Limix-LMM Documentation

Release 0.1.2

Francesco Paolo Casale, Danilo Horta, Oliver Stegle

Jan 27, 2019

Table of contents

1 Install **3**

2 Quick Start in Python **5**
- 2.1 GWAS with Linear Mixed Model 5
- 2.2 Multi-trait Linear Mixed Model (MTLMM) 7

3 Public Interface **9**
- 3.1 LMM Core Classes 9
- 3.2 Plot functions 12

4 Comments and bugs **15**

Python Module Index **17**

Attention: This a standalone module that implements the basic functionalities of LMM-based models for genome-wide association studies. However, we recommend using this module within LIMIX2.

CHAPTER 1

Install

Install using pip:

```
pip install limix-lmm
```

CHAPTER 2

Quick Start in Python

2.1 GWAS with Linear Mixed Model

We here show how to run structLMM and alternative linear mixed models implementations in Python.

```
import os
import numpy as np
import pandas as pd
import scipy as sp
from limix_core.util.preprocess import gaussianize
from limix_core.gp import GP2KronSumLR
from limix_core.covar import FreeFormCov
from limix_lmm import LMM
from limix_lmm import download, unzip
from pandas_plink import read_plink
from sklearn.impute import SimpleImputer
import geno_sugar as gs
import geno_sugar.preprocess as prep

# Download and unzip the dataset files
download("http://www.ebi.ac.uk/~casale/data_structlmm.zip")
unzip("data_structlmm.zip")

# import genotype file
bedfile = "data_structlmm/chrom22_subsample20_maf0.10"
(bim, fam, G) = read_plink(bedfile, verbose=False)

# subsample snps
Isnp = gs.is_in(bim, ("22", 17500000, 18000000))
G, bim = gs.snp_query(G, bim, Isnp)

# load phenotype file
phenofile = "data_structlmm/expr.csv"
dfp = pd.read_csv(phenofile, index_col=0)
```

(continues on next page)

(continued from previous page)

```
pheno = gaussianize(dfp.loc["gene1"].values[:, None])

# mean as fixed effect
covs = sp.ones((pheno.shape[0], 1))

# fit null model
wfile = "data_structlmm/env.txt"
W = sp.loadtxt(wfile)
W = W[:, W.std(0) > 0]
W -= W.mean(0)
W /= W.std(0)
W /= sp.sqrt(W.shape[1])

# larn a covariance on the null model
gp = GP2KronSumLR(Y=pheno, Cn=FreeFormCov(1), G=W, F=covs, A=sp.ones((1, 1)))
gp.covar.Cr.setCovariance(0.5 * sp.ones((1, 1)))
gp.covar.Cn.setCovariance(0.5 * sp.ones((1, 1)))
info_opt = gp.optimize(verbose=False)

# define lmm
lmm = LMM(pheno, covs, gp.covar.solve)

# define geno preprocessing function
imputer = SimpleImputer(missing_values=np.nan, strategy="mean")
preprocess = prep.compose(
    [
        prep.filter_by_missing(max_miss=0.10),
        prep.impute(imputer),
        prep.filter_by_maf(min_maf=0.10),
        prep.standardize(),
    ]
)

# loop on geno
res = []
queue = gs.GenoQueue(G, bim, batch_size=200, preprocess=preprocess)
for _G, _bim in queue:
    lmm.process(_G)
    pv = lmm.getPv()
    beta = lmm.getBetaSNP()
    _bim = _bim.assign(lmm_pv=pd.Series(pv, index=_bim.index))
    _bim = _bim.assign(lmm_beta=pd.Series(beta, index=_bim.index))
    res.append(_bim)

res = pd.concat(res)
res.reset_index(inplace=True, drop=True)

# export
print("Exporting to out/")
if not os.path.exists("out"):
    os.makedirs("out")
res.to_csv("out/res_lmm.csv", index=False)
```

```
.. read 200 / 994 variants (20.12%)
.. read 400 / 994 variants (40.24%)
.. read 600 / 994 variants (60.36%)
```

(continues on next page)

(continued from previous page)

```
.. read 800 / 994 variants (80.48%)
.. read 994 / 994 variants (100.00%)
Exporting to out/
```

