Documentation of PyMSQ / `msq` Module

Authors: Abdulraheem Musa and Norbert Reinsch
Last Updated: 08.04.2025

This module (`msq.py`) is part of the **PyMSQ** package, offering a suite of functions for genetic analysis and selection strategies:

1. **Data Loading** (`load_package_data`)
2. **Expected LD Matrices** (`expldmat`)
3. **Mendelian Sampling (Co)Variance** (`msvarcov`)
4. **Mendelian Sampling Correlations** (`msvarcov_corr`)
5. **Similarity Matrices** (`simmat`)
6. **Selection Strategies** (`selstrat`)

Below is an overview of each function, including parameters, return values, and usage examples.

Importing the Module

You can install and import **PyMSQ** like so:

```
# Example usage:
from PyMSQ import msq

# Now you can call, for example:
data = msq.load_package_data()
```

`1. msq.load_package_data()`

```
def load_package_data():
    ...
```

Description

Loads a bundled example dataset from Musa and Reinsch (2025). Files typically include:

- genetic map,
- marker effects for three milk traits (fat, protein, and pH),
- phased genotypes,
- phenotypic information,
- and a pedigree file.

Returns

dict of pandas.DataFrame:

Keys and their content:

- `chromosome_data` : Chromosome map info

- `marker_effect_data` : Marker effects
- `genotype_data` : Phased genotypes
- `group_data` : Group/sex info
- `pedigree_data` : Pedigree data

Raises

- `FileNotFoundError` : If a required file is missing.

Example

```
from PyMSQ import msq

data = msq.load_package_data()
data = load_package_data()
gmap = data['chromosome_data']
meff = data['marker_effect_data']
gmat = data['genotype_data']
group = data['group_data']
print(gmap.head())
```

2. `msq.expldmat(gmap, group, **kwargs)`

```
def expldmat(gmap, group, **kwargs):
    ...
```

Description

Builds expected within-family LD matrices (R) for each chromosome, based on a provided genetic map (`gmap`).

- If there are multiple columns in `gmap` describing distances/recombination rates (e.g., one column per group), the result may be group-specific.

Parameters

- gmap: `pd.DataFrame` : A genetic map DataFrame that must contain at least:
  - A chromosome identifier column (e.g., integer `chr`),
  - marker name,
  - bp position
  - and distance between markers (cM) or recombination rates
- group: `pd.DataFrame` : Group/sex info. Must include at least [ID, Group] columns.
- kwargs:
  - `mposunit` : `{"cM","reco"}` ; default `"cM"` Unit for marker positions: either centiMorgans ("cM") or direct recombination rates ("reco").
  - `method` : `int` ; default `1` Method for computing LD: `1` = Bonk et al., `2` = Santos et al.
  - `threshold` : `float` or `None` ; default = `None` If provided, sets the maximum distance (cM or reco) beyond which LD is assumed 0.

Returns

- list or dict
  - If only one distance column is used, returns a list of LD matrices (one per chromosome).
  - If multiple group-specific columns exist, returns a dict keyed by group name, each containing a list of chromosome-wise LD matrices.

Raises

- `ValueError`
  - If `mposunit` is not recognized (must be cM or reco).
  - If the first value of recombination rate on any chromosome is not zero (in certain modes).

Example

```
exp_ldmat = msq.expldmat(gmap, group, mposunit="cM", method=2)
print(exp_ldmat[0])  # if single group-dist column, it's a list
```

3. `msq.msvarcov(gmat, gmap, meff, exp_ldmat, group, **kwargs)`

```
def msvarcov(gmat, gmap, meff, exp_ldmat, group, **kwargs):
    ...
```

Description

Computes Mendelian sampling (co)variance for each individual (or subset) using phased genotypes and a genetic map with corresponding LD matrices.

- Supports multi-trait scenarios via an optional set of index weights (indwt).
- Optionally centers the marker data.

Parameters

- gmat: `pd.DataFrame` or `np.ndarray` Phased genotype data, shape (2*individuals, #markers)
- gmap: `pd.DataFrame` Genetic map information.
- meff: `pd.DataFrame` Marker effects, shape `(#markers, #traits)`.
- exp_ldmat: `list` or `dict` Expected LD matrices by `msq.expldmat`.
- group: `pd.DataFrame` At least `[ID, Group]`.
- kwargs: (optional)
  - `indwt` : `np.ndarray` or `None` Index weights for multiple traits (e.g., `[0.2, 0.5, 0.3]`).
  - `sub_id` : `pd.DataFrame` or `None` A single-column DataFrame of IDs to subset. If `None`, uses all individuals.
  - `center` : `bool`; default `False` Center genotype matrix.
  - `progress` : `bool`; default `False` Whether to display a text-based progress bar.

Returns

- pd.DataFrame Mendelian sampling variances/covariances, typically with columns for [ID, Group, trait-var/cov, ...].
  - For single-trait cases, returns a single variance column.
  - For multi-trait, returns flattened lower-triangle covariance columns plus (optionally) an aggregate genotype column.

Raises

- `ValueError`
  - If dimension checks fail (e.g., #markers mismatch).
  - If multi-trait but indwt is missing.

Example

```
msvmsc = msq.msvarcov(gmat, gmap, meff, exp_ldmat, group, indwt=[0.1, 0.5, 0.4],
progress=True)
msvmsc.head()
```

4. `msq.msvarcov_corr(msvmsc)`

```
def msvarcov_corr(msvmsc):
    ...
```

Description

Converts (co)variance data from `msvarcov` into correlation data.

- Preserves the same [ID, Group] columns.
- Any flattened covariance columns become correlation columns.

