Supporting Information for *Linear mixed model for heritability estimation that explicitly addresses environmental variation*

> Version 0.25
>
> Microsoft Research, April 19, 2016

This supplement is an IPython notebook that generates synthetic data and illustrates the application of our code to that data.

This notebook allows you to run these analyses on your computer, easily experimenting with different data-generation parameters. If you would like to do this, you need to:

- Install Python and FaST-LMM on your computer: https://github.com/MicrosoftGenomics/FaST-LMM/blob/master/README.md
- Download this IPython notebook from https://github.com/MicrosoftGenomics/FaST-LMM/blob/master/doc/ipynb/heritability_si.ipynb
- Start the IPython notebook from the command line by typing "ipython notebook" and then opening the downloaded notebook. For details see http://jupyter-notebook-beginner-guide.readthedocs.org/en/latest/execute.html

Preparing the Python Environment and Notebook

Run this cell only if these packages need to be install or updated.

In [ ]:

```
#Main package
!pip install fastlmm">="0.2.24

#Needed by this notebook to generate synthetic SNPs
!pip install GWAS_benchmark">="0.1.3
```

To prepare this notebook to run analyses, please run the following script.

In [1]:

```
# set some ipython notebook properties
%matplotlib inline

# set degree of verbosity (adapt to INFO for more verbose output or debug for info more)
import logging
logging.basicConfig(level=logging.INFO)

# set figure sizes
import pylab
pylab.rcParams['figure.figsize'] = (10.0, 8.0)

# set display width for pandas data frames
import pandas as pd
pd.set_option('display.width', 1000)

# supress warnings
import warnings
warnings.filterwarnings('ignore')
```

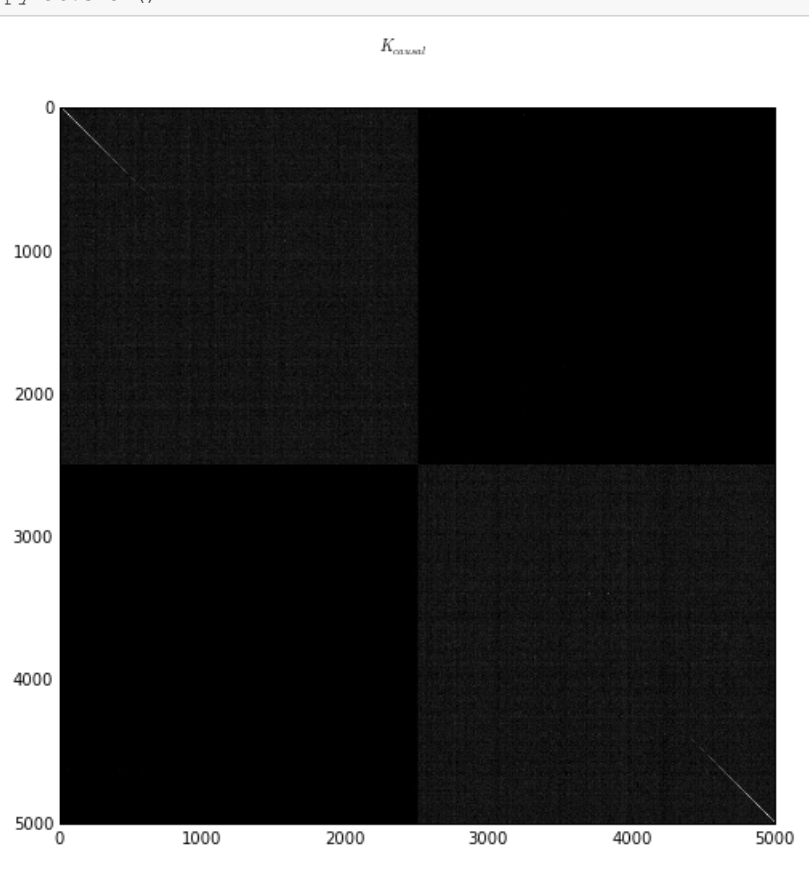
Data Generation

To generate the data, we create SNPs and geo-spatial locations. We then generate the phenotype from these SNPs and geo-spatial locations. Note that, because the data is synthetic, we know the causal SNPs. Consequently, we use K_{causal} rather than its approximation, K_{IBD}, in this analysis.