2.2 Multi-trait Linear Mixed Model (MTLMM)

```
import scipy as sp
import scipy.linalg as la
from limix_core.gp import GP2KronSum
from limix_core.covar import FreeFormCov

def generate_data(N, P, K, S):
    import scipy as sp

    # fixed eff
    F = sp.randn(N, K)
    B0 = 2 * (sp.arange(0, K * P) % 2) - 1.0
    B0 = sp.reshape(B0, (K, P))
    FB = sp.dot(F, B0)

    # gerenrate phenos
    h2 = sp.linspace(0.1, 0.5, P)

    # generate data
    G = 1.0 * (sp.rand(N, S) < 0.2)
    G -= G.mean(0)
    G /= G.std(0)
    G /= sp.sqrt(G.shape[1])
    Wg = sp.randn(P, P)
    Wn = sp.randn(P, P)

    B = sp.randn(G.shape[1], Wg.shape[1])
    Yg = G.dot(B).dot(Wg.T)
    Yg -= Yg.mean(0)
    Yg *= sp.sqrt(h2 / Yg.var(0))
    Cg0 = sp.cov(Yg.T)

    B = sp.randn(G.shape[0], Wg.shape[1])
    Yn = B.dot(Wn.T)
    Yn -= Yn.mean(0)
    Yn *= sp.sqrt((1 - h2) / Yn.var(0))
    Cn0 = sp.cov(Yn.T)

    Y = FB + Yg + Yn

    return Y, F, G, B0, Cg0, Cn0

N = 1000
P = 4
K = 2
S = 500
Y, F, G, B0, Cg0, Cn0 = generate_data(N, P, K, S)

# compute eigenvalue decomp of RRM
```

(continues on next page)

(continued from previous page)

```
R = sp.dot(G, G.T)
R /= R.diagonal().mean()
R += 1e-4 * sp.eye(R.shape[0])
Sr, Ur = la.eigh(R)

# fit null model
Cg = FreeFormCov(Y.shape[1])
Cn = FreeFormCov(Y.shape[1])
gp = GP2KronSum(Y=Y, S_R=Sr, U_R=Ur, Cg=Cg, Cn=Cn, F=F, A=sp.eye(P))
gp.covar.Cg.setCovariance(0.5 * sp.cov(Y.T))
gp.covar.Cn.setCovariance(0.5 * sp.cov(Y.T))
gp.optimize(factr=10)

# run MTLMM
from limix_lmm import MTLMM
mtlmm = MTLMM(Y, F=F, A=sp.eye(P), Asnp=sp.eye(P), covar=gp.covar)
mtlmm.process(G)
pv = mtlmm.getPv()
B = mtlmm.getBetaSNP()
print(pv[:5])
print(B[:5])
```

```
Marginal likelihood optimization.
    ('Converged:', True)
    Time elapsed: 2.22 s
    Log Marginal Likelihood: 1387.3330379.
    Gradient norm: 0.0000970.
```

A full description of all methods can be found in *Public Interface*.

CHAPTER 3

Public Interface

- *LMM Core Classes*
  - *lmm_core.LMMCore*
  - lmm_core.LMM
- *Plot functions*
  - *plot.plot_manhattan()*
  - *plot.qqplot()*

3.1 LMM Core Classes

class limix_lmm.lmm_core.**LMMCore**(*y*, *F*, *Ki_dot=None*)

Core LMM for interaction and association testing.

The model is

$$\mathbf{y} \sim \mathcal{N}(\underbrace{\mathbf{F}\mathbf{b}}_{\text{covariates}} + \underbrace{(\mathbf{X}\hat{\odot}\mathbf{I})\,\boldsymbol{\beta}}_{\text{genetics}}, \sigma^2 \underbrace{\mathbf{K}_{\boldsymbol{\theta}_0}}_{\text{covariance}})$$

$\hat{\odot}$ is defined as follows. Let

$$\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_m] \in \mathbb{R}^{n \times m}]$$

and

$$\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_l] \in \mathbb{R}^{n \times l}$$

then

$$\mathbf{A}\hat{\odot}\mathbf{B} = [\mathbf{a}_1 \odot \mathbf{b}_1 \ldots, \mathbf{a}_1 \odot \mathbf{b}_l, \mathbf{a}_2 \odot \mathbf{b}_1, \ldots, \mathbf{a}_2 \odot \mathbf{b}_l, \ldots, \mathbf{a}_m \odot \mathbf{b}_l, \ldots, \mathbf{a}_m \odot \mathbf{b}_l] \in \mathbb{R}^{n \times (ml)}$$

where $\odot$ is the element-wise product.

