Illustration: Utilizing Functions of PyMSQ

Authors: Abdulraheem Musa and Norbert Reinsch
Last Updated: 08.04.2025

Introduction

This guide demonstrates **practical applications** of **PyMSQ** functions using an example dataset of **Holstein-Friesian cattle**. PyMSQ is a **comprehensive tool** for deriving Mendelian sampling–related quantities such as (co)variances, correlations, and similarity matrices. The workflow involves:

1. **Importing** and **preparing** data
2. **Computing** expected LD matrices
3. **Estimating** Mendelian sampling (co)variance and correlations
4. **Deriving** similarity matrices (gametes or zygotes)
5. **Applying** selection strategies (GEBV, UC, index)

Throughout the examples, we use a Holstein-Friesian dataset with 265 cows, 10,304 markers, and multiple milk traits. This data is bundled with PyMSQ for illustrative purposes.

---

1. Importing Necessary Packages and Module

In [1]:
```
import numpy as np
import pandas as pd
import time  # to measure runtimes in examples

# Import the msq module from PyMSQ
from PyMSQ import msq

# Or equivalently:
# from PyMSQ.msq import load_package_data, expldmat, msvarcov, ...
```

```
C:\Users\musa\AppData\Roaming\Python\Python39\site-packages\pandas\core\arrays\masked.p
y:60: UserWarning: Pandas requires version '1.3.6' or newer of 'bottleneck' (version '1.
3.5' currently installed).
  from pandas.core import (
```

Note: Ensure you have installed PyMSQ (e.g., pip install PyMSQ).

2. Data Importation and Preparation

2.1 Dataset Overview

We use an example Holstein-Friesian cattle dataset included with PyMSQ. It comprises:

- **265 cows** in 5 half-sib families,
- **29 autosomal chromosomes** with 10,304 markers,
- **Marker effects** for key milk traits (fat, protein, pH),
- And **phenotypic/group** information.

This dataset is detailed in Musa and Reinsch [1] and derived from prior studies [2, 3]. You can substitute your own data if it matches the input format described below.

2.2. Loading Example Data

- **Function**: `msq.load_package_data()`
- **Purpose**: Retrieves example data from local files packaged with PyMSQ.
- **Usage**:

In [2]:
```
# Loading example data
data = msq.load_package_data()

# Extracting key DataFrames
gmap  = data['chromosome_data']      # Genetic map info
meff  = data['marker_effect_data']   # Marker effects
gmat  = data['genotype_data']        # Phased genotypes
group = data['group_data']           # Group/phenotypic info

# Quick checks
print("Genetic map:")
print(gmap.head(), "\n")

print("Marker effects:")
print(meff.head(), "\n")

print("Phased genotypes:")
print(gmat.head(), "\n")

print("Group/phenotypic info:")
print(group.head())
```

```
Genetic map:
   CHR SNPName  Position    group1
0    1    SNP1    113641  0.113641
1    1    SNP2    244698  0.244698
2    1    SNP3    369418  0.369418
3    1    SNP4    447277  0.447277
4    1    SNP5    487653  0.487653

Marker effects:
        fat   protein        pH
0  0.000059 -0.000211 -0.000163
1 -0.000051 -0.000006 -0.000773
2  0.000034 -0.000075 -0.000047
3  0.000026 -0.000118  0.000101
4  0.000021  0.000075  0.000066

Phased genotypes:
       0                                                     1
0  10001  12222212222222211221111122222222221122221222 2...
1  10001  22221221222222222222222222122222222222222221 2...
2  10002  22112222222222222222222212211211222222222222 2...
3  10002  22222221222222222222222221211221222212222112 1...
4  10003  22112222222222222222222212211211222222222222 2...

Group/phenotypic info:
      ID group
0  10001     F
1  10002     F
2  10003     F
3  10004     F
4  10005     F
```

2.3 Data Requirements & Structures

1. Group / Phenotypic Data (group)
   - Must have at least two columns:
     - Column 1: Individual IDs
     - Column 2: Group classification (e.g., “M”/“F”)
   - You can manually edit the group column to create subgroups, e.g., sexes.