First, we generate 1000 SNPs per individual for 5000 individuals using the Balding-Nichols model for two populations with an FST of 0.1. The heatmap shows the resulting similarity matrix, K_{causal}, highlighting the existence of two populations.

In [2]:

```
from GWAS_benchmark.snp_gen import snp_gen
from pysnptools.standardizer import Unit

seed = 0
N = 5000

#Generate SNPs
snpdata = snp_gen(fst=.1, dfr=0, iid_count=N, sid_count=1000, chr_count=10, label_with_pop=True,seed=seed)
K_causal = snpdata.read_kernel(Unit()).standardize()

pylab.suptitle("$K_{causal}$")
pylab.imshow(K_causal.val, cmap=pylab.gray(), vmin=0, vmax=1)
pylab.show()
```

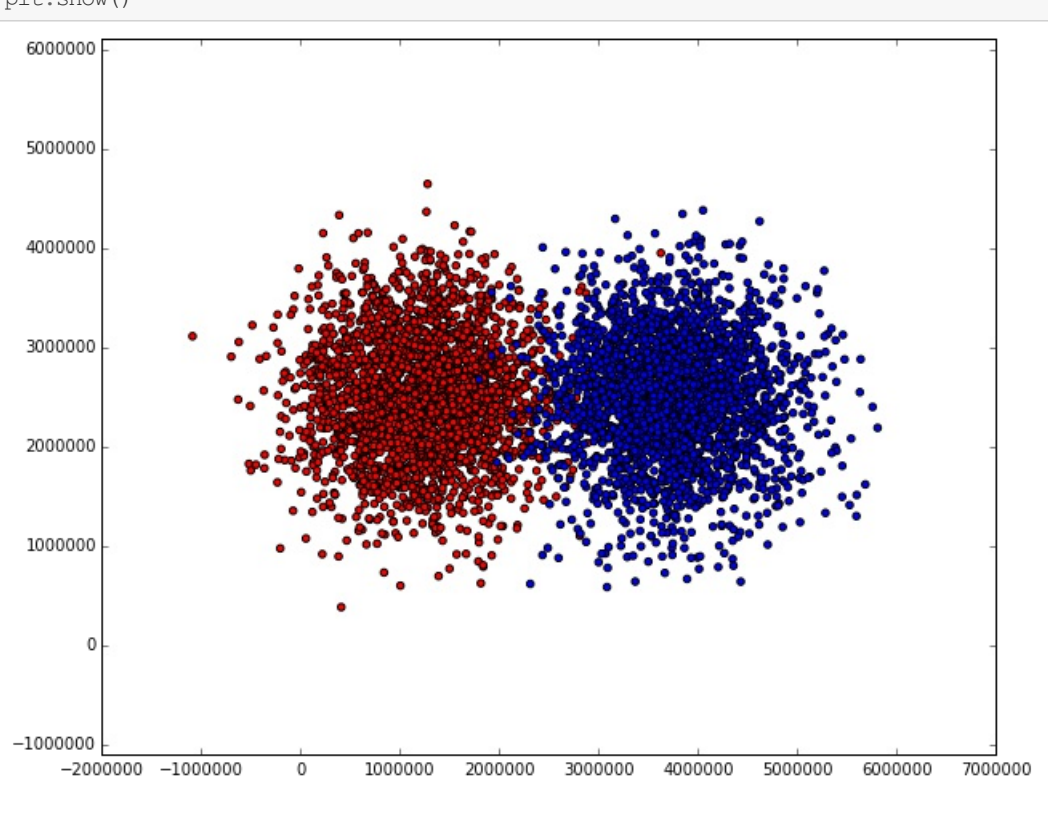


Next, we generate a geo-spatial location for each individual by sampling from a two-dimensional Gaussian distribution whose center depends on which population the individual is from. This creates a correlation between the SNPs and geo-spatial location.

The plot shows the position of each individual. The two colors reflect the two populations.

