow evaluation of multiple phenotypes easily. Covariates do not change during iteration. Missing is updated according to the missing content within the phenotype (and covariates as well).

`add_subject(ind_id, sex=None, phenotype=None)`
 Add new subject to study, with optional sex and phenotype
 Throws `MalformedInputFile` if sex is can't be converted to int

`covariate_data = None`
 All covariate data `[[cov1],[cov2],etc]`

`covariate_labels = None`
 List of covariate names from header, if provided SEX is implied, if `sex_as_covariate` is true.
 Covariates loaded without header are simply named Cov-N

`destandardize_variables(tv, blin, bvar, errBeta, nonmissing)`
 Destandardize betas and other components.

`do_standardize_variables = None`
 Allows you to turn off standardization

`freeze_subjects()`
 Converts variable data into numpy arrays.
 This is required after all subjects have been added via the `add_subject` function, since we don't know ahead of time who is participating in the analysis due to various filtering possibilities.

`individual_mask = None`
 True indicates an individual is to be excluded

`load_covarfile(file, indices=[], names=[], sample_file=False)`
 Load covariate data from file.
 Unlike phenofiles, if we already have data, we keep it (that would be the sex covariate)

`load_phenofile(file, indices=[], names=[], sample_file=False)`
 Load phenotype data from phenotype file
 Whitespace delimited, FAMID, INDID, VAR1, [VAR2], etc
 Users can specify phenotypes of interest via indices and names. Indices are 1 based and start with the first variable. names must match name specified in the header (case is ignored).

`missing_encoding = -9`
 Internal encoding for missingness

`pedigree_data = None`
 Pedigree information {FAMID:INDID => index, etc}

`phenotype_data = None`
 Raw phenotype data with every possible phenotype `[[ph1],[ph2],etc]`

`phenotype_names = None`
 List of phenotype names from header, if provided. If no header is found, the phenotype is simply named Pheno-N

`prep_testvars()`
 Make sure that the data is in the right form and standardized as expected.

`sex_as_covariate = False`
 Do we use sex as a covariate?

`test_variables = None`
 finalized data ready for analysis

1.2.11 pygwas.snp_boundary_check module

`class pygwas.snp_boundary_check.SnpBoundaryCheck(snps=//)`

Bases: *pygwas.boundary.BoundaryCheck*

RS (or other name) based boundary checking.

Same rules apply as those for `BoundaryCheck`, except users can provide multiple RS boundary regions. Though, all boundary groups must reside on a single chromosome.

Class members (these are not intended for public consumption):

- `start_bounds` bp location for boundary starts Currently, only one boundary is permitted. This is to remain consistent with plink
- `end_bounds` bp location for boundary end (inclusive)
- `ignored_rs` List of RS numbers to be ignored
- `target_rs` List of RS numbers to be targeted
- **`dropped_snps` indices of loci that are to be dropped** {chr=>[pos1, pos2, ...]}
- **`end_rs` This is used during iteration to identify when to turn “off” the current boundary group**

`NoExclusions()`

Determine that there are no exclusion criterion in play

Returns True if there is no real boundary specification of any kind.

Simple method allowing parsers to short circuit the determination of missingness, which can be moderately compute intensive.

`ReportConfiguration(f)`

Report the boundary configuration details

Parameters `f` -- File (or standard out/err)

Returns None

`TestBoundary(chr, pos, rsid)`

Test if locus is within the boundaries and not to be ignored.

Parameters

- `chr` -- Chromosome of locus
- `pos` -- BP position of locus
- `rsid` -- RSID (used to check for exclusions)

Returns True if locus isn't to be ignored

1.2.12 pygwas.standardizer module

`class pygwas.standardizer.NoStandardization(pc)`

Bases: *pygwas.standardizer.StandardizedVariable*

This is mostly a placeholder for standardizers. Each application will probably have a specific approach to standardizing/destandardizing the input/output.

`destandardize(estimated, se, **kwargs)`

When the pheno/covar data has been standardized, this can be used to rescale the betas back to a meaningful value using the original data.

For the “Un-standardized” data, we do no conversion.

`standardize()`

Standardize the variables within a range [-1.0 and 1.0]

This replaces the local copies of this data. When it’s time to scale back, use `destandardize` from the `datasource` for that.

`class pygwas.standardizer.StandardizedVariable(pc)`

Bases: `object`

Optional plugin object that can be used to standardize covariate and phenotype data.