In [3]:
```
print("Example group DataFrame:\n", group.head())
group_sexes = group.copy()
group_sexes.iloc[0:130, 1] = "M"  # Assume first 130 are males
group_sexes.iloc[130:, 1]  = "F"  # Remaining as females
```

```
Example group DataFrame:
       ID group
0  10001     F
1  10002     F
2  10003     F
3  10004     F
4  10005     F
```

1. **Genetic Map** (`gmap`)

- A DataFrame where:
  - Column 1: Chromosome ID (integer or name),
  - Column 2: Marker name/ID,
  - Column 3: Physical position or base pair coordinate,
  - Columns 4+: cM or recombination rates, possibly multiple columns if group-specific.
- If using recombination rates, the first marker in each chromosome should be 0.

In [4]:
```
print(gmap.head())  # Displaying the structure of the genetic map
```

```
   CHR SNPName  Position    group1
0    1    SNP1    113641  0.113641
1    1    SNP2    244698  0.244698
2    1    SNP3    369418  0.369418
3    1    SNP4    447277  0.447277
4    1    SNP5    487653  0.487653
```

1. **Marker effects** or **Allele substitution effects** (`meff`):

- Rows = markers, columns = trait names.
- For multi-trait scenarios, each column is a separate trait’s marker effects.

In [5]:
```
print("Example marker effects:\n", meff.head())
# Suppose columns = ["fat", "protein", "pH"]
```

```
Example marker effects:
         fat   protein        pH
0  0.000059 -0.000211 -0.000163
1 -0.000051 -0.000006 -0.000773
2  0.000034 -0.000075 -0.000047
3  0.000026 -0.000118  0.000101
4  0.000021  0.000075  0.000066
```

1. **Phased genotypes** `gmat` :

- Typically shape: (2 * N_individuals, #markers). Each individual has 2 rows (paternal haplotype, maternal haplotype).
- Alternatively, a single string column per row (e.g., "010210...") that PyMSQ will parse.

In [6]:
```
print("Example phased genotypes:\n", gmat.head())
```

```
Example phased genotypes:
        0                                                     1
0  10001  12222212222222211221111122222222221122221222 2...
1  10001  22221221222222222222222221222222222222222221...
2  10002  22112222222222222222222212211211222222222222...
3  10002  22222212222222222222222212112212222122221121...
4  10003  22112222222222222222222212211211222222222222...
```

Important: All genotypes must be biallelic (allows 0-9 coding) and free of missing codes.

3. Practical Use of PyMSQ Functions

This section illustrates key PyMSQ routines:

1. `expldmat` : Computing expected within-family LD matrices.
2. `msvarcov` : Calculating Mendelian sampling (co)variance.
3. `msvarcov_corr` : Converting (co)variance to correlation.
4. `simmat` : Building similarity matrices for gametes or zygotes.
5. `selstrat` : Running selection strategies (GEBV, UC, Index).

3.1. Deriving the Expected Within-Family LD Matrix

Function: `msq.expldmat(gmap, group, **kwargs)` **Purpose**: Compute the expected within-family LD matrix $**R**$ for each chromosome, using cM or recombination rates.

In [7]:
```
# Example 1: Default usage (mposunit="cM", method=1, threshold=None)
start = time.time()
exp_ldmat_default = msq.expldmat(gmap, group)
print("Time taken:", round(time.time() - start, 2), "sec")

# Inspect the first chromosome's LD matrix
ld_chr1 = exp_ldmat_default[0][0]  # If single group, a list of lists
print("LD matrix for chromosome1, method=1:\n", ld_chr1)

# Example 2: Using Santos' method = 2, threshold=50 cM
exp_ldmat_santos = msq.expldmat(gmap, group, mposunit="cM", method=2, threshold=50)
ld_chr1_santos = exp_ldmat_santos[0][0]
print("LD matrix for chromosome1, method=2, threshold=50:\n", ld_chr1_santos)
```

```
Time taken: 0.04 sec
LD matrix for chromosome1, method=1:
 [[0.25       0.24934559 0.24872437 ... 0.01002053 0.01001573 0.01000675]
 [0.24934559 0.25       0.24937719 ... 0.01004683 0.01004202 0.01003301]
 [0.24872437 0.24937719 0.25       ... 0.01007192 0.0100671  0.01005807]
 ...
 [0.01002053 0.01004683 0.01007192 ... 0.25       0.24988016 0.24965608]
 [0.01001573 0.01004202 0.0100671  ... 0.24988016 0.25       0.24977581]
 [0.01000675 0.01003301 0.01005807 ... 0.24965608 0.24977581 0.25      ]]
LD matrix for chromosome1, method=2, threshold=50:
 [[0.25       0.24934472 0.24872111 ... 0.         0.         0.        ]
 [0.24934472 0.25       0.2493764  ... 0.         0.         0.        ]
```

```
 [0.24872111 0.2493764  0.25       ... 0.         0.         0.        ]
 ...
 [0.         0.         0.         ... 0.25       0.24988014 0.24965584]
 [0.         0.         0.         ... 0.24988014 0.25       0.2497757 ]
 [0.         0.         0.         ... 0.24965584 0.2497757  0.25      ]]
```