In [3]:

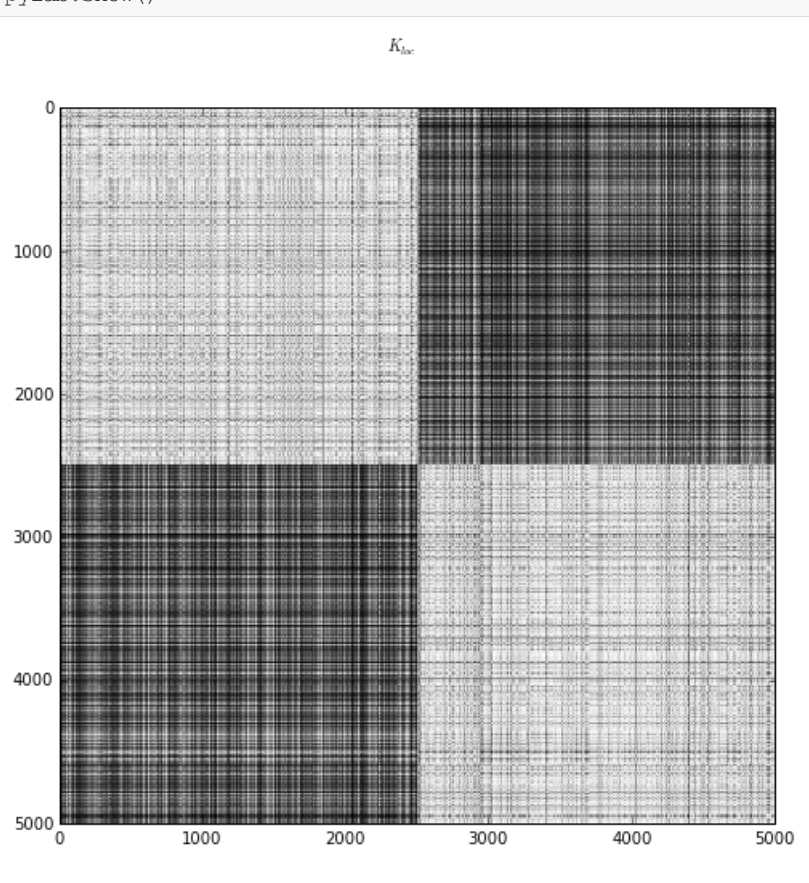
```
import numpy as np
from pysnptools.snpreader import SnpData

distance_between_centers = 2500000
x0 = distance_between_centers * 0.5
x1 = distance_between_centers * 1.5
y0 = distance_between_centers
y1 = distance_between_centers
sd = distance_between_centers/4.

spatial_iid = snpdata.iid
center_dict = {"0": (x0,y0), "1": (x1,y1)}
centers = np.array([center_dict[iid_item[0]] for iid_item in spatial_iid])
np.random.seed(seed)
logging.info("Generating positions for seed {0}".format(seed))
spatial_coor_gen_original = SnpData(iid=snpdata.iid,sid=["x","y"],val=centers+np.random.multivariate_norma
l([0,0],[[1,0],[0,1]],size=len(centers))*sd,parent_string="'spatial_coor_gen_original'")

import matplotlib.pyplot as plt
color_dict = {"0": "r", "1": "b", "2": "g"}
colors = [color_dict[iid_item] for iid_item in snpdata.iid[:,0]]
plt.axis('equal')
plt.scatter(spatial_coor_gen_original.val[:,0], spatial_coor_gen_original.val[:,1], c=colors)
plt.show()
```



Below is the the similarity matrix K_{loc} based on these locations and parameter alpha. $K_{loc\ ij} = \exp(-(\mathrm{distance}(i,j)/\alpha)^2)$.