Many algorithms require that input be standardized in some way in order to work properly, however, rescaling the results is algorithm specific. In order to facilitate this situation, application authors can write up application specific Standardization objects for use with the data parsers.

`covar_count = None`

number of covars

`covariates = None`

Standardized covariate data

`datasource = None`

Reference back to the `pheno_covar` object for access to raw data

`destandardize()`

Stub for the appropriate `destandardizer` function.

Each object type will do it’s own thing here.

`get_covariate_name(idx)`

Return label for a specific covariate

Parameters `idx` -- which covariate?

Returns string label

`get_covariate_names()`

Return all covariate labels as a list

Returns list of covariate names

`get_phenotype_name()`

Returns current phenotype name

`get_variables(missing_in_geno=None)`

Extract the complete set of data based on missingness over all for the current locus.

Parameters `missing_in_geno` -- mask associated with missingness in genotype

Returns (phenotypes, covariates, nonmissing used for this set of vars)

`idx = None`

index of the current phenotype

`missing = None`

mask representing missingness (1 indicates missing)

`pheno_count = None`

number of phenotypes

```
phenotypes = None
    standardized phenotype data

standardize()
    Stub for the appropriate standardizer function

    Each Standardizer object will do it's own thing here.

pygwas.standardizer.get_standardizer()

pygwas.standardizer.set_standardizer(std)
```

1.2.13 pygwas.transposed_pedigree_parser module

```
class pygwas.transposed_pedigree_parser.Parser(tfam, tped)
    Bases: pygwas.data_parser.DataParser

    Parse transposed pedigree dataset

    Class Members: tfam_file filename associated with the pedigree information tped_file Filename
    associated with the genotype data families Pedigree information for reporting genotype_file Actual
    pedigree file begin parsed (file object)

    ReportConfiguration(file)

    filter_missing()
        Filter out individuals and SNPs that have too many missing to be considered

    load_genotypes()
        This really just intializes the file by opening it up.

    load_tfam(pheno_covar)
        Load the pedigree portion of the data and sort out exclusions

    populate_iteration(iteration)
        Pour the current data into the iteration object

    process_genotypes(data)
        Parse pedigree line and remove excluded individuals from geno

        Translates alleles into numerical genotypes (0, 1, 2) counting number of minor alleles.

        Throws exceptions if an there are not 2 distinct alleles
```

1.2.14 Module contents

```
pygwas.BuildReportLine(key, value)
    Prepare key/value for reporting in configuration report

    Parameters

        • key -- configuration 'keyword'

        • value -- value reported to be associated with keyword

    Returns formatted line starting with a comment

pygwas.Exit(msg, code=1)
    Exit execution with return code and message :param msg: Message displayed prior to exit :param code:
    code returned upon exiting
```

`pygwas.ExitIf(msg, do_exit, code=1)`
Exit if `do_exit` is true

Parameters

- `msg` -- Message displayed prior to exit
- `do_exit` -- exit when true
- `code` -- application's return code upon exit

`pygwas.sys_call(cmd)`
Execute `cmd` and capture `stdout` and `stderr`

Parameters `cmd` -- command to be executed

Returns (`stdout`, `stderr`)

m

meanvar, 4
meanvar.mv_esteq, 1
meanvar.mvresult, 2
meanvar.mvstandardizer, 3
meanvar.simple_timer, 3

p

pygwas, 18
pygwas.bed_parser, 4
pygwas.boundary, 5
pygwas.data_parser, 7
pygwas.exceptions, 8
pygwas.impute_parser, 9
pygwas.locus, 10
pygwas.mach_parser, 11
pygwas.parsed_locus, 13
pygwas.pedigree_parser, 13
pygwas.pheno_covar, 14
pygwas.snp_boundary_check, 16
pygwas.standardizer, 16
pygwas.transposed_pedigree_parser, 18

A

add_subject() (pygwas.pheno_covar.PhenoCovar method), 14
 Additive (pygwas.impute_parser.Encoding attribute), 9
 alleles (pygwas.bed_parser.Parser attribute), 4
 alleles (pygwas.exceptions.TooManyAlleles attribute), 8
 alleles (pygwas.locus.Locus attribute), 10
 alleles (pygwas.pedigree_parser.Parser attribute), 13
 archives (pygwas.impute_parser.Parser attribute), 9

