LIMIX: genetic analysis of multiple traits

Christoph Lippert, F. Paolo Casale, Barbara Rakitsch, Oliver Stegle

May 13, 2014

Contents

1 Overview

This tutorial illustrates the use of LIMIX to build single-trait and multi-trait mixed models for the genetic analysis of quantitaitive traits. For this illustration, we consider gene expression levels from a yeast genetics study with freely available data [1]. These data span 109 individuals with 2,956 marker SNPs and expression levels for 5,493 in glucose and ethanol growth media respectively.

We start out by discussing how to do QTL mapping, implement models that consider multi loci and introduce the application of variance component models for single quantitative traits. Subsequently, these analysis are extended to the corresponding multi-trait models, combining the corresponding quantitative traits across both enviornment or multiple genes in a pathway. Finally, we show how LIMIX can be used to fit variance component models that account for gene expression heteorgeneity by fitting a suitale covariance matrix.

2 Setting up

```
In [1]: # activiate inline plotting
        %matplotlib inline
```

```
In [2]: import scipy as SP
        import pylab as PL
        from matplotlib import cm
        import h5py
        import pdb
        import pandas as pd
        SP.random.seed(0)
```

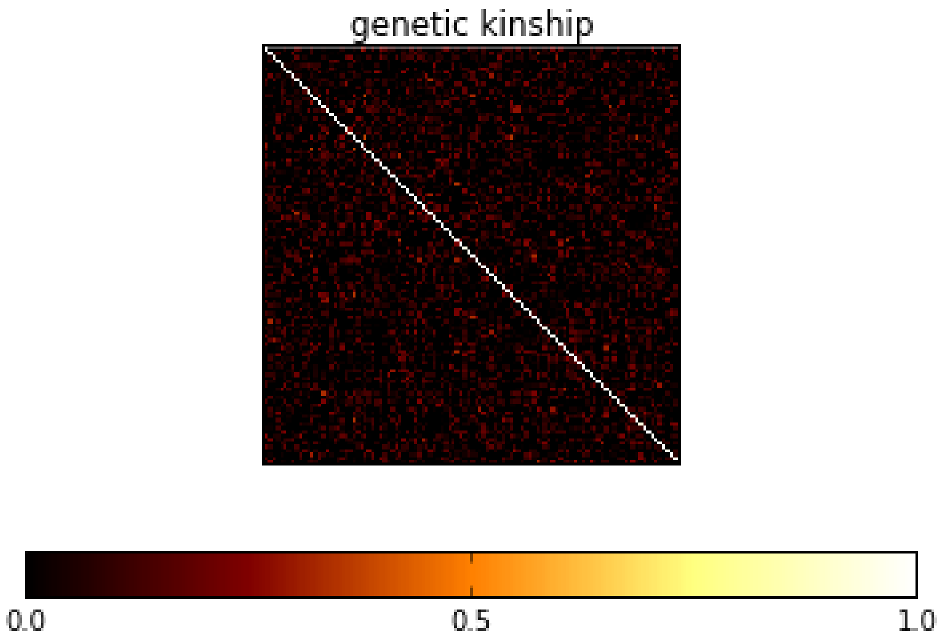
```
In [3]: # import LIMIX
        import sys
        import limix.modules.varianceDecomposition as VAR
        import limix.modules.qtl as QTL
        import limix.modules.data as DATA
        import limix.modules.genotype_reader as gr
        import limix.modules.phenotype_reader as phr
        import limix.modules.data_util as data_util
        # plotting and visualization utilties
        from limix.utils.util import *
        from limix.utils.plot import plot_manhattan
```

```
In [4]: #import data
        #the data used in this study have been pre-converted into an hdf5 file.
        #to preprocess your own data, please use limix command line brinary
        file_name = 'data/smith_2008/smith08.hdf5'
        geno_reader  = gr.genotype_reader_tables(file_name)
        pheno_reader = phr.pheno_reader_tables(file_name)

        #the data object allows to query specific genotype or phenotype data
        data = DATA.QTLData(geno_reader=geno_reader,pheno_reader=pheno_reader)
        #getting genotypes
        snps = data.getGenotypes() #SNPs
        position = data.getPos()
        chromBounds = data_util.estCumPos(position=position,offset=100000)

        # non-normalized and normalized sample relatedeness matrix
        sample_relatedness_unnormalized = data.getCovariance(normalize=False)
        sample_relatedness  = sample_relatedness_unnormalized/ \
            sample_relatedness_unnormalized.diagonal().mean()
```

```
In [5]: #genes from lysine biosynthesis pathway
        lysine_group = ['YIL094C', 'YDL182W', 'YDL131W', 'YER052C', 'YBR115C', 'YDR158W',
                        'YNR050C', 'YJR139C', 'YIR034C', 'YGL202W', 'YDR234W']
```

```
In [6]: # plot (genetic) sample relatedeness matrix
        plt = PL.subplot(1,1,1)
        PL.title('genetic kinship')
        PL.imshow(sample_relatedness,vmin=0,vmax=1,interpolation='none',cmap=cm.afmhot)
        PL.colorbar(ticks=[0,0.5,1],orientation='horizontal')
        plt.set_xticks([])
        plt.set_yticks([])
```

```
Out[6]: []
```