In [4]:

```
from fastlmm.association.heritability_spatial_correction import spatial_similarity
from pysnptools.kernelreader import KernelData

alpha = distance_between_centers
spatial_val = spatial_similarity(spatial_coor_gen_original.val, alpha, power=2)
K_loc = KernelData(iid=snpdata.iid,val=spatial_val).standardize()

pylab.suptitle("$K_{loc}$")
pylab.imshow(K_loc.val, cmap=pylab.gray(), vmin=0, vmax=1)
pylab.show()
```

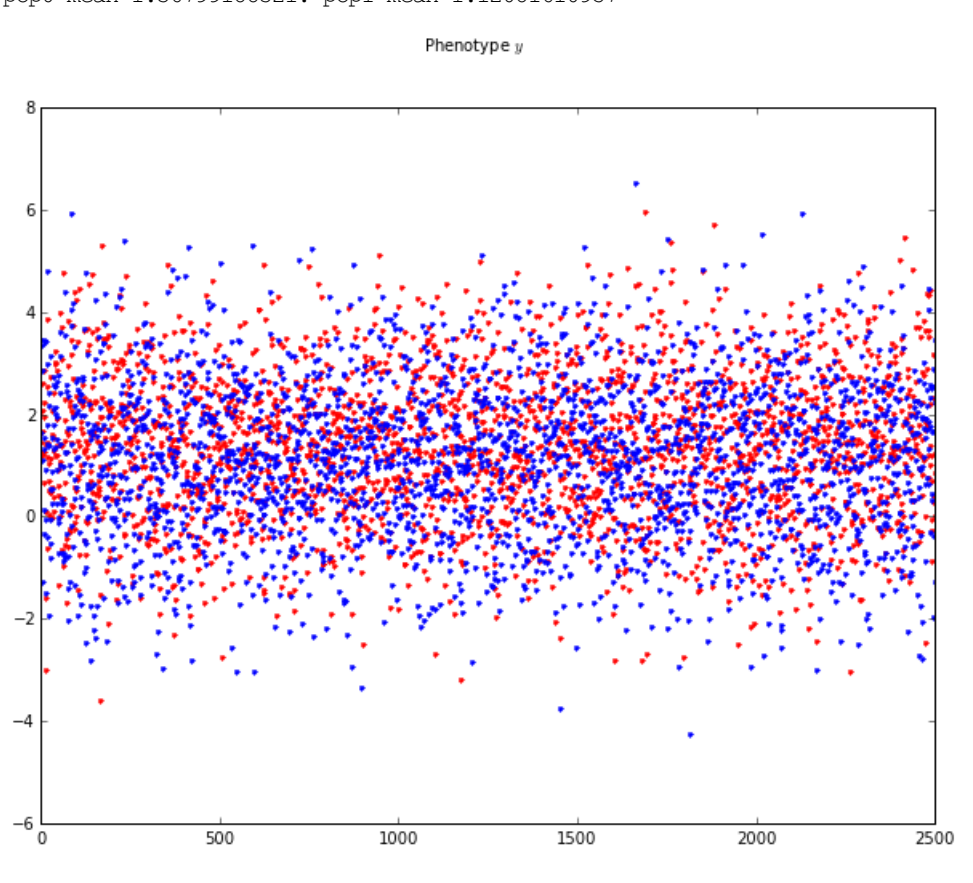


Finally, we generate the phenotype y by sampling from the mixed model $y \sim N(\mu; \sigma_g^2 K_{causal} + \sigma_e^2 K_{loc} + \sigma_r^2 I)$ with $\mu = 0$ and $\sigma_g^2 = \sigma_e^2 = \sigma_r^2 = 1$. Rather than draw a sample from this distribution directly, we equivalently draw a sample from each variant component separately and then add them, so that we can optionally permute the sample from the spatial component in order to break the dependence between SNPs and location.

