Illustration: Using functions of PyMSQ

Import relevant packages and modules

In [1]:
```
from PyMSQ import msq
import numpy as np
import time
```

Importation of example data and expected data formats

In the following, we import a Holstein-Friesian cattle dataset with 265 cows from five half-sib families from Musa and Reinsch [1], a subset of data from previous studies [2, 3]. The dataset contains 39780 markers for 29 (autosomal) chromosomes and marker effects for three example milk traits.

The genetic map, marker effects, phased genotypes, and phenotypic information can be imported using the function `msq.example_data()` :

In [2]:
```
gmap, meff, gmat, group, _ = msq.example_data()
```

The main information required by the functions in the msq module and how they should be provided are explained below:

1. `Phenotypic information` : This information should be provided as a pandas data frame with two columns. As indicated below, the first column should provide group classification (e.g., sex, breed, or strain), whereas the second column should provide identification numbers (ID) of individuals:

In [3]:
```
print(group)
```

```
    group     ID
0       F  10001
1       F  10002
2       F  10003
3       F  10004
4       F  10005
..    ...    ...
260     F  10261
261     F  10262
262     F  10263
263     F  10264
264     F  10265

[265 rows x 2 columns]
```

All individuals in this data are females (F). In order to demonstrate the use of some functions for zygotes, let us assume the first 130 individuals are males (M) like:

In [4]:
```
group.iloc[0:130, 0] = "M"
print(group)
```

```
    group     ID
0       M  10001
1       M  10002
2       M  10003
3       M  10004
4       M  10005
..    ...    ...
260     F  10261
261     F  10262
262     F  10263
263     F  10264
264     F  10265

[265 rows x 2 columns]
```

1. `Genetic map` : The genetic map should be provided as a pandas data frame. The first three columns must be the chromosome number, marker name, and marker position in base pairs. The remaining column(s) should be maker distance/position (cM) or recombination rates in the order listed in the group column of the phenotypic information (1). Let us assume the group classification is sex. If males are listed first, then the fourth column should be the marker distance/position (or recombination rates) for males. The fifth column should be the marker distance/position (or recombination rates) for females. If an average map for all groups is provided, the data frame will contain four columns only, with the average marker distance/position as the fourth column (or recombination rates). Below is an example of how the genetic map should be provided:

In [5]:
```
print(gmap)
```

```
       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
...    ...       ...       ...        ...
39775   29  SNP39776  51899151  51.899151
39776   29  SNP39777  51920849  51.920849
39777   29  SNP39778  51986600  51.986600
39778   29  SNP39779  52030414  52.030414
39779   29  SNP39780  52112161  52.112161

[39780 rows x 4 columns]
```

Let us add another column to the data frame to simulate a data frame with group-specific maps.

In [6]:
```
gmap.insert(4, "group2", gmap.iloc[:, 3], True)
print(gmap)
```

```
       CHR   SNPName  Position     group1     group2
0        1      SNP1    113641   0.113641   0.113641
1        1      SNP2    244698   0.244698   0.244698
2        1      SNP3    369418   0.369418   0.369418
3        1      SNP4    447277   0.447277   0.447277
4        1      SNP5    487653   0.487653   0.487653
...    ...       ...       ...        ...        ...
39775   29  SNP39776  51899151  51.899151  51.899151
39776   29  SNP39777  51920849  51.920849  51.920849
39777   29  SNP39778  51986600  51.986600  51.986600
39778   29  SNP39779  52030414  52.030414  52.030414
39779   29  SNP39780  52112161  52.112161  52.112161

[39780 rows x 5 columns]
```

PyMSQ will use group1 as the marker distance/position (or recombination rates) for males and group2 for females because males are first in the data frame for phenotypic information.

Note: When using recombination rates, the first marker on a chromosome must be zero, followed by recombination rate between markers m and m+1. However, the first element on each chromosome does not have to be zero when using genetic position/distance (specified in cM) because the function utilizes the differences between given values. This allows any subset of markers from a genetic map to be conveniently used by copying their physical (base-pairs) and genetic map (cM) positions.