3 Single Trait Analysis

3.1 Marginal single Locus Analysis

3.1.1 The Genetic Model

Indicating with N the number of samples, the standard LMM considered by LIMIX is

$$\mathbf{y} = \mathbf{F}\boldsymbol{\alpha} + \mathbf{x}\beta + \mathbf{g} + \boldsymbol{\psi}, \quad \mathbf{g} \sim \mathcal{N}\left(\mathbf{0}, \sigma_g^2\mathbf{R}\right),\ \boldsymbol{\psi} \sim \mathcal{N}\left(\mathbf{0}, \sigma_e^2\mathbf{I}_N\right) \tag{1}$$

where

$$\begin{aligned}
\mathbf{y} &= \text{phenotype vector} \in \mathcal{R}^{N,1} && (2)\\
\mathbf{F} &= \text{matrix of } K \text{ covariates} \in \mathcal{R}^{N,K} && (3)\\
\boldsymbol{\alpha} &= \text{effect of covariates} \in \mathcal{R}^{K,1} && (4)\\
\mathbf{x} &= \text{genetic profile of the SNP being tested} \in \mathcal{R}^{N,1} && (5)\\
\boldsymbol{\beta} &= \text{effect size of the SNP} \in \mathcal{R} && (6)\\
\mathbf{R} &= \text{sample relatedeness matrix} \in \mathcal{R}^{N,N} && (7)\\
& && (8)
\end{aligned}$$

Association between phenotypic changes and the genetic markers is tested by testing $\beta \neq 0$.
The model can be rewritten using the delta-representation

$$\mathbf{y} \sim \mathcal{N}\left(\mathbf{W}\boldsymbol{\alpha} + \mathbf{x}\beta, \sigma_g^2\left(\mathbf{R} + \delta\mathbf{I}\right)\right) \tag{9}$$

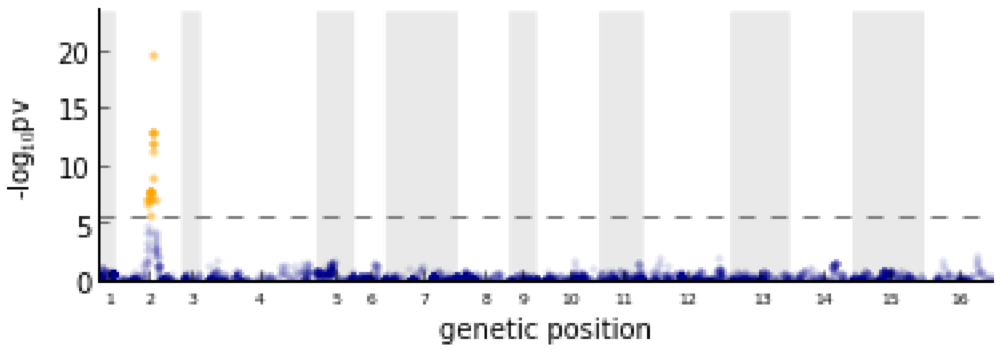
where $\delta = \sigma_e^2/\sigma_g^2$ is the signal-to-noise ratio, which can be efficiently fit for every genome wide test (parameter searchDelta). For efficient inference, LIMIX uses the identical speedups implemented in [2] and GEMMA [3].