In [5]:

```
from pysnptools.snpreader import SnpData

iid = K_causal.iid
iid_count = K_causal.iid_count
np.random.seed(seed)
pheno_causal = SnpData(iid=iid,sid=["causal"],val=np.random.multivariate_normal(np.zeros(iid_count),K_caus
al.val).reshape(-1,1),parent_string="causal")
np.random.seed(seed^998372)
pheno_noise = SnpData(iid=iid,sid=["noise"],val=np.random.normal(size=iid_count).reshape(-1,1),parent_stri
ng="noise")
np.random.seed(seed^12230302)
pheno_loc_original = SnpData(iid=iid,sid=["loc_original"],val=np.random.multivariate_normal(np.zeros(iid_c
ount),K_loc.val).reshape(-1,1),parent_string="loc_original")

do_shuffle = False
if do_shuffle:
    idx = np.arange(iid_count)
    np.random.seed(seed)
    np.random.shuffle(idx)
    pheno_loc = pheno_loc_original.read(view_ok=True) #don't need to copy, because the next line will be f
resh memory
    pheno_loc.val = pheno_loc.val[idx,:]
else:
    pheno_loc = pheno_loc_original

pheno = SnpData(iid=iid,sid=["pheno_all"],val=pheno_causal.val+pheno_noise.val+pheno_loc.val)

is_pop0 = snpdata.iid[:,0]=="0"
is_pop1 = snpdata.iid[:,0]=="1"
print "pop0 mean={0}. pop1 mean={1}".format(pheno.val[is_pop0].mean(),pheno.val[is_pop1].mean())
pylab.suptitle("Phenotype $y$")
plt.plot(range(sum(is_pop0)),pheno.val[is_pop0],"r.",range(sum(is_pop1)),pheno.val[is_pop1],"b.")
plt.show()
```

```
pop0 mean=1.36799188321. pop1 mean=1.12081610957
```



Analysis

Now let us estimate heritabilities using our methodology. The method takes as input K_{causal}, geo-spatial location, and phenotype. It searches for the best α and then estimates $\sigma_g^2, \sigma_e^2, \sigma_r^2$ using REML. The Python function that performs the analysis is called 'heritability_spatial_correction'.

Here we loop over multiple random seeds in order to generate and analyze multiple data sets. Python function 'loop_generate_and_analyze' does the looping. It calls 'generate_and_analyze' which generates the data for each seed and calls 'heritability_spatial_correction' to analyze the data.

In [6]:

```
from GWAS_benchmark.snp_gen import snp_gen
from pysnptools.standardizer import Unit
import numpy as np
from fastlmm.association.heritability_spatial_correction import spatial_similarity
from pysnptools.kernelreader import KernelData
from pysnptools.snpreader import SnpData
from GWAS_benchmark.semisynth_simulations import generate_phenotype
from fastlmm.association import heritability_spatial_correction

def generate_and_analyze(seed, N, do_shuffle, just_testing=True, map_function=None, cache_folder=None):

    #Generate SNPs
    snpdata = snp_gen(fst=.1, dfr=0, iid_count=N, sid_count=1000, chr_count=10, label_with_pop=True,seed=s
eed)
    K_causal = snpdata.read_kernel(Unit()).standardize()

    #Generate geo-spatial locations and K_loc
    distance_between_centers = 2500000
    x0 = distance_between_centers * 0.5
    x1 = distance_between_centers * 1.5
    y0 = distance_between_centers
    y1 = distance_between_centers
    sd = distance_between_centers/4.

    spatial_iid = snpdata.iid
    center_dict = {"0": (x0,y0), "1": (x1,y1)}
    centers = np.array([center_dict[iid_item[0]] for iid_item in spatial_iid])
    np.random.seed(seed)
    logging.info("Generating positions for seed {0}".format(seed))
    spatial_coor = SnpData(iid=snpdata.iid,sid=["x","y"],val=centers+np.random.multivariate_normal([0,0],[
[1,0],[0,1]],size=len(centers))*sd,parent_string="'spatial_coor_gen_original'")
    alpha = distance_between_centers
    spatial_val = spatial_similarity(spatial_coor.val, alpha, power=2)
    K_loc = KernelData(iid=snpdata.iid,val=spatial_val).standardize()

    #Generate phenotype
    iid = K_causal.iid
    iid_count = K_causal.iid_count
    np.random.seed(seed)
    pheno_causal = SnpData(iid=iid,sid=["causal"],val=np.random.multivariate_normal(np.zeros(iid_count),K_
causal.val).reshape(-1,1),parent_string="causal")
    np.random.seed(seed^998372)
    pheno_noise = SnpData(iid=iid,sid=["noise"],val=np.random.normal(size=iid_count).reshape(-1,1),parent_
string="noise")
    np.random.seed(seed^12230302)
    pheno_loc_original = SnpData(iid=iid,sid=["loc_original"],val=np.random.multivariate_normal(np.zeros(i
id_count),K_loc.val).reshape(-1,1),parent_string="loc_original")

    if do_shuffle:
        idx = np.arange(iid_count)
        np.random.seed(seed)
        np.random.shuffle(idx)
        pheno_loc = pheno_loc_original.read(view_ok=True) #don't need to copy, because the next line will
be fresh memory
        pheno_loc.val = pheno_loc.val[idx,:]
    else:
        pheno_loc = pheno_loc_original

    pheno = SnpData(iid=iid,sid=["pheno_all"],val=pheno_causal.val+pheno_noise.val+pheno_loc.val)

    #Analyze data
    alpha_list = [int(v) for v in np.logspace(np.log10(100),np.log10(1e10), 100)]
    dataframe = heritability_spatial_correction(
        snpdata,
        spatial_coor.val,spatial_iid,
        alpha_list=[alpha] if just_testing else alpha_list,pheno=pheno, alpha_power=2,
        jackknife_count=0,permute_plus_count=0,permute_times_count=0,just_testing=just_testing,
        map_function=map_function,cache_folder=cache_folder)

    logging.info(dataframe)
    return dataframe

def loop_generate_and_analyze(N, seed_count, do_shuffle, just_testing, map_function, cache_folder=None):
    '''
    run generate_and_analyze on different seeds
    '''
    df_list = []
    for seed in xrange(seed_count):
        if cache_folder is not None:
            cache_folder_per_seed = cache_folder.format(seed)
        else:
            cache_folder_per_seed = None
        df = generate_and_analyze(seed=seed,N=N,do_shuffle=do_shuffle,
                          just_testing=just_testing, map_function=map_function, cache_folder=cache_folder
_per_seed)
        df_list.append(df)

    #format the output dataframe
    df2 = pd.concat(df_list)
    df2["seed"] = range(seed_count)
    df2["N"]=N
    df2["do_shuffle"]=do_shuffle
    df3 = df2[["seed","N","do_shuffle", "alpha","h2corr","e2","h2uncorr"]]
    df3.set_index(["seed"],inplace=True)
    return df3
```

