example

July 25, 2024

This is a tutorial for multi-ancestry fine-mapping using MultiSuSiE. You should be able to run this jupyter notebook interactively from the MultiSuSiE/examples directory or click on the .ipynb file on github to view it statically.

In this tutorial, we'll do the following: - Simulate a quantitative phenotype using real HapMap3 genotypes for a very small region on chromosome 1 including three populations with distinct continental genetic ancestries, YRI (Yoruba in Ibada, Nigeria), CEU (Utah residents with Northern and Western European ancestry), and JPT (Japanese in Tokyo, Japan). - Generate summary statistics using the simulated quantitative phenotype and real genotypes. - Fine-map our example locus using the summary statistics and individual level data.

1 Load packages

[1]:
```
import numpy as np
import MultiSuSiE
from IPython.display import Markdown as md
```

2 Load data and simulate a quantitative phenotype

For this tutorial, we'll use a small piece of chromosome 1 from 3 HapMap3 populations, YRI, CEU, and JPT. We can load the example data using the following code chunk.

[2]:
```
geno_YRI = np.loadtxt('../example_data/geno_YRI.txt')
geno_CEU = np.loadtxt('../example_data/geno_CEU.txt')
geno_JPT = np.loadtxt('../example_data/geno_JPT.txt')
geno_list = [geno_YRI, geno_CEU, geno_JPT]
```

We're going to pretend that we have three causal variants. The first variant has varying effect sizes across ancestries. The second has identical effect sizes across ancestries. The third only has an effect in YRI.

[3]:
```
beta_YRI = np.zeros(40)
beta_CEU = np.zeros(40)
beta_JPT = np.zeros(40)
beta_YRI[10]=.75
beta_CEU[10]=1
beta_JPT[10]=.5
beta_YRI[3]=.5
```

```
beta_CEU[3]=.5
beta_JPT[3]=.5
beta_YRI[38]=1
beta_CEU[38]=0
beta_JPT[38]=0
beta_list = [beta_YRI, beta_CEU, beta_JPT]
```

3 Calculate summary statistics

For this tutorial, we'll calculate summary statistics in Python, but you'll likely calculate your association summary statistics using Plink2 and calculate LD matrices using LDStore2.

[4]:
```
rng = np.random.default_rng(2)
y_list = [geno.dot(beta) + rng.standard_normal(geno.shape[0]) for (geno, beta)␣
 ↪in zip(geno_list, beta_list)]
y_list = [y - np.mean(y) for y in y_list]
```

[5]:
```
XTY_list = [geno.T.dot(y) for (geno, y) in zip(geno_list, y_list)]
XTX_diagonal_list = [np.diagonal(geno.T.dot(geno)) for geno in geno_list]
with np.errstate(divide='ignore',invalid='ignore'): # just to silence a divide␣
 ↪by zero error
    beta_hat_list = [XTY / XTX_diag for (XTY, XTX_diag) in zip(XTY_list,␣
 ↪XTX_diagonal_list)]

N_list = [geno.shape[0] for geno in geno_list]
residuals_list = [np.expand_dims(y, 1) - (geno * beta) for (y, geno, beta) in␣
 ↪zip(y_list, geno_list, beta_hat_list)]
sum_of_squared_residuals_list = [np.sum(resid ** 2, axis = 0) for resid in␣
 ↪residuals_list]
se_list = [np.sqrt(ssr / ((N - 2) * XTX)) for (ssr, N, XTX) in␣
 ↪zip(sum_of_squared_residuals_list, N_list, XTX_diagonal_list)]

with np.errstate(divide='ignore',invalid='ignore'): # just to silence a divide␣
 ↪by zero error
    R_list = [np.corrcoef(geno, rowvar = False) for geno in geno_list]

YTY_list = [y.dot(y) for y in y_list]
varY_list = [np.var(y, ddof = 1) for y in y_list]

z_list = [b/s for (b,s) in zip(beta_hat_list, se_list)]
```