3.1.2 Example: Single-trait eQTL mapping for gene YBR115C in the glucose condition

```
In [7]: #create a complex query on the gene_ID and environment:
        # select environment 0 (glucose) for gene YBR115C
        phenotype_query = "(gene_ID=='YBR115C') & (environment==0)"

        # getting the appropriate data subset
        data_subsample = data.subsample_phenotypes(phenotype_query=phenotype_query,
                                                   intersection=True)

        #get variables we need from data
        snps = data_subsample.getGenotypes(impute_missing=True)
        phenotypes,sample_idx = data_subsample.getPhenotypes(phenotype_query=phenotype_query,
                                                             intersection=True);
        assert sample_idx.all()

        #set parameters for the analysis
        N, P = phenotypes.shape
        S    = snps.shape[1]

        print "loaded %d samples, %d phenotypes, %s snps" % (N,P,S)

        covs = None                 #covariates
        searchDelta = False         #specify if delta should be optimized for each SNP
        test="lrt"                  #specify type of statistical test

        # Running the analysis
        # when cov are not set (None), LIMIX considers an intercept (covs=SP.ones((N,1)))
        lmm = QTL.test_lmm(snps=snps,pheno=phenotypes.values,
                           K=sample_relatedness,covs=covs, test=test)

        pvalues = lmm.getPv()       # 1xS vector of p-values (S=X.shape[1])
        #convert P-values to a DataFrame for nice output writing:
        pvalues = pd.DataFrame(data=pvalues.T,index=data_subsample.geno_ID,
                               columns=phenotypes.columns)
        pvalues = pd.concat([position,pvalues],join="outer",axis=1)

        betas = lmm.getBetaSNP()    # 1xS vector of effect sizes (S=X.shape[1])
        #convert betas to a DataFrame for nice output writing:
        betas = pd.DataFrame(data=betas.T,index=data_subsample.geno_ID,
                             columns=phenotypes.columns)
        betas = pd.concat([position,pvalues],join="outer",axis=1)
```

```
loaded 109 samples, 1 phenotypes, 2956 snps
```

```
In [8]: plt = PL.subplot(2,1,1)
        plot_manhattan(plt,pvalues['pos_cum'],pvalues['YBR115C:0'],chromBounds)
```