If multiple group-specific columns in gmap exist (e.g., "M" vs. "F"), expldmat returns a dict keyed by group name, with each value holding the per-chromosome LD matrices.

3.2. Estimating Mendelian Sampling (Co)variance

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

Purpose: Compute each individual's Mendelian sampling variance (MSV) for single or multiple traits.

In [8]:
```
# Basic multi-trait scenario
index_weights = [1, 1, 1]  # e.g., equal weighting for 3 traits
start = time.time()
msvmsc_g = msq.msvarcov(
    gmat       = gmat,
    gmap       = gmap,
    meff       = meff,
    exp_ldmat  = exp_ldmat_default,
    group      = group,
    indwt      = index_weights,
    center     = False,
    progress   = True
)
runtime = round(time.time() - start, 2)
print(f"Computed MSV for all individuals in {runtime} sec.\n", msvmsc_g.head())

# Subset usage
selected_ids = group.iloc[[0, 10, 20, 30], 0]  # picking 4 IDs
msvmsc_subset = msq.msvarcov(
    gmat       = gmat,
    gmap       = gmap,
    meff       = meff,
    exp_ldmat  = exp_ldmat_default,
    group      = group,
    indwt      = index_weights,
    sub_id     = selected_ids
)
print("Subset-based MSV:\n", msvmsc_subset)

# Group-specific maps usage
gmap_group_specific = gmap.copy()
# Insert the 4th column's values as a new column named "group2" at position 4
gmap_group_specific.insert(4, "group2", gmap_group_specific.iloc[:, 3])
# Derive group secific LD matrices
exp_ldmat_group_specific = msq.expldmat(gmap_group_specific, group_sexes)
msvmsc_sex_map = msq.msvarcov(
    gmat      = gmat,
    gmap      = gmap_group_specific,
    meff      = meff,
    exp_ldmat = exp_ldmat_group_specific,
    group     = group_sexes,
    indwt     = index_weights
)
print("MSV with group-specific map:\n", msvmsc_sex_map.head())
```

```
Formatting phased haplotypes
phased genotype data has 530 rows and 10304 columns
Allele 1: 829180
```

```
Allele 2: 4631940
Major allele: 2
Progress: |##################################################| 100% Complete
Computed MSV for all individuals in 2.76 sec.
       ID Group       fat   protein_fat   protein    pH_fat  pH_protein  \
0  10001     F  0.000023  1.895190e-06  0.000115  0.000011   -0.000001
1  10002     F  0.022089  1.588602e-02  0.011507  0.012589    0.009055
2  10003     F  0.022267  1.597741e-02  0.011675  0.013504    0.009686
3  10004     F  0.000033 -1.067220e-07  0.000102  0.000032   -0.000019
4  10005     F  0.022131  1.585642e-02  0.011514  0.012576    0.009006

         pH    AG_fat  AG_protein     AG_pH        AG
0  0.000135  0.000036    0.000115  0.000145  0.000296
1  0.007375  0.050564    0.036448  0.029018  0.116029
2  0.008541  0.051748    0.037338  0.031731  0.120818
3  0.000376  0.000065    0.000083  0.000389  0.000536
4  0.007419  0.050564    0.036377  0.029002  0.115943
phased genotype data has 530 rows and 10304 columns
Subset-based MSV:
       ID Group       fat  protein_fat   protein    pH_fat  pH_protein  \
0  10001     F  0.000023     0.000002  0.000115  0.000011   -0.000001
1  10011     F  0.000022     0.000020  0.000173  0.000008   -0.000012
2  10021     F  0.022280     0.016096  0.011705  0.014233    0.010277
3  10031     F  0.000021     0.000014  0.000095 -0.000002    0.000020

         pH    AG_fat  AG_protein     AG_pH        AG
0  0.000135  0.000036    0.000115  0.000145  0.000296
1  0.000086  0.000050    0.000181  0.000083  0.000314
2  0.009365  0.052608    0.038078  0.033875  0.124562
3  0.000365  0.000032    0.000128  0.000382  0.000543
phased genotype data has 530 rows and 10304 columns
MSV with group-specific map:
       ID Group       fat   protein_fat   protein    pH_fat  pH_protein  \
0  10001     M  0.000023  1.895190e-06  0.000115  0.000011   -0.000001
1  10002     M  0.022089  1.588602e-02  0.011507  0.012589    0.009055
2  10003     M  0.022267  1.597741e-02  0.011675  0.013504    0.009686
3  10004     M  0.000033 -1.067220e-07  0.000102  0.000032   -0.000019
4  10005     M  0.022131  1.585642e-02  0.011514  0.012576    0.009006

         pH    AG_fat  AG_protein     AG_pH        AG
0  0.000135  0.000036    0.000115  0.000145  0.000296
1  0.007375  0.050564    0.036448  0.029018  0.116029
2  0.008541  0.051748    0.037338  0.031731  0.120818
3  0.000376  0.000065    0.000083  0.000389  0.000536
4  0.007419  0.050564    0.036377  0.029002  0.115943
```