Parameters

- msvmsc: (`pd.DataFrame`) The output of `msq.msvarcov`. Must have multiple traits (otherwise no covariance to convert).

Returns

- `pd.DataFrame` The same structure as the input, but where each (co)variance entry is replaced by the corresponding correlation.

Raises

- `ValueError` If there is only one trait (i.e., no off-diagonal covariances to standardize).

Example

```
msv_corr = msq.msvarcov_corr(msv)
msv_corr.head()
```

5. `msq.simmat(gmat, gmap, meff, group, exp_ldmat, **kwargs)`

```
def simmat(gmat, gmap, meff, group, exp_ldmat, **kwargs):
    ...
```

Description

Builds a similarity matrix of gametes or zygotes, highlighting how parental haplotypes might share large-effect heterozygous segments.

- Can compute for all individuals (gametes) or for mate pairs (zygotes) if `sub_id` has two columns.

Parameters

- gmat: `pd.DataFrame` or `np.ndarray` Phased genotypes `( 2 * #individulas, #markers )`.
- gmap: `pd.DataFrame` Genetic map info.
- meff: `pd.DataFrame` : Marker effects `( #markers x #traits )`.
- group: `pd.DataFrame` `[ID, Group]`.
- exp_ldmat: `list` or `dict` : From `msq.expldmat`.
- kwargs: (optional)
  - `sub_id` : `pd.DataFrame` or `None`
    - If None or 1-col => gametic approach
    - If 2-col => zygotic approach
  - `indwt:` `np.ndarray` or `None` Index weights for multi-trait.
  - `chrinterest` : `list` or `None` Selects chromosomes whose similarity matrix should be written to disk is save is True and chrinterest is not None.
  - `save` : `bool` ; default `False` : Save results to disk.
  - `stdsim` : `bool` ; default `False` Standardize similarity (divide by diagonal, etc.).
  - `center` : `bool` ; default `False` Center genotype matrix.
  - `progress` : `bool` ; default `False` Show progress bar.

Returns

- list or dict Similarity matrices, possibly a dict if multiple groups.

Raises

- `ValueError` If `sub_id` has invalid shape (not 1-col or 2-col) or if exp_ldmat is neither a list nor dict.

Example

```
sim_g = msq.simmat(gmat, gmap, meff, group, exp_ldmat)
```

6. `msq.selstrat(gmat, meff, group, **kwargs)`

```
def selstrat(gmat, meff, group, **kwargs):
    ...
```

Description

Computes selection criteria—GEBV, UC (usefulness criterion), or index-based—for either gametes or mate pairs.

- If `sub_id` has one column, calculates selection values per individual.
- If `sub_id` has two columns ([MaleID, FemaleID]), calculates for zygotes from each pair.

Parameters

- gmat: `pd.DataFrame` or `np.ndarray` Genotype data. If phased/haplotypic, shape = `(2*#individuals, #markers)`.
- meff pd.DataFrame `Marker effects` (#markers, #traits)`.
- group: `pd.DataFrame` `[ID, Group]` data.
- kwargs: (optional)
  - `indwt` : `np.ndarray` or `None` Index weights for multi-trait.
  - `sub_id` : `pd.DataFrame` or `None`
    - 1-col => individuals
    - 2-col => pair (zygotes)
  - `msvmsc:` pd.DataFrame) Required for `uc` or `index`.
  - `criterion` : `{"gebv","uc","index"}` ; default `"gebv"`
    - `"gebv"` => basic GEBVs
    - `"uc"` => usefulness criterion => GEBV + k * MSV weighting (requires `prop_sel` & `msvmsc`)
    - `"index"` => GEBV + k * MSV weighting (requires `prop_sel` & ``msvmsc```)
  - `prop_sel` : `float` Proportion of selection `[0,1]`, for `uc` or `index`.
  - `aggregate` : `bool` ; default `True` Whether to use the aggregate genotype "AG" column for multi-trait data.
  - `haplotype` : `bool` ; default `True` If True, gmat is haplotypes.
  - `center` : `bool` ; default `False` Center marker data.
  - `progress` : `bool` ; default `False` Show progress bar.

Returns

- `pd.DataFrame` Each row = an individual (if 1-col sub_id) or a pair (if 2-col sub_id). Columns typically include:
  - `[ID, Group, GEBV or UC or Index, (and for multi-trait: trait-specific columns + aggregate breeding value “ABV”)]`.

Raises

- `ValueError`
  - If multi-trait but `indwt` is missing.
  - If `msvmsc` or p `rop_sel` is missing when using uc/index.
  - If `criterion` is not one of the supported strategies.

Example

```python
# GEBV for all individuals:
res_gebv = msq.selstrat(gmat, meff, group, criterion="gebv", haplotype=True)

# UC for a subset (single-column sub_id) at 30% selection:
res_uc = msq.selstrat(
    gmat, meff, group,
    sub_id=sub_id_df,
    msvmsc=msvmsc_df,
    criterion="uc",
    prop_sel=0.3
)
res_uc.head()
```

Additional Notes

- Data Shapes: Most functions assume phased genotype data in shape `(2*#individuals, #markers)`. If single-trait, indwt is optional; multi-trait requires specifying `indwt`.
- ValueError Checks: Dimension mismatches or missing columns typically raise `ValueError`.
- List/Dict Structures: Some routines (`expldmat`, `simmat`) can produce or consume `list` vs. `dict` if multiple groups are present.
- File I/O: Save/load uses pandas or NumPy `.npy` formats when indicated by `kwargs` (e.g., `save=True`).
- Performance: For large datasets (many markers or individuals) and limited RAM, consider chromosome-wise analysis and summing up results.