Test Run

To test 'loop_generate_and_analyze', we'll run it first locally with 'N=100' and 'just_testing=True'. With these settings, it should return in just a few seconds, but without meaningful results.

In [7]:

```
df = loop_generate_and_analyze(N=100,seed_count=7,do_shuffle=True,just_testing=True,map_function=map)
df
```

Out[7]:

	N	do_shuffle	alpha	h2corr	e2	h2uncorr
seed						
0	100	True	2500000	0	0	0
1	100	True	2500000	0	0	0
2	100	True	2500000	0	0	0
3	100	True	2500000	0	0	0
4	100	True	2500000	0	0	0
5	100	True	2500000	0	0	0
6	100	True	2500000	0	0	0

Full Run

The full run with 5000 individuals and 7 datasets requires a cluster. Here we use HPC cluster (Azure or private), but any cluster that can mimic Python's 'map' function may be used. Even with a cluster, the run can take a long time. If you'd like to get a result more quickly, you could (e.g.) set N=500 and generate fewer datasets.

In [8]:

```
#Define a version of 'map' in terms of a cluster
from fastlmm.association.heritability_spatial_correction import work_item
from fastlmm.util import distributed_map
from fastlmm.util.runner import HPC

azure_runner_function = lambda taskcount: HPC(min(taskcount,10100), 'GCR',r"\\GCR\Scratch\AZ-USCentral\esc
ience",
                                    remote_python_parent=r"\\GCR\Scratch\AZ-USCentral\escience\carlk\d
ata\carlk\pythonpath",
                                    unit='Core', #Core, Socket, Node
                                    update_remote_python_parent=True,
                                    template="Azure IaaS USCentral",
                                    runtime="0:8:0",
                                    mkl_num_threads=1
                                    )

msr_core_runner_function = lambda taskcount: HPC(min(taskcount,10100), 'GCR',r"\\GCR\Scratch\B99\escience"
,
                                        remote_python_parent=r"\\GCR\Scratch\B99\escience\carlk\data\c
arlk\pythonpath",
                                        unit='node', #core, socket, node
                                        update_remote_python_parent=True,
                                        template="ExpressQ",
                                        priority="Normal",
                                        runtime="0:4:0", # day:hour:min
                                        )

map_function = lambda ignore, arg_list, runner_function=msr_core_runner_function, fn_list=[] : distributed
_map.d_map(work_item, arg_list, runner_function(len(arg_list)), fn_list)
#Generate data and analyze it.
N=5000
seed_count=1
do_shuffle=False
cache_pattern=r"D:\data\synth\N{0}\seed{1}".format(N,"{0}")
df = loop_generate_and_analyze(N=N,seed_count=seed_count,do_shuffle=do_shuffle,just_testing=False,
                          map_function=map_function,
                          cache_folder=cache_pattern
                          )
df
```