Key Columns in the returned DataFrame usually include:

- **ID** and **Group**,
- One column per single trait's variance (and possibly their covariances),
- An "AG" (aggregate genotype) column if multi-trait with `indwt` specified.

3.3. Converting Mendelian Sampling Covariances to Correlations

Function: `msq.msvarcov_corr(msvmsc)`

Purpose: Convert the lower-triangular (co)variance columns to correlations.

In [9]:
```
msvmsc_gcorr = msq.msvarcov_corr(msvmsc=msvmsc_g)
print(msvmsc_gcorr.head())
```

```
      ID Group  protein_fat    pH_fat  pH_protein    AG_fat  AG_protein  \
0  10001     F     0.036531  0.195637   -0.009354  0.436113    0.625978
1  10002     F     0.996419  0.986304    0.982884  0.998772    0.997474
2  10003     F     0.990927  0.979225    0.969887  0.997708    0.994151
3  10004     F    -0.001846  0.288299   -0.098526  0.487664    0.353708
4  10005     F     0.993303  0.981447    0.974409  0.998197    0.995601

      AG_pH
0  0.723546
1  0.991987
2  0.987773
3  0.865729
4  0.988830
```

Note: Raises a ValueError if only one trait (no covariance).

3.4. Deriving Similarity Matrices

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

Purpose: Compute similarity matrices for:

- **Gametes** (1-col sub_id or None),
- **Zygotes** (2-col sub_id: [MaleID, FemaleID]).

