example

May 14, 2024

This is a tutorial for multi-ancestry fine-mapping using MultiSuSiE. For now, this tutorial focuses on summary statistic-based fine-mapping. In the future, this tutorial will be expanded to include individual-level fine-mapping, which is also supported by the MultiSuSiE software package. You should be able to run this jupyter notebook from the MultiSuSiE/examples directory or just following along with the pdf.

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

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.

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.

[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_CEU, geno_YRI, geno_JPT]
```

[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're going to calculate our 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(1)
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]
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)]

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]
```

```
/var/folders/77/mk7r4vgs7w1_m2d24r01lhd40000gn/T/ipykernel_26877/4224691857.py:3
: RuntimeWarning: invalid value encountered in divide
  beta_hat_list = [XTY / XTX_diag for (XTY, XTX_diag) in zip(XTY_list,
XTX_diagonal_list)]
/Users/jor6523/miniconda3/envs/MultiSuSiE/lib/python3.12/site-
packages/numpy/lib/function_base.py:2897: RuntimeWarning: invalid value
encountered in divide
  c /= stddev[:, None]
/Users/jor6523/miniconda3/envs/MultiSuSiE/lib/python3.12/site-
packages/numpy/lib/function_base.py:2898: RuntimeWarning: invalid value
```

```
encountered in divide
  c /= stddev[None, :]
```

4 Demonstrate summary statistic based fine-mapping

To run summary-statistic based multi-ancestry fine-mapping, we need five inputs for each population: GWAS variant effect sizes, GWAS variant standard errors, an LD matrix, the sample phenotype variance, and the GWAS sample size. Each of these inputs should be formatted as a list of numpy arrays (for the GWAS variant effect sizes, GWAS variant standard errors, and LD matrix) or a list of scalars(for the sample phenotype variance and GWAS sample size). The summary statistics we calculated above are already in this format.

Now, to run MultiSuSiE, we just have to 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,
    mac_filter = 10,
    maf_filter = 0
)
```

```
censored 15 variants in population 0
censored 14 variants in population 1
censored 16 variants in population 2
```

Here, we've decreased the threshold for the MAC_filter argument from the default value due to our extremely small sample sizes. 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 38th variants, and moderate PIP to the 4th variant, all of which we've assigned 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 other variant.

5 Show equivalence of individual and ss based methods

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) have not been implemented for `susie_multi` (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.

[7]:
```
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,
    min_abs_corr = 0,
    recover_R = False,
    float_type = np.float64,
    estimate_prior_method = 'EM',
    pop_spec_effect_priors = False,
    iter_before_zeroing_effects = 0,
    mac_filter = 0,
    maf_filter = 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,
    scaled_prior_variance = 0.2,
    standardize = False,
    intercept = False,
    float_dtype = np.float64
)
```

[8]:
```
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))))
```

[8]: Here, we can see that the maximum difference in PIP between the two methods is 2.7755575615628914e-15