Out[8]:

	N	do_shuffle	alpha	h2corr	e2	h2uncorr
seed						
0	5000	False	2782559	0.308252	0.399458	0.455674

Here are the outputs for 50 seeds.

seed	N	do_shuffle	alpha	h2corr	e2	h2uncorr
0	5000	FALSE	2782559	0.308	0.399	0.456
1	5000	FALSE	2782559	0.201	0.586	0.408
2	5000	FALSE	2782559	0.399	0.198	0.477
3	5000	FALSE	3351602	0.273	0.465	0.465
4	5000	FALSE	2310129	0.367	0.242	0.460
5	5000	FALSE	2310129	0.396	0.237	0.473
6	5000	FALSE	2310129	0.332	0.331	0.431
7	5000	FALSE	3351602	0.393	0.208	0.482
8	5000	FALSE	2310129	0.391	0.211	0.461
9	5000	FALSE	2310129	0.380	0.267	0.495
10	5000	FALSE	2782559	0.201	0.564	0.431
11	5000	FALSE	2782559	0.292	0.439	0.455
12	5000	FALSE	2310129	0.303	0.423	0.469
13	5000	FALSE	2782559	0.199	0.582	0.386
14	5000	FALSE	2310129	0.405	0.177	0.474
15	5000	FALSE	2310129	0.377	0.264	0.488
16	5000	FALSE	2310129	0.438	0.110	0.469
17	5000	FALSE	2782559	0.293	0.433	0.458
18	5000	FALSE	2310129	0.337	0.342	0.452
19	5000	FALSE	1917910	0.373	0.258	0.450
20	5000	FALSE	2782559	0.359	0.272	0.459
21	5000	FALSE	2310129	0.380	0.230	0.453
22	5000	FALSE	2310129	0.342	0.363	0.482
23	5000	FALSE	2782559	0.273	0.442	0.428
24	5000	FALSE	2310129	0.326	0.359	0.465
25	5000	FALSE	2782559	0.352	0.319	0.484
26	5000	FALSE	2310129	0.336	0.303	0.421
27	5000	FALSE	4037017	0.364	0.297	0.507
28	5000	FALSE	2310129	0.302	0.376	0.407
29	5000	FALSE	2782559	0.259	0.503	0.477
30	5000	FALSE	1321941	0.470	0.063	0.492
31	5000	FALSE	2310129	0.322	0.354	0.451
32	5000	FALSE	2782559	0.256	0.481	0.430
33	5000	FALSE	2782559	0.289	0.398	0.412
34	5000	FALSE	2782559	0.357	0.268	0.476
35	5000	FALSE	2310129	0.314	0.398	0.440
36	5000	FALSE	2782559	0.371	0.280	0.492
37	5000	FALSE	2310129	0.293	0.419	0.405
38	5000	FALSE	2782559	0.285	0.435	0.444
39	5000	FALSE	2310129	0.344	0.314	0.474
40	5000	FALSE	2782559	0.263	0.479	0.452
41	5000	FALSE	2782559	0.253	0.492	0.411
42	5000	FALSE	2782559	0.235	0.515	0.422
43	5000	FALSE	2782559	0.318	0.355	0.431
44	5000	FALSE	2782559	0.353	0.327	0.487
45	5000	FALSE	3351602	0.247	0.504	0.463
46	5000	FALSE	2782559	0.412	0.206	0.501
47	5000	FALSE	2782559	0.287	0.404	0.450
48	5000	FALSE	2310129	0.396	0.259	0.491
49	5000	FALSE	2310129	0.359	0.280	0.471

The mean (± SE) of h2corr, e2, and h2uncorr are 0.33±0.01, 0.35±0.02, and 0.46±0.01, respectively.