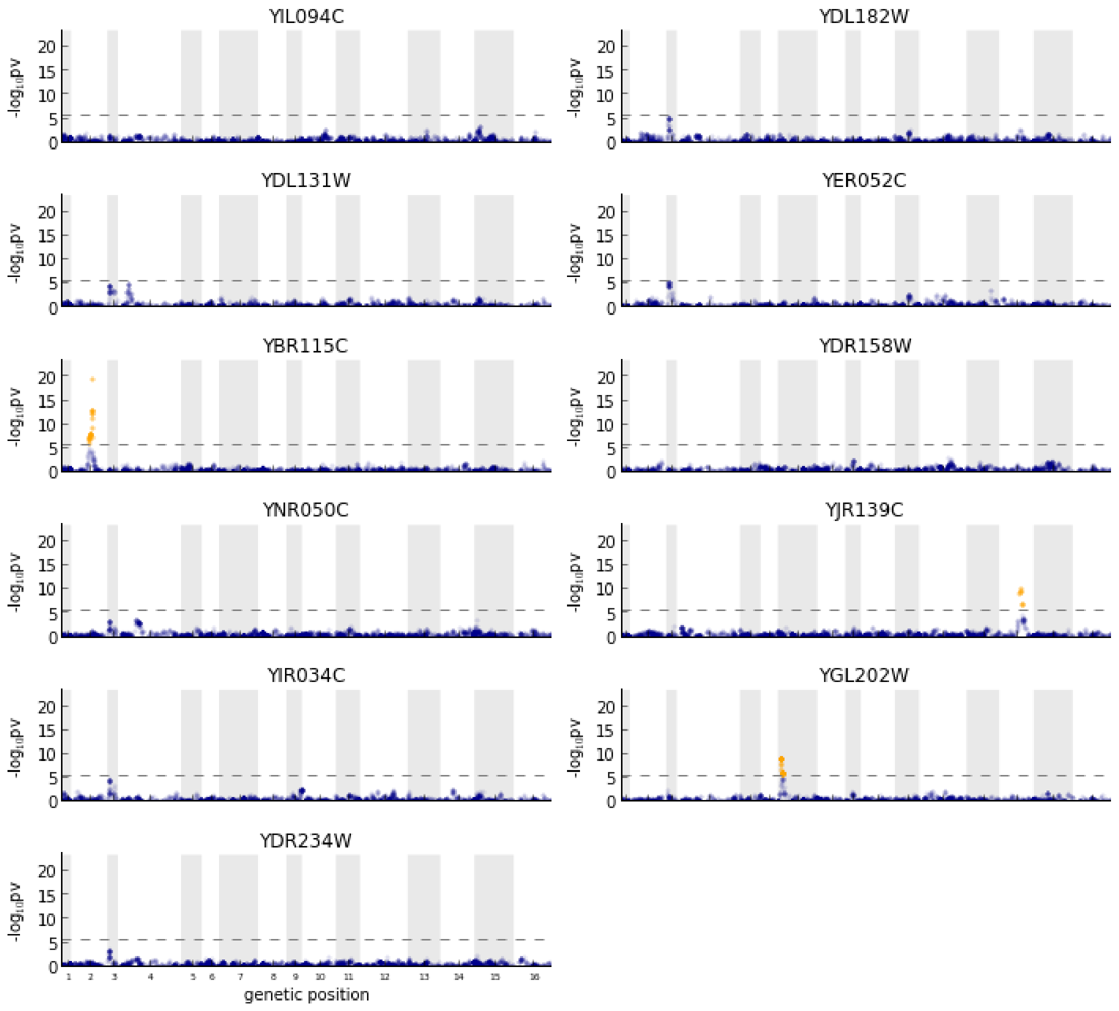


3.1.3 Example: Single-trait eQTL mapping for all genes in the pathways in the glucos condition

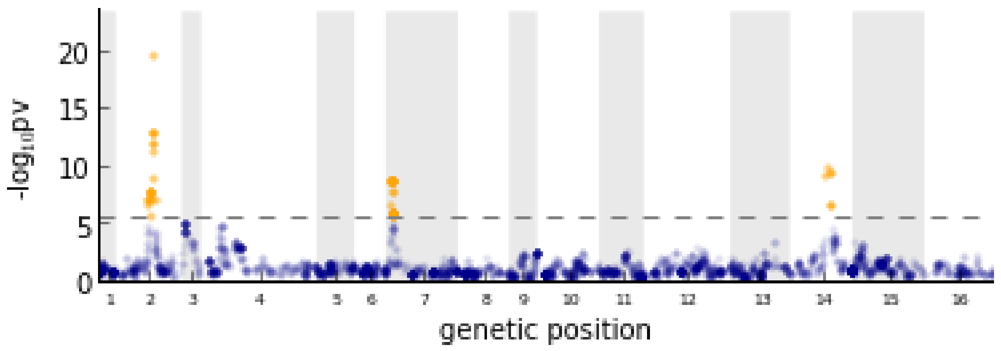
```
In [9]: #create a complex query on the gene_ID and environment:
        # select environment 0 for all genes in lysine_group
        phenotype_query = "(gene_ID in %s) & (environment==0)" % str(lysine_group)

        data_subsample = data.subsample_phenotypes(phenotype_query=phenotype_query,
                                                   intersection=True)

        #get variables we need from data
        snps = data_subsample.getGenotypes(impute_missing=True)
        phenotypes,sample_idx = data_subsample.getPhenotypes(phenotype_query=phenotype_query,
                                                             intersection=True)
        assert sample_idx.all()

        sample_relatedness = data_subsample.getCovariance()

        #set parameters for the analysis
        N, P = phenotypes.shape
        S    = snps.shape[1]

        print "loaded %d samples, %d phenotypes, %s snps" % (N,P,S)

        covs = None                 #covariates
        searchDelta = False         #specify if delta is optimized for each SNP
        test="lrt"                  #specify type of statistical test
        # Running the analysis
        # when cov are not set (None), LIMIX considers an intercept
        # (covs=SP.ones((N,1))) note, as phenotypes is a matrix,
        # LIMIX automatically carries out an association scan for each phenotype
        lmm = QTL.test_lmm(snps=snps,pheno=phenotypes.values,
                           K=sample_relatedness,covs=covs,test=test)

        # get p-values
        pvalues = lmm.getPv()       # 1xS vector of p-values (S=X.shape[1])
        #convert P-values to a DataFrame for nice output writing:
        pvalues = pd.DataFrame(data=pvalues.T,index=data_subsample.geno_ID,
                               columns=phenotypes.columns)