1. `Allele substitution effects` : The allele substitution or marker effects should be provided as a pandas data frame with named columns. This data frame provides the trait names used in PyMSQ functions. As a result, users should name these columns according to how they want them to appear in the results. The number of rows of this data frame should be equal to the number of markers, and the number of columns should be equal to the number of traits plus one. The first column should be the marker name (same as in the genetic map), and the remaining column(s) should be allele substitution or marker effects for each trait. Here is an example:

In [7]:
```
print(meff)
```

```
        SNPName       fat        pH   protein
0          SNP1  0.000059 -0.000163 -0.000211
1          SNP2 -0.000051 -0.000773 -0.000006
2          SNP3  0.000034 -0.000047 -0.000075
3          SNP4  0.000026  0.000101 -0.000118
4          SNP5  0.000021  0.000066  0.000075
...         ...       ...       ...       ...
39775  SNP39776 -0.000104 -0.000078 -0.000017
39776  SNP39777  0.000205 -0.000145 -0.000053
39777  SNP39778  0.000034  0.000025  0.000043
39778  SNP39779 -0.000148 -0.000033 -0.000003
39779  SNP39780 -0.000059 -0.000004  0.000013

[39780 rows x 4 columns]
```

1. `Phased genotypes` : Phased genotypes: Individuals' paternal and maternal haplotypes should be arranged in rows with their IDs. The phased genotypes should also be provided as a pandas data frame in one of the following ways:

- i: as strings array – The data frame should have two columns (ID and haplotypes as strings without spaces), and the number of rows should be 2 * the number of individuals, as follows:

In [8]:
```
print(gmat)
```

```
         0                                                  1
0    10001  0222222022222222222222200222220222000002000222222...
1    10001  2222020202222220222222222022222222222222220022...
2    10002  2200222222220022222222202222222222222220002222...
3    10002  2222222022202222222222202222222222222222000222...
4    10003  2200222222220022222222202222222222222220002222...
..     ...                                                ...
525  10263  2000222222222222000022022222222222222220002222...
526  10264  0222220222222222222200222202220000222222222022...
527  10264  2222222022202222222222022202022222222222222222...
528  10265  2222222222222222222222022222222222222222222222...
529  10265  2200222222220022222222202222222222222220002222...

[530 rows x 2 columns]
```

- ii: as an integer array. Here the number of rows is still the same as above, but the number of columns is 1 (ID number) + the number of markers.

Note: Since genotypes are already phased into haplotypes, PyMSQ does not expect missing values and, therefore, does not accept data with missing values. It does, however, accept any coding for phased haplotypes as long as they are biallelic (i.e., two integers must be used for coding) and taken from integers ranging from 0 to 9. As a result, outputs from several genotype phasing software can be used with ease.

Checking the input information for errors

PyMSQ requires that the main information, including index weights, be checked for errors using the function msq.Datacheck to enhance the smooth running of other functions. Briefly, it performs checks to ensure the following:

- The number of traits matches the length of index weights.
- The ID of individuals in phenotypic information and phased genotypes are ordered, and the same.
- There are no missing allele substitution or marker effects, and all entries are numeric.
- Marker names in the genetic map and marker effects are ordered, and the same.
- The number of maps is the same as the number of groups if the map number is more than one.
- The number of markers in phased genotypes, genetic map, and marker effects is the same.
- The marker distance/position or recombination rates on each chromosome are ordered.

A corresponding error message is printed to screen if any of the above checks fail.

In addition to checking for errors, the function transforms phased haplotypes to genotypes and saves all relevant information as a class object, subsequently used by other functions.