In [10]:

```python
# Example 1: Gametic similarity for all individuals
sim_gametes = msq.simmat(
    gmat      = gmat,
    gmap      = gmap,
    meff      = meff,
    group     = group,
    exp_ldmat = exp_ldmat_default,
    indwt     = index_weights,
    stdsim    = False,        # no standardization
    progress  = True
)
print("Gametic similarity:\n", sim_gametes)

# Example 2: Standardized similarity, restricted subset
some_ids = group.iloc[:10, 0]  # first 10 individuals
sim_gametes_std = msq.simmat(
    gmat      = gmat,
    gmap      = gmap,
    meff      = meff,
    group     = group,
    exp_ldmat = exp_ldmat_default,
    sub_id    = some_ids,
    indwt     = index_weights,
    stdsim    = True,         # standardize
    center    = True
)
print("Standardized similarity matrix for 10 individuals:\n", sim_gametes_std)

# Example 3: Zygotic similarity for parent pairs
mate_pairs = pd.DataFrame({
    "Male_ID":   [10001, 10002, 10003],
    "Female_ID": [10261, 10262, 10263]
})
sim_zygotes = msq.simmat(
    gmat         = gmat,
    gmap         = gmap_group_specific,
    meff         = meff,
```

```
    group        = group_sexes,
    exp_ldmat    = exp_ldmat_group_specific,
    sub_id       = mate_pairs,
    indwt        = index_weights,
    chrinterest  = None,    # or specific chromosomes
    stdsim       = True,
    progress     = True
)
print("Zygotic similarity for parent pairs:\n", sim_zygotes)
```

```
Formatting phased haplotypes
phased genotype data has 530 rows and 10304 columns
Allele 1: 829180
Allele 2: 4631940
Major allele: 2
Progress: |##################################################| 100% Complete
Creating similarity matrix based on aggregate genotype
Progress: |##################################################| 100% Complete
Gametic similarity:
 [array([[0.00029632, 0.0003095 , 0.00035975, ..., 0.0001671 , 0.00026274,
         0.00027542],
        [0.0003095 , 0.11602936, 0.11822478, ..., 0.00084436, 0.11597751,
         0.11934897],
        [0.00035975, 0.11822478, 0.12081815, ..., 0.00094194, 0.11827583,
         0.12172188],
        ...,
        [0.0001671 , 0.00084436, 0.00094194, ..., 0.00043942, 0.0007067 ,
         0.0007694 ],
        [0.00026274, 0.11597753, 0.11827582, ..., 0.0007067 , 0.11631628,
         0.11959661],
        [0.00027542, 0.11934897, 0.12172189, ..., 0.0007694 , 0.11959659,
         0.12324664]], dtype=float32)]
phased genotype data has 20 rows and 10304 columns
Creating similarity matrix based on aggregate genotype
Standardized similarity matrix for 10 individuals:
 [array([[1.        , 0.9969873 , 0.99560523, 0.9969822 , 0.9968444 ,
         0.99743974, 0.99708706, 0.99743885, 0.9972078 , 0.99442196],
        [0.99698645, 1.        , 0.9965543 , 0.996747  , 0.9971177 ,
         0.99782145, 0.99778724, 0.9978411 , 0.99647135, 0.99445313],
        [0.99560404, 0.99655426, 1.        , 0.99580735, 0.9954717 ,
         0.9963611 , 0.9963444 , 0.9959287 , 0.9958141 , 0.99449515],
        [0.99698144, 0.99674803, 0.99580866, 1.        , 0.9960099 ,
         0.9966733 , 0.996063  , 0.9964043 , 0.996654  , 0.99438465],
        [0.9968437 , 0.9971175 , 0.99547184, 0.99600893, 1.        ,
         0.9969095 , 0.9967607 , 0.9973844 , 0.99724543, 0.99437654],
        [0.9974386 , 0.9978213 , 0.9963611 , 0.9966718 , 0.9969095 ,
         1.        , 0.9979273 , 0.99779   , 0.99621075, 0.9954104 ],
        [0.99708647, 0.9977878 , 0.99634516, 0.996063  , 0.9967614 ,
         0.99792814, 1.        , 0.99793285, 0.99616843, 0.9964428 ],
        [0.99743766, 0.9978412 , 0.99592966, 0.9964032 , 0.99738467,
         0.99779046, 0.99793243, 1.        , 0.9968518 , 0.99602884],
        [0.9972069 , 0.9964716 , 0.9958148 , 0.99665356, 0.99724585,
         0.99621147, 0.9961686 , 0.9968521 , 1.        , 0.99454737],
        [0.99442106, 0.9944548 , 0.9944967 , 0.9943849 , 0.9943769 ,
         0.995411  , 0.99644274, 0.9960302 , 0.9945471 , 1.        ]],
      dtype=float32)]
Processing M
Formatting phased haplotypes
phased genotype data has 6 rows and 10304 columns
Allele 1: 9425
Allele 2: 52399
Major allele: 2
Processing F
Formatting phased haplotypes
phased genotype data has 6 rows and 10304 columns
Allele 1: 9471
```

```
Allele 2: 52353
Major allele: 2
Processing similarity matrices for zygotes
Progress: |##################################################| 100% Complete
Creating similarity matrix based on aggregate genotype
Progress: |##################################################| 100% Complete
Zygotic similarity for parent pairs:
 [[1.         0.721888   0.00930449]
 [0.72188777 1.         0.6952067 ]
 [0.00930449 0.6952067  1.        ]]
```

3.5. Computing Selection Criteria

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

Purpose: Calculate GEBV [4], usefulness criterion (UC) [5], or multi-trait index (adapted from [6]) for gametes or zygotes.