        pvalues = pd.concat([position,pvalues],join="outer",axis=1)

        betas = lmm.getBetaSNP()    # 1xS vector of effect sizes (S=X.shape[1])
        #convert betas to a DataFrame for nice output writing:
        betas = pd.DataFrame(data=betas.T,index=data_subsample.geno_ID,
                             columns=phenotypes.columns)
        betas = pd.concat([position,pvalues],join="outer",axis=1)
```

```
loaded 109 samples, 11 phenotypes, 2956 snps
```

```
In [10]: # plotting all manhattan plots
         PL.figure(1,figsize=(10,16))
         lim = -1.2*SP.log10(lmm.getPv().min())
         for g in range(P):
             plt = PL.subplot(P,2,g+1)
             PL.title(lysine_group[g])
             if g!=P-1:   xticklabels = False
             else:        xticklabels = True
             plot_manhattan(plt,pvalues['pos_cum'],pvalues[lysine_group[g]+":0"],
                            chromBounds,lim=lim,xticklabels=xticklabels)
         PL.tight_layout()
```



```
In [11]: # plotting minimum pv across genes
         pv_min = lmm.getPv().min(0)
         plt = PL.subplot(2,1,1)
         plot_manhattan(plt,pvalues['pos_cum'],pv_min,chromBounds)
```



3.2 Multi locus Single Trait Model

3.2.1 The Genetic Model

In addition to a single-locus marginal analysis, LIMIX can be used to performa multi-locus scan by step-wise regression. In the univariate (single-trait) case, this approach is analogus to the proecudere described in [4]. Briefly, the standard univariate scan is extended by iterative inclusion of the most associated SNP as a covariate. The proceudere is terminated after a predefined p-value or q-value treshold or the maximum number of iterations is reached.

Note, care needs to be taken when interpreting these signifance levels as they are compromised by an intrinsic multiple testing bias. It is adviced to interpret the model output carefully, consdering both signifiance levels and other metrics, such as BIC [4].