Importantly, $\mathbf{K}_{\boldsymbol{\theta}_0}$ is not re-estimated under the laternative model, i.e. only σ^2 is learnt. The covariance parameters $\boldsymbol{\theta}_0$ need to be learnt using another module, e.g. limix-core. Note that as a consequence of this, such an implementation of single-variant analysis does not require the specification of the whole covariance but just of a method that specifies the product of its inverse by a vector.

The test $\boldsymbol{\beta} \neq 0$ is done in bocks of `step` variants, where `step` can be specifed by the user.

Parameters

- **y** ((*N*, 1) ndarray) – phenotype vector
- **F** ((*N*, L) ndarray) – fixed effect design for covariates.
- **Ki_dot** (*function*) – method that takes an array and returns the dot product of the inverse of the covariance and the input array.

Examples

Example with Inter and step=1

```
>>> from numpy.random import RandomState
>>> import scipy as sp
>>> from limix_lmm.lmm_core import LMMCore
>>> from limix_core.gp import GP2KronSumLR
>>> from limix_core.covar import FreeFormCov
>>> random = RandomState(1)
>>> from numpy import set_printoptions
>>> set_printoptions(4)
>>>
>>> N = 100
>>> k = 1
>>> m = 2
>>> S = 1000
>>> y = random.randn(N,1)
>>> E = random.randn(N,k)
>>> G = 1.*(random.rand(N,S)<0.2)
>>> F = sp.concatenate([sp.ones((N,1)), random.randn(N,1)], 1)
>>> Inter = random.randn(N, m)
>>>
>>> gp = GP2KronSumLR(Y=y, Cn=FreeFormCov(1), G=E, F=F, A=sp.ones((1,1)))
>>> gp.covar.Cr.setCovariance(0.5*sp.ones((1,1)))
>>> gp.covar.Cn.setCovariance(0.5*sp.ones((1,1)))
>>> info_null = gp.optimize(verbose=False)
>>>
>>> lmm = LMMCore(y, F, gp.covar.solve)
>>> lmm.process(G, Inter)
>>> pv = lmm.getPv()
>>> beta = lmm.getBetaSNP()
>>> beta_ste = lmm.getBetaSNPste()
>>> lrt = lmm.getLRT()
>>>
>>> print(pv.shape)
(1000,)
>>> print(beta.shape)
(2, 1000)
>>> print(pv[:4])
[0.1428 0.3932 0.4749 0.3121]
>>> print(beta[:,:4])
```

(continues on next page)

(continued from previous page)

```
[[ 0.0535  0.2571  0.122  -0.2328]
 [ 0.3418  0.3179 -0.2099  0.0986]]
```

Example with step=4

```
>>> lmm.process(G, step=4)
>>> pv = lmm.getPv()
>>> beta = lmm.getBetaSNP()
>>> lrt = lmm.getLRT()
>>>
>>> print(pv.shape)
(250,)
>>> print(beta.shape)
(4, 250)
>>> print(pv[:4])
[0.636  0.3735 0.7569 0.3282]
>>> print(beta[:,:4])
[[-0.0176 -0.4057  0.0925 -0.4175]
 [ 0.2821  0.2232 -0.2124  0.2433]
 [-0.0575  0.1528 -0.0384  0.019 ]
 [-0.2156 -0.2327 -0.1773 -0.1497]]
```

Example with step=4 and Inter

```
>>> lmm = LMMCore(y, F, gp.covar.solve)
>>> lmm.process(G, Inter=Inter, step=4)
>>> pv = lmm.getPv()
>>> beta = lmm.getBetaSNP()
>>> lrt = lmm.getLRT()
>>>
>>> print(pv.shape)
(250,)
>>> print(beta.shape)
(8, 250)
>>> print(pv[:4])
[0.1591 0.2488 0.4109 0.5877]
>>> print(beta[:,:4])
[[ 0.0691  0.1977  0.435  -0.3214]
 [ 0.1701 -0.3195  0.0179  0.3344]
 [ 0.1887 -0.0765 -0.3847  0.1843]
 [-0.2974  0.2787  0.2427 -0.0717]
 [ 0.3784  0.0854  0.1566  0.0652]
 [ 0.4012  0.5255 -0.1572  0.1674]
 [-0.413   0.0278  0.1946 -0.1199]
 [ 0.0268 -0.0317 -0.1059  0.1414]]
```

getBetaCov()
get beta of covariates

Returns **beta**

Return type ndarray

getBetaSNP()
get effect size SNPs

Returns **pv**

Return type ndarray

getBetaSNPste()
get standard errors on betas

Returns beta_ste

Return type ndarray

getLRT()
get lik ratio test statistics

Returns lrt

Return type ndarray

getPv()
Get pvalues

Returns pv

Return type ndarray

process(*G*, *Inter=None*, *step=1*, *verbose=False*)
Fit genotypes one-by-one.