In [9]:
```
index_wt = [1, 1, 1]
start = time.time()
data = msq.Datacheck(gmap=gmap, meff=meff, gmat=gmat, group=group,
                     indwt=index_wt, progress=True)
print('Time taken: ', round(time.time() - start, 2), 'secs')
```

```
Converting phased haplotypes to genotypes
Data passed the test!
Number of individuals:   265
Number of groups:        2 :  ['M' 'F']
Number of specific maps: 2
Number of chromosomes:   29
Total no. markers:       39780
Number of trait(s):      3
Trait name(s) and Index weight(s)
fat :  1
pH :  1
protein :  1
Time taken:  2.9 secs
```

The main information may be deleted because they are already stored as a class object.

In [10]:
```
del gmap, meff, gmat, group, index_wt
```

Setting up the population covariance matrix

The population covariance matrix is derived using function `msq.popcovmat` according to Bonk et al. [4] if parameter method=1 or Santos et al. [5] if method=2. The map type must be specified using mposunit= "cM" for genetic position/distance (centimorgan), or mposunit= "reco" for recombination rates. If mposunit='reco', an additional check is run to ensure that the first marker on each chromosome has a value of zero. An error message is displayed on the screen if the value is not zero. When group-specific maps are available, this function returns a named list of lists containing population covariance matrices (for each chromosome) for each group.

In [11]:
```
start = time.time()
covmatrices = msq.popcovmat(info=data, mposunit="cM", method=1)
print('Time taken: ', round(time.time() - start, 2), 'secs')
```

```
Time taken:  3.94 secs
```

Estimation of Mendelian sampling (co)variance for gametes produced by an individual

The function `msq.msvarcov_g` can be used to estimate the Mendelian sampling variance (MSV) for each trait, MSV of aggregate genotype (AG) for multiple traits, as well as their covariances. As a result, a data frame with trait names indicating Mendelian sampling variance for those traits, as well as a combination of trait names separated by "_" indicating Mendelian covariances between those traits, is created.

In [12]:
```
start = time.time()
msvmsc_g = msq.msvarcov_g(info=data, covmat=covmatrices,
                          sub_id=None, progress=True)
print('Time taken: ', round(time.time() - start, 2), 'secs')
print(msvmsc_g)
```

```
Progress: |██████████████████████████████████████████████████| 100% Complete
Time taken:  25.21 secs
        ID Group       fat    pH_fat        pH  protein_fat  protein_pH  \
0    10001     M  0.000466  0.000154  0.001805     0.000136   -0.000157
1    10002     M  0.023262  0.014908  0.013225     0.017306    0.011082
2    10003     M  0.021665  0.013760  0.010475     0.015662    0.009849
3    10004     M  0.000949  0.000317  0.004836    -0.000044   -0.000158
4    10005     M  0.021892  0.014447  0.012676     0.015576    0.010120
..     ...   ...       ...       ...       ...          ...         ...
260  10261     F  0.022112  0.012953  0.010050     0.017090    0.009524
261  10262     F  0.023853  0.016183  0.012896     0.017691    0.012189
262  10263     F  0.000604  0.000096  0.002694     0.000025    0.000126
263  10264     F  0.022809  0.015301  0.012748     0.017123    0.011174
264  10265     F  0.022491  0.016226  0.013892     0.016460    0.011973

      protein    AG_fat     AG_pH  AG_protein        AG
0    0.002321  0.000756  0.001802    0.002300  0.004858
1    0.015098  0.055475  0.039215    0.043486  0.138176
2    0.013710  0.051087  0.034084    0.039221  0.124392
3    0.001352  0.001223  0.004995    0.001149  0.007367
4    0.013318  0.051915  0.037243    0.039013  0.128172
..        ...       ...       ...         ...       ...
260  0.015024  0.052155  0.032527    0.041637  0.126319
261  0.015089  0.057727  0.041268    0.044968  0.143963
262  0.001841  0.000725  0.002916    0.001992  0.005633
263  0.014931  0.055232  0.039223    0.043227  0.137683
264  0.013582  0.055177  0.042091    0.042014  0.139282

[265 rows x 12 columns]
```