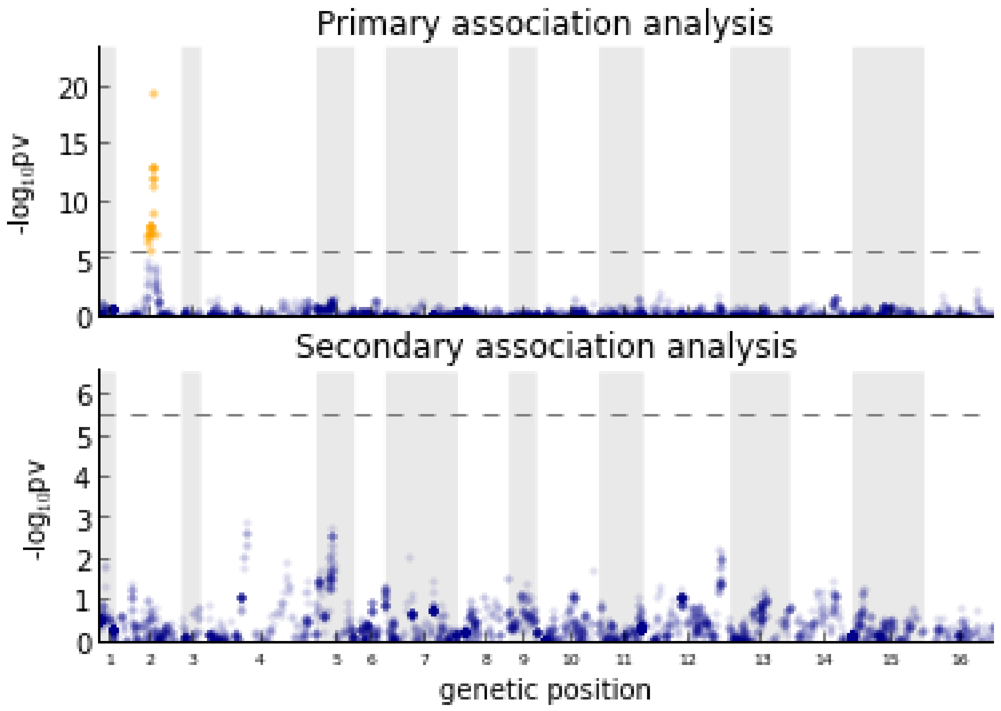
3.2.2 Example: Single-trait eQTL mapping for gene YBR115C the glucose condition

```
In [12]: #getting data
         #create a complex query on the gene_ID and environment:
         #select environment 0 for gene YBR115C
         phenotype_query = "(gene_ID=='YBR115C') & (environment==0)"

         data_subsample = data.subsample_phenotypes(phenotype_query=phenotype_query,
                                                    intersection=True)

         #get variables we need from data
         snps = data_subsample.getGenotypes(impute_missing=True)
         phenotypes,sample_idx = data_subsample.getPhenotypes(phenotype_query=phenotype_query,
                                                               intersection=True)
         assert sample_idx.all()

         sample_relatedness = data_subsample.getCovariance()

         #set parameters for the analysis
         N, P = phenotypes.shape
         S    = snps.shape[1]

         print "loaded %d samples, %d phenotypes, %s snps" % (N,P,S)

         covs = None                 #covariates
         searchDelta = False         #specify if delta should be optimized for each SNP
         test="lrt"                  #specify type of statistical test

         # Running the analysis
         threshold = 0.01/snps.shape[1]
         # inclusion threshold on pvalues
         #(Bonferroni corrected significance level of 1%)
         maxiter = 10                # maximum number of iterations
         lmm, RV = QTL.forward_lmm(snps,phenotypes.values,K=sample_relatedness,
                                   qvalues=False,threshold=threshold,maxiter=maxiter)
         # RV contains:
         # - iadded:   array of indices of SNPs included in order of inclusion
         # - pvadded:  array of Pvalues obtained by the included SNPs before inclusion
         # - pvall:   [maxiter x S] SP.array of Pvalues for all iterations
```

```
loaded 109 samples, 1 phenotypes, 2956 snps
```

```
In [13]: # plot primay p-values and conditional association results
```

```
pvall = RV['pvall']
plt = PL.subplot(2,1,1)
PL.title('Primary association analysis')
plot_manhattan(plt,pvalues['pos_cum'],pvall[0],chromBounds,xticklabels = False)
plt = PL.subplot(2,1,2)
PL.title('Secondary association analysis')
plot_manhattan(plt,pvalues['pos_cum'],pvall[1],chromBounds,xticklabels = True)
```



3.3 Variance Component Models

In addition to association testing using fixed effects, LIMIX enables flexible fitting of variance component models. Here, we illustrate the usage of variance component models fit to single genes.