B

bed_file (pygwas.bed_parser.Parser attribute), 4
 beta_pvalues (meanvar.mvresult.MVResult attribute), 2
 beta_stderr (meanvar.mvresult.MVResult attribute), 2
 betas (meanvar.mvresult.MVResult attribute), 2
 beyond_upper_bound (pygwas.boundary.BoundaryCheck attribute), 6
 bim_file (pygwas.bed_parser.Parser attribute), 4
 boundary (pygwas.data_parser.DataParser attribute), 7
 BoundaryCheck (class in pygwas.boundary), 5
 bounds (pygwas.boundary.BoundaryCheck attribute), 6
 BuildReportLine() (in module pygwas), 18

C

check_inclusions() (in module pygwas.data_parser), 8
 chr (meanvar.mvresult.MVResult attribute), 2
 chr (pygwas.exceptions.TooManyAlleles attribute), 8
 chr (pygwas.locus.Locus attribute), 10
 chrom (pygwas.boundary.BoundaryCheck attribute), 6
 chroms (pygwas.impute_parser.Parser attribute), 9
 chrpos_encoding (pygwas.mach_parser.Parser attribute), 12

chunk_stride (pygwas.mach_parser.Parser attribute), 12
 compressed_pedigree (pygwas.data_parser.DataParser attribute), 7
 covar_count (pygwas.standardizer.StandardizedVariable attribute), 17
 covar_labels (meanvar.mvresult.MVResult attribute), 2
 covariate_data (pygwas.pheno_covar.PhenoCovar attribute), 15
 covariate_labels (pygwas.pheno_covar.PhenoCovar attribute), 15
 covariates (pygwas.standardizer.StandardizedVariable attribute), 17
 cur_idx (pygwas.parsed_locus.ParsedLocus attribute), 13
 current_chrom (pygwas.impute_parser.Parser attribute), 9
 current_file (pygwas.impute_parser.Parser attribute), 9
 current_info (pygwas.impute_parser.Parser attribute), 9

D

DataParser (class in pygwas.data_parser), 7
 datasource (pygwas.pedigree_parser.Parser attribute), 13
 datasource (pygwas.standardizer.StandardizedVariable attribute), 17
 destandardize() (meanvar.mvstandardizer.Standardizer method), 3
 destandardize() (pygwas.standardizer.NoStandardization method), 16
 destandardize() (pygwas.standardizer.StandardizedVariable method), 17
 destandardize_variables() (pygwas.pheno_covar.PhenoCovar method), 15