As seen above, Mendelian sampling (co-)variance will be estimated for all individuals if parameter sub_id is None. There may be cases where a user might be interested in a subset of individuals. In this case, the user can provide individuals of interest as a pandas data frame with one column. Assuming the candidates of interest are

In [13]:
```
aaa = np.arange(0, 265, 40)
df_gam = data.group.iloc[aaa, 1]
print(df_gam)
```

```
0      10001
40     10041
80     10081
120    10121
160    10161
200    10201
240    10241
Name: ID, dtype: int64
```

Then the Mendelian sampling (co-)variances of traits for these individuals are given below:

In [14]:
```
start = time.time()
msvmsc_gsub = msq.msvarcov_g(info=data, covmat=covmatrices,
                             sub_id=df_gam, progress=False)
print('Time taken: ', round(time.time() - start, 2), 'secs')
print(msvmsc_gsub)
```

```
Time taken:  0.66 secs
      ID Group       fat    pH_fat        pH  protein_fat  protein_pH  \
0  10001     M  0.000466  0.000154  0.001805     0.000136   -0.000157
1  10041     M  0.000421  0.000161  0.003291     0.000095   -0.000169
2  10081     M  0.000629  0.000242  0.003329     0.000291   -0.000025
3  10121     M  0.022551  0.014175  0.011532     0.016422    0.010774
4  10161     F  0.000469 -0.000055  0.002331     0.000368   -0.000489
5  10201     F  0.000466 -0.000022  0.002190     0.000254   -0.000319
6  10241     F  0.000566  0.000213  0.001609     0.000415    0.000152

    protein    AG_fat     AG_pH  AG_protein        AG
0  0.002321  0.000756  0.001802    0.002300  0.004858
1  0.001163  0.000676  0.003282    0.001089  0.005048
2  0.003116  0.001162  0.003546    0.003382  0.008090
3  0.014838  0.053148  0.036481    0.042034  0.131663
4  0.002253  0.000782  0.001786    0.002132  0.004701
5  0.002331  0.000698  0.001850    0.002266  0.004815
6  0.002229  0.001194  0.001974    0.002796  0.005965
```

Mendelian correlations

Mendelian sampling covariance can be converted to correlations using the function `msq.msvarcov_gcorr` to better compare trait combinations. The function returns a data frame containing correlations between traits. Note that the output does not contain a trait's correlation with itself (equals one).

In [15]:
```
start = time.time()
msvmsc_gsubcorr = msq.msvarcov_gcorr(msvmsc_gsub)
print('Time taken: ', round(time.time() - start, 2), 'secs')
print(msvmsc_gsubcorr)
```

```
Time taken:  0.01 secs
      ID Group    pH_fat  protein_fat  protein_pH    AG_fat     AG_pH  \
0  10001     M  0.168243     0.130355   -0.076619  0.502305  0.608696
1  10041     M  0.136506     0.135905   -0.086558  0.464180  0.805301
2  10081     M  0.167167     0.208082   -0.007698  0.515131  0.683314
3  10121     M  0.879009     0.897727    0.823645  0.975373  0.936237
4  10161     F -0.052918     0.358214   -0.213430  0.526718  0.539680
5  10201     F -0.021511     0.244284   -0.141048  0.466434  0.569633
6  10241     F  0.223733     0.369907    0.080044  0.650271  0.637240

   AG_protein
0    0.684896
1    0.449414
2    0.673654
3    0.950996
4    0.655163
5    0.676601
6    0.766810
```

Estimation of selection criteria

1. `Gametic approach` : The function `msq.selstrat_g` can be used to obtain selection criteria for each characteristic as well as an individual's aggregate breeding value (ABV). The parameter selstrat can be "GEBV," "PBTI," or "INDEX," repre- senting genomic estimated breeding values, probability to breed top-ranking individuals, or a selection index combining GEBV and Mendelian standard deviations, respectively. If selstrat is "PBTI" or "INDEX", parameter sub_id is ignored and estimates are provided for individuals in Mendelian sampling (co-)variance (msvmsc parameter) data frame.