3.3.1 The Genetic Model

The model considered by the LIMIX variance decomposition module is an extension of the genetic model employed in standard GWAS, however considers multiple random effects terms:

$$\mathbf{y} = \mathbf{F}\boldsymbol{\alpha} + \sum_x \mathbf{u}^{(x)} + \boldsymbol{\psi}, \quad \mathbf{u}^{(x)} \sim \mathcal{N}\left(\mathbf{0}, {\sigma^{(x)}}^2 \mathbf{R}^{(x)}\right), \; \boldsymbol{\Psi} \sim \mathcal{N}\left(\mathbf{0}, \sigma_e^2 \mathbf{I}_N\right) \tag{10}$$

where

$$\mathbf{y} = \text{phenotype vector} \in \mathcal{R}^{N,1} \tag{11}$$
$$\mathbf{F} = \text{matrix of } K \text{ covariates} \in \mathcal{R}^{N,K} \tag{12}$$
$$\boldsymbol{\alpha} = \text{effect of covariates} \in \mathcal{R}^{K,1} \tag{13}$$
$$\mathbf{R}^{(x)} = \text{is the sample covariance matrix for contribution } x \in \mathcal{R}^{N,N} \tag{14}$$
$$\tag{15}$$

If $\mathbf{R}$ is a genetic contribution from set of SNPs $\mathcal{S}$, with a bit of abuse of notation we can write $\mathbf{R} = \frac{1}{C}\mathbf{X}_{:,\mathcal{S}}\mathbf{X}_{:,\mathcal{S}}{}^T$ where $C = \frac{1}{N}\text{trace}\left(\mathbf{X}_{:,\mathcal{S}_i}\mathbf{X}_{:,\mathcal{S}_i}{}^T\right)$.

LIMIX supports an arbitrary number of fixed or random effects to be included in the model.

3.3.2 Example: cis/trans Variance Decomposition in the glucose condition

Here we use the LIMIX variance decomposition module to quantify the variability in gene expression explained by proximal (cis) and distal (trans) genetic variation. To do so, we build a linear mixed model with a fixed effect intercept, two random effects for cis and trans genetic effects and a noise random effect:

$$\mathbf{y} = \mathbf{1}_N\mu+\mathbf{u}^{(cis)}+\mathbf{u}^{(trans)}+\boldsymbol{\psi}, \quad \mathbf{u}^{(cis)} \sim \mathcal{N}\left(\mathbf{0}, {\sigma^{(x)}}^2\mathbf{R}^{(cis)}\right), \mathbf{u}^{(trans)} \sim \mathcal{N}\left(\mathbf{0}, {\sigma^{(x)}}^2\mathbf{R}^{(trans)}\right), \boldsymbol{\Psi} \sim \mathcal{N}\left(\mathbf{0}, \sigma_e^2\mathbf{I}_N\right) \tag{16}$$

where $\mathbf{R}^{(\text{cis})}$ and $\mathbf{R}^{(\text{trans})}$ are the local and distal relatedeness matrices, built considering all SNPs in cis and trans (i.e., not in cis) respectively. As cis region is defined by the 50kb region around each gene.