do_standardize_variables (pygwas.pheno_covar.PhenoCovar attribute), 15

Dominant (pygwas.impute_parser.Encoding attribute), 9

Dosage (pygwas.mach_parser.Encoding attribute), 11

dosage_ext (pygwas.mach_parser.Parser attribute), 12

dropped_snps (pygwas.boundary.BoundaryCheck attribute), 6

E

eff_alcount (meanvar.mvresult.MVResult attribute), 2

Encoding (class in pygwas.impute_parser), 9

Encoding (class in pygwas.mach_parser), 11

Exit() (in module pygwas), 18

ExitIf() (in module pygwas), 18

exp_hetero_freq (pygwas.locus.Locus attribute), 10

extract_genotypes() (pygwas.bed_parser.Parser method), 4

F

fam_details (pygwas.impute_parser.Parser attribute), 9

fam_file (pygwas.bed_parser.Parser attribute), 4

families (pygwas.bed_parser.Parser attribute), 4

filter_missing() (pygwas.bed_parser.Parser method), 4

filter_missing() (pygwas.transposed_pedigree_parser.Parser method), 18

flip() (pygwas.locus.Locus method), 10

freeze_subjects() (pygwas.pheno_covar.PhenoCovar method), 15

G

gen_ext (pygwas.impute_parser.Parser attribute), 9

geno_conversions (pygwas.bed_parser.Parser attribute), 4

Genotype (pygwas.impute_parser.Encoding attribute), 9

genotype_data (pygwas.parsed_locus.ParsedLocus attribute), 13

genotype_file (pygwas.bed_parser.Parser attribute), 5

genotypes (pygwas.pedigree_parser.Parser attribute), 14

get_covariate_name() (pygwas.standardizer.StandardizedVariable method), 17

get_covariate_names() (pygwas.standardizer.StandardizedVariable method), 17

get_effa_freq() (pygwas.data_parser.DataParser method), 7

get_effa_freq() (pygwas.impute_parser.Parser method), 9

get_effa_freq() (pygwas.mach_parser.Parser method), 12

get_loci() (pygwas.data_parser.DataParser method), 7

get_loci() (pygwas.pedigree_parser.Parser method), 14

get_next_line() (pygwas.impute_parser.Parser method), 9

get_phenotype_name() (pygwas.standardizer.StandardizedVariable method), 17

get_standardizer() (in module pygwas.standardizer), 18

get_variables() (pygwas.standardizer.StandardizedVariable method), 17

H

has_fid (pygwas.data_parser.DataParser attribute), 7

has_liability (pygwas.data_parser.DataParser attribute), 7

has_parents (pygwas.data_parser.DataParser attribute), 7

has_pheno (pygwas.data_parser.DataParser attribute), 7

has_sex (pygwas.data_parser.DataParser attribute), 7

hetero_count (pygwas.locus.Locus attribute), 10

hetero_freq (pygwas.locus.Locus attribute), 10

I

idx (pygwas.standardizer.StandardizedVariable attribute), 17

ignored_rs (pygwas.boundary.BoundaryCheck attribute), 6

ind_count (pygwas.bed_parser.Parser attribute), 5

ind_exclusions (pygwas.data_parser.DataParser attribute), 7

ind_inclusions (pygwas.data_parser.DataParser attribute), 7

ind_mask (pygwas.bed_parser.Parser attribute), 5

ind_miss_tol (pygwas.data_parser.DataParser attribute), 7

index (pygwas.exceptions.TooManyAlleles attribute), 8

individual_mask (pygwas.pedigree_parser.Parser attribute), 14

individual_mask (pygwas.pheno_covar.PhenoCovar attribute), 15

- [info_ext \(pygwas.impute_parser.Parser attribute\), 10](#)
[info_ext \(pygwas.mach_parser.Parser attribute\), 12](#)
[info_files \(pygwas.impute_parser.Parser attribute\), 10](#)
[info_threshold \(pygwas.impute_parser.Parser attribute\), 10](#)
[init_genotype_file\(\) \(pygwas.bed_parser.Parser method\), 5](#)
[invalid_loci \(pygwas.pedigree_parser.Parser attribute\), 14](#)
[InvalidBoundarySpec, 8](#)
[InvalidSelection, 8](#)
[InvariantVar, 8](#)
- ## L
- [lmpv \(meanvar.mvresult.MVResult attribute\), 2](#)
[load_bim\(\) \(pygwas.bed_parser.Parser method\), 5](#)
[load_covarfile\(\) \(pygwas.pheno_covar.PhenoSovar method\), 15](#)
[load_fam\(\) \(pygwas.bed_parser.Parser method\), 5](#)
[load_family_details\(\) \(pygwas.impute_parser.Parser method\), 10](#)
[load_family_details\(\) \(pygwas.mach_parser.Parser method\), 12](#)
[load_genotypes\(\) \(pygwas.bed_parser.Parser method\), 5](#)
[load_genotypes\(\) \(pygwas.impute_parser.Parser method\), 10](#)
[load_genotypes\(\) \(pygwas.mach_parser.Parser method\), 12](#)
[load_genotypes\(\) \(pygwas.pedigree_parser.Parser method\), 14](#)
[load_genotypes\(\) \(pygwas.transposed_pedigree_parser.Parser method\), 18](#)