In [16]:
```
start = time.time()
sel_g = msq.selstrat_g(selstrat="PBTI", info=data, sub_id=df_gam,
                       msvmsc=msvmsc_g, throrconst=0.05)
print('Time taken: ', round(time.time() - start, 2), 'secs')
print(sel_g)
```

```
Time taken:  0.49 secs
        ID Group       fat        pH   protein       ABV
0    10001     M  0.996264  0.000000  0.651731  0.033476
1    10002     M  0.084399  0.000003  0.134932  0.047714
2    10003     M  0.052664  0.006790  0.128033  0.141537
3    10004     M  0.999113  0.000138  0.000003  0.183958
4    10005     M  0.192267  0.000004  0.000017  0.007792
..     ...   ...       ...       ...       ...       ...
260  10261     F  0.002609  0.001245  0.000039  0.003536
261  10262     F  0.003807  0.019924  0.000075  0.011257
262  10263     F  0.623545  0.998058  0.015548  0.999982
263  10264     F  0.002814  0.004171  0.000219  0.007632
264  10265     F  0.000879  0.099573  0.000032  0.012122

[265 rows x 6 columns]
```

If selstrat is GEBV, estimates are provided for individuals in the sub_id data frame. Other parameters may be set to None because they are not required for GEBV. For example, a user may set the sub_id parameter to None if interested in all individuals.

In [17]:
```
start = time.time()
sel_g = msq.selstrat_g(selstrat="GEBV", info=data, sub_id=None,
                       msvmsc=None, throrconst=None)
print('Time taken: ', round(time.time() - start, 2), 'secs')
print(sel_g)
```

```
Time taken:  0.31 secs
        ID Group       fat        pH   protein       ABV
0    10001     M  0.270993 -0.214728  0.291675  0.347941
1    10002     M  0.003386 -0.284890  0.137310 -0.144194
2    10003     M -0.025118 -0.017753  0.139902  0.097031
3    10004     M  0.309569 -0.018075  0.106850  0.398344
4    10005     M  0.084600 -0.269435 -0.205395 -0.390230
..     ...   ...       ...       ...       ...       ...
260  10261     F -0.202100 -0.068357 -0.211195 -0.481652
261  10262     F -0.198903  0.001456 -0.192614 -0.390060
262  10263     F  0.220997  0.384744  0.180376  0.786116
263  10264     F -0.204883 -0.062984 -0.156713 -0.424580
264  10265     F -0.255890  0.083525 -0.192927 -0.365292

[265 rows x 6 columns]
```

If parameter selstrat="INDEX", a constant of throrconst = 1.5 is used to weight Mendelian sampling standard deviations

In [18]:
```
start = time.time()
sel_gsub = msq.selstrat_g(selstrat="INDEX", info=data, sub_id=None,
                          msvmsc=msvmsc_gsub, throrconst=1.5)
print('Time taken: ', round(time.time() - start, 2), 'secs')
print(sel_gsub)
```

```
Time taken:  0.02 secs
      ID Group       fat        pH   protein       ABV
0  10001     M  0.167868 -0.043638  0.218103  0.278515
1  10041     M  0.144649  0.132887 -0.023245  0.192882
2  10081     M  0.136536  0.168263  0.244436  0.476266
3  10121     M  0.149041  0.149686  0.147139  0.421094
4  10161     F -0.246005 -0.108670  0.080309 -0.347634
5  10201     F -0.235311 -0.055924 -0.015538 -0.377672
6  10241     F  0.033224  0.193590  0.103777  0.279773
```

1. `Zygotic approach` : Selection criteria for zygotes can be obtained using the function `msq.selstrat_z` . Parameter sub_idz plays a slightly different role. Here, if sub_idz is None, the best (based on chosen selection criterion) combinations of top males and females in the Mendelian sampling (co-)variance data frame (msvmsc parameter) are outputted. The maximum allowed allocation of males can also be set. However, if sub_idz is a data frame with IDs of parent pairs, the parameters selmale (selection proportion for males; 0.01 selects 1%, 1 selects all males), selfm (selection proportion for females), and maxmale (maximum number of allocation allowed per male) can be set to None.