Parameters

- **G** ((*N*, *S*) ndarray) –
- **Inter** ((*N*, *M*) ndarray) – Matrix of *M* factors for *N* inds with which each variant interact By default, Inter is set to a matrix of ones.
- **step** (*int*) – Number of consecutive variants that should be tested jointly.
- **verbose** (*bool*) – verbose flag.

3.2 Plot functions

limix_lmm.plot.**plot_manhattan**(*ax*, *df*, *pv_thr=None*, *colors=None*, *offset=None*, *callback=None*)
Utility function to make manhattan plot

Parameters

- **ax** (*pyplot plot*) – subplot
- **df** (*pandas.DataFrame*) – pandas DataFrame with chrom, pos and pv
- **colors** (*list*) – colors to use in the manhattan plot
- **offset** (*float*) – offset between in chromosome expressed as fraction of the length of the longest chromosome (default is 0.2)
- **callback** (*function*) – callback function that takes as input df

Examples

```
>>> from matplotlib import pyplot as plt
>>> from limix_lmm.plot import plot_manhattan
>>> import scipy as sp
>>> import pandas as pd
>>> n_chroms = 5
>>> n_snps_per_chrom = 10000
```

(continues on next page)

(continued from previous page)

```
>>> chrom = sp.kron(sp.arange(1, n_chroms + 1), sp.ones(n_snps_per_chrom))
>>> pos = sp.kron(sp.ones(n_chroms), sp.arange(n_snps_per_chrom))
>>> pv = sp.rand(n_chroms * n_snps_per_chrom)
>>> df = pd.DataFrame({'chrom': chrom, 'pos': pos, 'pv': pv})
>>>
>>> ax = plt.subplot(111)
>>> plot_manhattan(ax, df)
```

limix_lmm.plot.**qqplot**(*ax*, *pv*, *color='k'*, *plot_method=None*, *line=False*, *pv_thr=0.01*, *plot_xyline=False*, *xy_labels=False*, *xyline_color='r'*)

Utility function to make manhattan plot

Parameters

- **ax** (*pyplot plot*) – subplot
- **df** (*pandas.DataFrame*) – pandas DataFrame with chrom, pos and pv
- **color** (*color*) – colors to use in the manhattan plot (default is black)
- **plot_method** (*function*) – function that takes x and y and plots a custom scatter plot The default value is None.
- **pv_thr** (*float*) – threshold on pvs
- **plot_xyline** (*bool*) – if True, the x=y line is plotteed. The default value is False.

Examples

```
>>> from limix_lmm.plot import qqplot
>>> import scipy as sp
>>> from matplotlib import pyplot as plt
>>>
>>> pv1 = sp.rand(10000)
>>> pv2 = sp.rand(10000)
>>> pv3 = sp.rand(10000)
>>>
>>> ax = plt.subplot(111)
>>> qqplot(ax, pv1, color='C0')
>>> qqplot(ax, pv2, color='C1')
>>> qqplot(ax, pv3, color='C2', plot_xyline=True, xy_labels=True)
```

CHAPTER 4

Comments and bugs

You can get the source and open issues on Github.

Python Module Index

l

limix_lmm.lmm_core, 9
limix_lmm.plot, 12

Index

G

getBetaCov() (limix_lmm.lmm_core.LMMCore method), 11
getBetaSNP() (limix_lmm.lmm_core.LMMCore method), 11
getBetaSNPste() (limix_lmm.lmm_core.LMMCore method), 11
getLRT() (limix_lmm.lmm_core.LMMCore method), 12
getPv() (limix_lmm.lmm_core.LMMCore method), 12

L

limix_lmm.lmm_core (module), 9
limix_lmm.plot (module), 12
LMMCore (class in limix_lmm.lmm_core), 9

P

plot_manhattan() (in module limix_lmm.plot), 12
process() (limix_lmm.lmm_core.LMMCore method), 12

Q

qqplot() (in module limix_lmm.plot), 13