[load_mapfile\(\) \(pygwas.pedigree_parser.Parser method\), 14](#)
[load_phenofile\(\) \(pygwas.pheno_covar.PhenoSovar method\), 15](#)
[load_tfam\(\) \(pygwas.transposed_pedigree_parser.Parser method\), 18](#)
[LoadExclusions\(\) \(pygwas.boundary.BoundaryCheck method\), 6](#)
[LoadSNPs\(\) \(pygwas.boundary.BoundaryCheck method\), 6](#)
[Locus \(class in pygwas.locus\), 10](#)
[locus_count \(pygwas.pedigree_parser.Parser attribute\), 14](#)
- ## M
- [maf \(meanvar.mvresult.MVResult attribute\), 2](#)
[maf \(pygwas.locus.Locus attribute\), 11](#)
[maj_allele \(meanvar.mvresult.MVResult attribute\), 2](#)
[maj_allele_count \(pygwas.locus.Locus attribute\), 11](#)
[major_allele \(pygwas.locus.Locus attribute\), 11](#)
[MalformedInputFile, 8](#)
[mapfile \(pygwas.pedigree_parser.Parser attribute\), 14](#)
[markers \(pygwas.bed_parser.Parser attribute\), 5](#)
[markers \(pygwas.pedigree_parser.Parser attribute\), 14](#)
[markers_maf \(pygwas.pedigree_parser.Parser attribute\), 14](#)
[max_maf \(pygwas.data_parser.DataParser attribute\), 7](#)
[meanvar \(module\), 4](#)
[meanvar.mv_esteq \(module\), 1](#)
[meanvar.mvresult \(module\), 2](#)
[meanvar.mvstandardizer \(module\), 3](#)
[meanvar.simple_timer \(module\), 3](#)
[MeanVarEstEQ\(\) \(in module meanvar.mv_esteq\), 1](#)
[min_allele \(meanvar.mvresult.MVResult attribute\), 2](#)
[min_allele_count \(pygwas.locus.Locus attribute\), 11](#)
[min_maf \(pygwas.data_parser.DataParser attribute\), 7](#)
[min_rsquared \(pygwas.mach_parser.Parser attribute\), 12](#)
[minor_allele \(pygwas.locus.Locus attribute\), 11](#)
[missing \(pygwas.standardizer.StandardizedVariable attribute\), 17](#)
[missing_allele_count \(pygwas.locus.Locus attribute\), 11](#)
[missing_encoding \(pygwas.pheno_covar.PhenoSovar attribute\), 15](#)
[missing_representation \(pygwas.data_parser.DataParser attribute\), 7](#)
[missing_storage \(pygwas.data_parser.DataParser attribute\), 7](#)
[MVResult \(class in meanvar.mvresult\), 2](#)
- ## N
- [NaNInResult, 8](#)
[next\(\) \(pygwas.parsed_locus.ParsedLocus method\), 13](#)
[NoExclusions\(\) \(pygwas.boundary.BoundaryCheck method\), 6](#)
[NoExclusions\(\) \(pygwas.snp_boundary_check.SnpBoundaryCheck method\), 16](#)
[NoMatchedPhenoCovars, 8](#)
[non_miss \(meanvar.mvresult.MVResult attribute\), 2](#)
[NoStandardization \(class in pygwas.standardizer\), 16](#)
- ## O
- [openfile\(\) \(pygwas.mach_parser.Parser method\), 12](#)

P

p (pygwas.locus.Locus attribute), 11
p_mvtest (meanvar.mvresult.MVResult attribute), 2
p_variance (meanvar.mvresult.MVResult attribute), 2
parse_genotypes() (pygwas.mach_parser.Parser method), 12
ParsedLocus (class in pygwas.parsed_locus), 13
Parser (class in pygwas.bed_parser), 4
Parser (class in pygwas.impute_parser), 9
Parser (class in pygwas.mach_parser), 11
Parser (class in pygwas.pedigree_parser), 13
Parser (class in pygwas.transposed_pedigree_parser), 18
pedigree_data (pygwas.pheno_covar.PhenoSovar attribute), 15
ph_label (meanvar.mvresult.MVResult attribute), 2
pheno_count (pygwas.standardizer.StandardizedVariable attribute), 17
PhenoCovar (class in pygwas.pheno_covar), 14
phenotype_data (pygwas.pheno_covar.PhenoSovar attribute), 15
phenotype_names (pygwas.pheno_covar.PhenoSovar attribute), 15
phenotypes (pygwas.standardizer.StandardizedVariable attribute), 17
populate_iteration() (pygwas.bed_parser.Parser method), 5
populate_iteration() (pygwas.impute_parser.Parser method), 10
populate_iteration() (pygwas.mach_parser.Parser method), 12
populate_iteration() (pygwas.pedigree_parser.Parser method), 14
populate_iteration() (pygwas.transposed_pedigree_parser.Parser method), 18
pos (meanvar.mvresult.MVResult attribute), 2
pos (pygwas.exceptions.TooManyAlleles attribute), 9
pos (pygwas.locus.Locus attribute), 11
prep_testvars() (pygwas.pheno_covar.PhenoSovar method), 15
print_header() (meanvar.mvresult.MVResult method), 2
print_result() (meanvar.mvresult.MVResult method), 3
process_genotypes() (pygwas.transposed_pedigree_parser.Parser method), 18
pygwas (module), 18
pygwas.bed_parser (module), 4
pygwas.boundary (module), 5
pygwas.data_parser (module), 7
pygwas.exceptions (module), 8