In [19]:
```
start = time.time()
sel_z = msq.selstrat_z(selstrat="PBTI", info=data, sub_idz=None,
                       msvmsc=msvmsc_g, throrconst=0.05,
                       selmale=["M", 1], selfm=["F", 1],
                       maxmale=2)
print('Time taken: ', round(time.time() - start, 2), 'secs')
print(sel_z)
```

```
Time taken:  1.22 secs
    MaleID FemaleID MaleIndex FemaleIndex       fat        pH   protein  \
0    10009    10131         8         130       0.0  0.015273  0.196209
1    10009    10132         8         131       0.0  0.000266  0.009127
2    10046    10133        45         132   0.33206  0.321037  0.168052
3    10046    10134        45         133  0.252743  0.251529  0.151957
4    10031    10135        30         134       0.0  0.000101  0.000019
..     ...      ...       ...         ...       ...       ...       ...
130  10017    10261        16         260  0.116728  0.002163  0.008054
131  10017    10262        16         261  0.127544  0.011316  0.009853
132  10085    10263        84         262  0.136705  0.302792  0.020996
133  10085    10264        84         263  0.037648   0.02904  0.006313
134  10061    10265        60         264   0.02589  0.025851  0.020836

          ABV
0    0.002325
1    0.000001
2    0.516713
3     0.42109
4         0.0
..        ...
130  0.060115
131  0.089903
132  0.254442
133  0.049238
134  0.055669

[135 rows x 8 columns]
```

Same results can be obtained by providing a data frame sub_idz with male and female IDs in the first and second columns, provided these individuals are in Mendelian sampling (co-)variance data frame (msvmsc_g).

In [20]:
```
df_zyg = sel_z.iloc[0:5, 0:2]        # dataframe containing a subset of mate pairs
start = time.time()
sel_z = msq.selstrat_z(selstrat="PBTI", info=data, sub_idz=df_zyg,
                       msvmsc=msvmsc_g, throrconst=0.05,
                       selmale=None, selfm=None, maxmale=None)
print('Time taken: ', round(time.time() - start, 2), 'secs')
print(sel_z)
```

```
Time taken:  0.34 secs
  MaleID FemaleID  MaleIndex  FemaleIndex       fat        pH   protein  \
0  10009    10131          8          130  0.000000  0.015273  0.196209
1  10009    10132          8          131  0.000000  0.000266  0.009127
2  10046    10133         45          132  0.332060  0.321037  0.168052
3  10046    10134         45          133  0.252743  0.251529  0.151957
4  10031    10135         30          134  0.000000  0.000101  0.000019

            ABV
0  2.324804e-03
1  1.146761e-06
2  5.167133e-01
3  4.210896e-01
4  9.598058e-10
```

Derivation of similarity matrices based on Mendelian sampling values

1. `Gametic approach` : This can be done using the function `msq.simmat_g` . The function returns a named list containing similarity or standardized similarity matrix based on aggregate genotypes for each group. If parameter chrinterest is a string of 'all' or 'none,' all or none of the chromosome-wise similarity matrices are saved to file. If chrinterest is a list (e.g., [4, 14]), similarity matrices for only chromosomes 4 and 14 will be saved to file. Trait-specific (standardized or) similarity matrices will be saved to file when save=True. If stdsim=True, a standardized similarity matrix is outputted. If progress = True, the progress of the calculations is printed to screen.