4 Summary statistic based fine-mapping

To run summary-statistic based multi-ancestry fine-mapping, there are two potential sets of inputs for each population: 1. GWAS variant effect sizes, GWAS variant standard errors, an LD matrix, the sample phenotype variance, and the GWAS sample size 2. GWAS Z-scores, an LD matrix, and the GWAS sample size

The first input set is preferrable because it allows us to keep genotypes and phenotypes on their original scale, allows automatic filtering of low minor allele count variants, and is validated in the MultiSuSiE manuscript. If you choose to use the second input set, it's very important to censor low minor allele count variants in the population that they have low minor allele count in. You can either zero out the corresponding entries in that population's Z-score array, and the corresponding rows and columns in that populations LD matrix, or provide `mac_list` or `maf_list` and let `multisusie_rss` do it for you. We'll demonstrate this below.

In either case, the inputs should be formatted as lists of numpy arrays (for the GWAS variant effect sizes, GWAS variant standard errors, GWAS Z-scores and LD matrix) or lists of scalars (for the sample phenotype variance and GWAS sample size). Each list should have length equal to the number of populations. The summary statistics we calculated above are already in this format.

To run MultiSuSiE using the first input set, we can do the following:

[6]:
```
ss_fit = MultiSuSiE.multisusie_rss(
    b_list = beta_hat_list,
    s_list = se_list,
    R_list = R_list,
    varY_list = varY_list,
    rho = np.array([[1, 0.75, 0.75], [0.75, 1, 0.75], [0.75, 0.75, 1]]),
    population_sizes = N_list,
    L = 10,
    scaled_prior_variance = 0.2,
    low_memory_mode = False,
    float_type = np.float64,
    single_population_mac_thresh = 10 # set this to 20 unless your sample size␣
 ↪is tiny, like in this example
)
```

```
Censored 14 variants in population 0 due to low population-specific MAC
Censored 15 variants in population 1 due to low population-specific MAC
Censored 16 variants in population 2 due to low population-specific MAC
```

[7]:
```
print(f'The PIPs of the causal variants are {ss_fit.pip[3]}, {ss_fit.pip[10]},␣
 ↪and {ss_fit.pip[38]}')
```

```
The PIPs of the causal variants are 0.4655837427342536, 0.9999781922032149, and
0.969634716536287
```

By examining the `pip` attribute of the object returned by `MultiSuSiE`, we can see that we've correctly assigned very high PIP to the 11th and 39th variants, and moderate PIP to the 4th variant. These variants have true causal effects in our simulations. By examining the `sets` attribute, we can see that the causal variant with lower PIP has been placed in a 95% credible set with only two other variants.

[8]:
```
ss_fit.sets[0]
```

```
[8]: [array([10]), array([ 3,  7, 11]), array([38]), [], [], [], [], [], [], []]
```

If we'd rather use the second input option (Z-scores, LD, and sample size) than the first, we have to remember to censor low minor allele count variants, or provide multisusie_rss with `maf_list` or `mac_list` so it can do the censoring. Typically a minor allele count threshold of 20 is used for this, but due to the very low sample size in this example we'll use 10.

```
[9]: maf_YRI = np.loadtxt('../example_data/maf_YRI.txt')
maf_CEU = np.loadtxt('../example_data/maf_CEU.txt')
maf_JPT = np.loadtxt('../example_data/maf_JPT.txt')
maf_list = [maf_YRI, maf_CEU, maf_JPT]
ss_fit = MultiSuSiE.multisusie_rss(
    z_list = z_list,
    R_list = R_list,
    rho = np.array([[1, 0.75, 0.75], [0.75, 1, 0.75], [0.75, 0.75, 1]]),
    population_sizes = N_list,
    L = 10,
    scaled_prior_variance = 0.2,
    low_memory_mode = False,
    single_population_mac_thresh = 10,
    maf_list = maf_list
)
```

```
Censored 14 variants in population 0 due to low population-specific MAC
Censored 15 variants in population 1 due to low population-specific MAC
Censored 16 variants in population 2 due to low population-specific MAC
```

```
[10]: print(f'The PIPs of the causal variants are {ss_fit.pip[3]}, {ss_fit.pip[10]},␣
  ↪and {ss_fit.pip[38]}')
```

```
The PIPs of the causal variants are 0.46658849716186523, 0.9999762177467346, and
0.9698582291603088
```