pygwas.impute_parser (module), 9
pygwas.locus (module), 10
pygwas.mach_parser (module), 11
pygwas.parsed_locus (module), 13
pygwas.pedigree_parser (module), 13
pygwas.pheno_covar (module), 14
pygwas.snp_boundary_check (module), 16
pygwas.standardizer (module), 16
pygwas.transposed_pedigree_parser (module), 18

Q

q (pygwas.locus.Locus attribute), 11

R

Raw (pygwas.impute_parser.Encoding attribute), 9
Recessive (pygwas.impute_parser.Encoding attribute), 9
report() (meanvar.simple_timer.SimpleTimer method), 3
ReportableException, 8
ReportConfiguration() (pygwas.bed_parser.Parser method), 4
ReportConfiguration() (pygwas.boundary.BoundaryCheck method), 6
ReportConfiguration() (pygwas.impute_parser.Parser method), 9
ReportConfiguration() (pygwas.mach_parser.Parser method), 12
ReportConfiguration() (pygwas.pedigree_parser.Parser method), 13
ReportConfiguration() (pygwas.snp_boundary_check.SnpBoundaryCheck method), 16
ReportConfiguration() (pygwas.transposed_pedigree_parser.Parser method), 18
reset() (meanvar.simple_timer.SimpleTimer method), 3
result() (meanvar.simple_timer.SimpleTimer method), 3
rsid (meanvar.mvresult.MVResult attribute), 3
rsid (pygwas.exceptions.TooManyAlleles attribute), 9
rsid (pygwas.locus.Locus attribute), 11
rsids (pygwas.pedigree_parser.Parser attribute), 14
RunAnalysis() (in module meanvar.mv_esteq), 1
RunMeanVar() (in module meanvar.mv_esteq), 1
runtime (meanvar.mvresult.MVResult attribute), 3
runtime() (meanvar.simple_timer.SimpleTimer method), 3

S

sample_size (pygwas.locus.Locus attribute), 11

[set_standardizer\(\)](#) (in module `pygwas.standardizer`), [18](#)
[SetEncoding\(\)](#) (in module `pygwas.impute_parser`), [10](#)
[sex_as_covariate](#) (`pygwas.pheno_covar.PhenoCovar` attribute), [15](#)
[SimpleTimer](#) (class in `meanvar.simple_timer`), [3](#)
[snp_miss_tol](#) (`pygwas.data_parser.DataParser` attribute), [7](#)
[SnpBoundaryCheck](#) (class in `pygwas.snp_boundary_check`), [16](#)
[standardize\(\)](#) (`meanvar.mvstandardizer.Standardizer` method), [3](#)
[standardize\(\)](#) (`pygwas.standardizer.NoStandardization` method), [17](#)
[standardize\(\)](#) (`pygwas.standardizer.StandardizedVariable` method), [18](#)
[StandardizedVariable](#) (class in `pygwas.standardizer`), [17](#)
[Standardizer](#) (class in `meanvar.mvstandardizer`), [3](#)
[stringify\(\)](#) (`meanvar.mvresult.MVResult` method), [3](#)
[sys_call\(\)](#) (in module `pygwas`), [19](#)

T

[target_rs](#) (`pygwas.boundary.BoundaryCheck` attribute), [7](#)
[test_variables](#) (`pygwas.pheno_covar.PhenoCovar` attribute), [15](#)
[TestBoundary\(\)](#) (`pygwas.boundary.BoundaryCheck` method), [6](#)
[TestBoundary\(\)](#) (`pygwas.snp_boundary_check.SnpBoundaryCheck` method), [16](#)
[TooFewAlleles](#), [8](#)
[TooManyAlleles](#), [8](#)
[total_allele_count](#) (`pygwas.locus.Locus` attribute), [11](#)

U

[UnsolvedLocus](#), [9](#)

V

[valid](#) (`pygwas.boundary.BoundaryCheck` attribute), [7](#)
[valid_indid\(\)](#) (`pygwas.data_parser.DataParser` static method), [7](#)