Here are the outputs for the same 50 seeds with doShuffle=true, which breaks the dependence between SNPs and location.

seed	N	do_shuffle	alpha	h2corr	e2	h2uncorr
0	5000	TRUE	2782559	0.275	0.465	0.382
1	5000	TRUE	2310129	0.282	0.421	0.382
2	5000	TRUE	2310129	0.396	0.203	0.465
3	5000	TRUE	2310129	0.344	0.323	0.353
4	5000	TRUE	1917910	0.407	0.157	0.450
5	5000	TRUE	2310129	0.386	0.256	0.463
6	5000	TRUE	3351602	0.181	0.634	0.438
7	5000	TRUE	2310129	0.438	0.120	0.446
8	5000	TRUE	1917910	0.444	0.101	0.460
9	5000	TRUE	2782559	0.301	0.418	0.411
10	5000	TRUE	1917910	0.318	0.309	0.414
11	5000	TRUE	2782559	0.315	0.395	0.359
12	5000	TRUE	2310129	0.314	0.400	0.460
13	5000	TRUE	2782559	0.222	0.533	0.393
14	5000	TRUE	1917910	0.399	0.187	0.476
15	5000	TRUE	2782559	0.280	0.453	0.468
16	5000	TRUE	2310129	0.433	0.119	0.468
17	5000	TRUE	2782559	0.311	0.398	0.390
18	5000	TRUE	2782559	0.297	0.421	0.385
19	5000	TRUE	2310129	0.312	0.379	0.458
20	5000	TRUE	2782559	0.354	0.282	0.447
21	5000	TRUE	2310129	0.390	0.209	0.418
22	5000	TRUE	2782559	0.277	0.485	0.337
23	5000	TRUE	2782559	0.277	0.435	0.425
24	5000	TRUE	2310129	0.356	0.299	0.436
25	5000	TRUE	2310129	0.402	0.224	0.473
26	5000	TRUE	1917910	0.352	0.271	0.351
27	5000	TRUE	4862601	0.316	0.388	0.491
28	5000	TRUE	2782559	0.251	0.479	0.365
29	5000	TRUE	2782559	0.299	0.427	0.477
30	5000	TRUE	1592282	0.474	0.056	0.491
31	5000	TRUE	2782559	0.276	0.447	0.413
32	5000	TRUE	2782559	0.266	0.460	0.348
33	5000	TRUE	2310129	0.326	0.322	0.426
34	5000	TRUE	3351602	0.309	0.368	0.476
35	5000	TRUE	2782559	0.246	0.527	0.332
36	5000	TRUE	2310129	0.421	0.184	0.482
37	5000	TRUE	2782559	0.275	0.454	0.400
38	5000	TRUE	2782559	0.269	0.468	0.401
39	5000	TRUE	2310129	0.363	0.275	0.448
40	5000	TRUE	2310129	0.342	0.324	0.451
41	5000	TRUE	2782559	0.273	0.450	0.311
42	5000	TRUE	2310129	0.300	0.381	0.301
43	5000	TRUE	2782559	0.288	0.417	0.423
44	5000	TRUE	2782559	0.377	0.281	0.383
45	5000	TRUE	2782559	0.320	0.358	0.461
46	5000	TRUE	2310129	0.441	0.150	0.464
47	5000	TRUE	3351602	0.258	0.465	0.350
48	5000	TRUE	2310129	0.394	0.262	0.458
49	5000	TRUE	2782559	0.303	0.395	0.421

The mean (± SE) of h2corr, e2, and h2uncorr are 0.33±0.01, 0.35±0.02, and 0.42±0.01, respectively.

Note that the values of h2corr and e2 are consistent with the generating parameters. Also note that h2uncorr is larger than h2corr for 99 of the 100 rows.