In [11]:
```python
# Example 1: Single-trait GEBV for entire dataset
single_trait_meff = meff.iloc[:, [0]]  # e.g., "fat" only
res_gebv = msq.selstrat(
    gmat       = gmat,
    meff       = single_trait_meff,
    group      = group,
    criterion  = "gebv",
    haplotype  = True,
    center     = False
)
print("Single-trait GEBV:\n", res_gebv.head())

# Example 2: Multi-trait index with 10% selection
res_index = msq.selstrat(
    gmat       = gmat,
    meff       = meff,
    group      = group,
    indwt      = index_weights,
    msvmsc     = msvmsc_g,
    criterion  = "index",
    prop_sel   = 0.1,
    aggregate  = True
)
print("Multi-trait index selection:\n", res_index.head())

# Example 3: Zygotic approach (two-column sub_id)
zygotic_subid = pd.DataFrame({
    "Male_ID":   [10001, 10002],
    "Female_ID": [10261, 10262]
})
zygote_gebv = msq.selstrat(
    gmat       = gmat,
    meff       = meff,
    group      = group_sexes,
    indwt      = index_weights,
    sub_id     = zygotic_subid,
    criterion  = "gebv"
)
print("Zygotic approach (GEBV):\n", zygote_gebv)
```

```
phased genotype data has 530 rows and 10304 columns
Single-trait GEBV:
       ID Group       fat
0  10001     F  0.231010
1  10002     F -0.076839
```

```
2  10003     F -0.070086
3  10004     F  0.231345
4  10005     F -0.037961
phased genotype data has 530 rows and 10304 columns
Multi-trait index selection:
       ID Group       fat   protein        pH       ABV
0  10001     F  0.363604  0.242749  0.173021  0.754689
1  10002     F  0.412604  0.270716  0.216208  0.896694
2  10003     F  0.420838  0.323337  0.260004  0.998957
3  10004     F  0.366108  0.218948  0.226536  0.781639
4  10005     F  0.451835  0.248698  0.224403  0.920710
phased genotype data has 530 rows and 10304 columns
Zygotic approach (GEBV):
     M_ID   F_ID       fat   protein        pH       ABV
0  10001  10261  0.187088  0.102256  0.092694  0.382038
1  10002  10262  0.039162 -0.007854  0.016210  0.047518
```

Key:

- `"gebv"` => basic GEBVs
- `"uc"` => usefulness criterion => GEBV + k * MSV weighting (requires `prop_sel` & `msvmsc`)
- `"index"` => GEBV + k * MSV weighting (requires `prop_sel` & ``msvmsc```)

Concluding Remarks

This illustration walks through typical PyMSQ workflows, from data loading to Mendelian sampling analysis, similarity matrix generation, and selection strategy application. Adjust parameters, subset IDs, or marker maps to match your breeding context. For more advanced or large-scale scenarios (e.g., thousands of animals, high-density markers), you may want to split data by chromosome.

If you have questions or feature requests, please consult the PyMSQ GitHub repository for issue reporting and community support.

References

1. Musa, A. A., & Reinsch, N. (2025). A similarity matrix for hedging haplotype diversity among parents in genomic selection. *Journal of Animal Breeding and Genetics*, https://doi.org/10.1111/jbg.12930.

2. Melzer, N., Wittenburg, D., & Repsilber, D. (2013). Integrating milk metabolite profile information for the prediction of traditional milk traits based on SNP information for Holstein cows. *PLoS ONE*, *8*, e70256.

3. Hampel, A., Teuscher, F., Gomez-Raya, L., Doschoris, M., & Wittenburg, D. (2018). Estimation of recombination rate and maternal linkage disequilibrium in half-sibs. *Frontiers in Genetics*, *9* (JUN), 186.

4. Meuwissen, T. H. E., Hayes, B. J., & Goddard, M. E. (2001). Prediction of total genetic value using genome-wide dense marker maps. *Genetics*, *157*, 1819–1829.

5. Lehermeier, C., Teyssèdre, S., & Schön, C. C. (2017). Genetic gain increases by applying the usefulness criterion with improved variance prediction in selection of crosses. *Genetics*, *207*, 1651–1661.

6. Bijma, P., Wientjes, Y. C. J., & Calus, M. P. L. (2020). Breeding top genotypes and accelerating response to gametic variance. *Genetics*, *214*, 91–107.