In [21]:
```
start = time.time()
sim_g = msq.simmat_g(info=data, covmat=covmatrices, sub_id=df_gam,
                     chrinterest="none", save=False, stdsim=False,
                     progress=True)
print('Time taken: ', round(time.time() - start, 2), 'secs')
sim_g
```

```
Processing group  M
Progress: |██████████████████████████████████████████████████| 100% Complete
Creating similarity matrix based on aggregate genotype
Progress: |██████████████████████████████████████████████████| 100% Complete
Processing group  F
Progress: |██████████████████████████████████████████████████| 100% Complete
Creating similarity matrix based on aggregate genotype
Progress: |██████████████████████████████████████████████████| 100% Complete
Time taken:  6.85 secs
```

Out[21]:
```
{'M': [array([[0.00485757, 0.00267605, 0.00388947, 0.00525877],
         [0.00267605, 0.00504775, 0.00347402, 0.00408954],
         [0.00388947, 0.00347402, 0.00809001, 0.0053334 ],
         [0.00525877, 0.00408954, 0.0053334 , 0.13166252]])],
 'F': [array([[0.00470106, 0.00280465, 0.00311769],
         [0.00280465, 0.00481454, 0.0030983 ],
         [0.00311769, 0.0030983 , 0.00596466]])]}
```

When group-specific maps are used, as seen in the example above, a named list of group-specific similarity matrices is returned (e.g., M and F). Because the individuals are organized into groups, the order of the individuals in each group's similarity matrix is saved in a (csv) file. However, if an average genetic map is used, a similarity matrix for all groups will be returned. In this case, individual orders are not saved to file because they remain unmodified.

1. `Zygotic approach` : Similarity matrices between Mendelian sampling values of zygotes produced by mate pairs can be derived using the function `msq.simmat_z` . A data frame with IDs of parent pairs must be provided to use this function. For example, if the selection criterion is estimated using the function `msq.selstrat_z` , the IDs in the first and second columns of the data frame may be provided as sub_idz parameter.

In [22]:
```
start = time.time()
sim_z = msq.simmat_z(info=data, sub_idz=df_zyg, covmat=covmatrices,
                     chrinterest="none", stdsim=True, save=False,
                     progress=False)
print('Time taken: ', round(time.time() - start, 2), 'secs')
sim_z
```

```
Time taken:  6.52 secs
```

Out[22]:
```
array([[1.        , 0.76503754, 0.2302074 , 0.19034405, 0.48636912],
       [0.76503754, 1.        , 0.31258909, 0.30383955, 0.60487113],
       [0.2302074 , 0.31258909, 1.        , 0.97591852, 0.30035285],
       [0.19034405, 0.30383955, 0.97591852, 1.        , 0.30850437],
       [0.48636912, 0.60487113, 0.30035285, 0.30850437, 1.        ]])
```

References

1. Musa AA, Reinsch N. A similarity matrix for hedging haplotype diversity among parents in genomic selection. Submitted. 2021.
2. Melzer N, Wittenburg D, Repsilber D. Integrating Milk Metabolite Profile Information for the Prediction of Traditional Milk Traits Based on SNP Information for Holstein Cows. PLoS One. 2013;8:e70256.
3. Hampel A, Teuscher F, Gomez-Raya L, Doschoris M, Wittenburg D. Estimation of Recombination Rate and Maternal Linkage Disequilibrium in Half-Sibs. Front Genet. 2018;9 JUN:186.
4. Bonk S, Reichelt M, Teuscher F, Segelke D, Reinsch N. Mendelian sampling covariability of marker effects and genetic values. Genet Sel Evol. 2016;48:1–11. doi:10.1186/s12711-016-0214-0.
5. Santos DJA, Cole JB, Lawlor TJ, VanRaden PM, Tonhati H, Ma L. Variance of gametic diversity and its application in selection programs. J Dairy Sci. 2019;102:5279–94.