5 Summary statistics vs individual level fine-mapping

Next, we'll demonstrate that the individual and summary statistic based versions of MultiSuSiE give identical results. At the time when this tutorial was being written, some of the default parameters for `multisusie_rss` (the top-level summary-statistic based fine-mapping function) had not been implemented for `multisusie` (the top-level individual-level based fine-mapping function), so we'll have to use some non-default parameters to get the same results for both functions.

```
[11]: ss_fit = MultiSuSiE.multisusie_rss(
    b_list = beta_hat_list,
    s_list = se_list,
    R_list = R_list,
    varY_list = varY_list,
    rho = np.array([[1, 0.75, 0.75], [0.75, 1, 0.75], [0.75, 0.75, 1]]),
    population_sizes = N_list,
    L = 10,
    low_memory_mode = False,
    recover_R = False,
    float_type = np.float64,
    single_population_mac_thresh = 0,
)
indiv_fit = MultiSuSiE.multisusie(
    X_list = geno_list,
    Y_list = y_list,
    rho = np.array([[1, 0.75, 0.75], [0.75, 1, 0.75], [0.75, 0.75, 1]]),
    L = 10,
    standardize = False,
    intercept = False,
    float_type = np.float64
)
```

[12]:
```
md("Here, we can see that the maximum difference in PIP between the two methods␣
 ↪is  {}".format(np.max(np.abs(ss_fit.pip - indiv_fit.pip))))
```

[12]: Here, we can see that the maximum difference in PIP between the two methods is 1.3322676295501878e-15

6 Z-score vs individual level fine-mapping

If you choose to do z-score based fine-mapping, this is equivalent to doing individual level fine-mapping after standardizing the genotype and phenotype matrices. We can see this below. Here, we're not censoring low MAC variants which is not recommended.

[13]:
```
with np.errstate(divide='ignore',invalid='ignore'):
    geno_std_list = [geno / np.std(geno, axis = 0, ddof = 1) for geno in␣
 ↪geno_list]
y_std_list = [y / np.std(y, ddof = 1) for y in y_list]
XTX_std_list = [geno.T.dot(geno) for geno in geno_std_list]
XTY_std_list = [geno.T.dot(pheno) for (geno, pheno) in zip(geno_std_list,␣
 ↪y_std_list)]
```

[14]:
```
ss_fit = MultiSuSiE.multisusie_rss(
    z_list = z_list,
    R_list = R_list,
    varY_list = None,
    rho = np.array([[1, 0.75, 0.75], [0.75, 1, 0.75], [0.75, 0.75, 1]]),
    population_sizes = N_list,
    L = 10,
    low_memory_mode = False,
    recover_R = False,
    float_type = np.float64,
    single_population_mac_thresh = 0,
)
indiv_fit = MultiSuSiE.multisusie(
    X_list = [np.nan_to_num(geno, 0) for geno in geno_std_list],
    Y_list = y_std_list,
    rho = np.array([[1, 0.75, 0.75], [0.75, 1, 0.75], [0.75, 0.75, 1]]),
    L = 10,
    standardize = False,
    intercept = False,
    float_type = np.float64
)
```

[15]:
```
md("Here, we can see that the maximum difference in PIP between the two methods␣
 ↪is  {}".format(np.max(np.abs(ss_fit.pip - indiv_fit.pip))))
```

[15]:

Here, we can see that the maximum difference in PIP between the two methods is 3.6637359812630166e-15