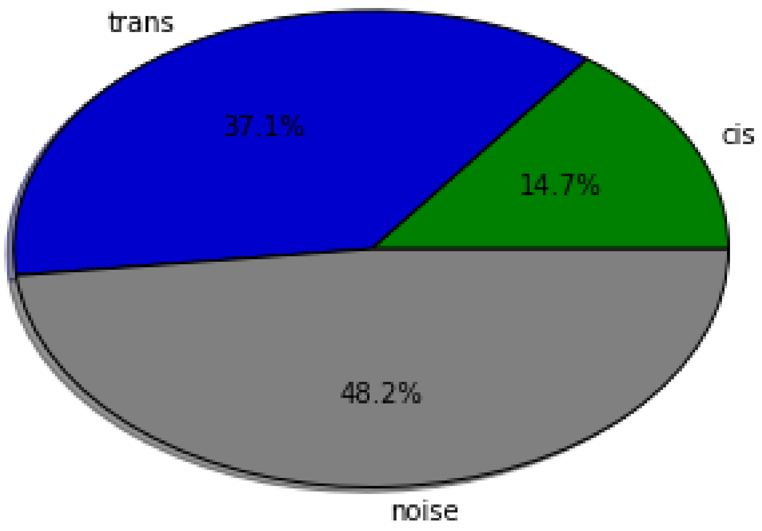
The gene-model is fitted to gene expression in environment 0 for all genes in the Lysine Biosynthesis pathway and variance components are averaged thereafter to obtain pathway based variance components.

```
In [14]: # loop over genes and fit variance component model
         var = []
         genes_analyzed = []
         K_all = data.getCovariance(normalize=False)

         for gene_idx,gene in enumerate(lysine_group):
             #create a complex query on the gene_ID and environment:
             # select environment 0 for all genes in lysine_group
             phenotype_query = "(gene_ID=='%s') & (environment==0)" % gene
             print "running variance decomposition on %s" % phenotype_query
             data_subsample = data.subsample_phenotypes(phenotype_query=phenotype_query,
                                                        intersection=True)

             #get variables we need from data
             phenotypes,sample_idx = data_subsample.getPhenotypes(phenotype_query=phenotype_query,
                                                                  intersection=True)
             assert sample_idx.all()
             if phenotypes.shape[1]==0:
                 continue
             #snps = data_subsample.getGenotypes(impute_missing=True)
             pos_gene = data_subsample.pheno_reader.get_pos(phenotype_query=phenotype_query)
             K_cis = data_subsample.getCovariance(normalize=False,pos_start=pos_gene["start"][0],
                                                  pos_end=pos_gene["end"][0],windowsize=5e5)
             K_trans = K_all-K_cis
             K_cis /= K_cis.diagonal().mean()
             K_trans /= K_trans.diagonal().mean()

             # variance component model
             vc = VAR.VarianceDecomposition(phenotypes.values)
             vc.addFixedEffect()
             vc.addRandomEffect(K=K_cis)
             vc.addRandomEffect(K=K_trans)
             vc.addRandomEffect(is_noise=True)
             vc.optimize()

             # get variances
```

```
            # vc.getVariances() returs a vector of variances explained
            # by the three random effects in order of addition (cis,trans,noise)
            _var = vc.getVarianceComps()

            var.append(_var)
            genes_analyzed.append(gene)
        # concatenate in a unique matrix
        var = SP.concatenate(var)
```

```
running variance decomposition on (gene_ID=='YIL094C') & (environment==0)
running variance decomposition on (gene_ID=='YDL182W') & (environment==0)
running variance decomposition on (gene_ID=='YDL131W') & (environment==0)
running variance decomposition on (gene_ID=='YER052C') & (environment==0)
running variance decomposition on (gene_ID=='YBR115C') & (environment==0)
running variance decomposition on (gene_ID=='YDR158W') & (environment==0)
running variance decomposition on (gene_ID=='YNR050C') & (environment==0)
running variance decomposition on (gene_ID=='YJR139C') & (environment==0)
running variance decomposition on (gene_ID=='YIR034C') & (environment==0)
running variance decomposition on (gene_ID=='YGL202W') & (environment==0)
running variance decomposition on (gene_ID=='YDR234W') & (environment==0)
```

```
In [15]: #normalize variance component and average
         var/=var.sum(1)[:,SP.newaxis]
         var_mean = var.mean(0)
         column_labels = ["cis","trans","noise"]
         colors = ['Green','MediumBlue','Gray']
         PL.pie(var_mean,labels=column_labels,autopct='%1.1f%%',colors=colors,
                shadow=True, startangle=0)
         #convert var to a DataFrame for nice output writing:
         var = pd.DataFrame(data=var,index=genes_analyzed,columns=column_labels)

         #convert betas to a DataFrame for nice output writing:
         var_mean = pd.DataFrame(data=var_mean[SP.newaxis,:],index=["lysine_biosynthesis"],
                                 columns=column_labels)
```



4 Multi Trait Analysis

Denoting with N and P the number of samples and phenotypes in the analysis, LIMIX models the $N \times P$ matrix-variate phenotype $\mathbf{Y}$ as a sum of fixed and random effects in a multivariate mixed model framework.

Single-trait models only consider the across-sample axis of variation and thus - fixed effects need only specification of a sample design (denoted by $\mathbf{F}$ so far) - random effects need only specification of a sample covariance matrix (denoted by $\mathbf{R}$ so far), which describes the correlation across samples induced by the random effect

Conversely, multi-trait models also consider the across-trait axis of variation and thus - fixed effects also require a trait design (denoted with $\mathbf{A}$ in the following) - random effects also require a trait covariance matrix (denoted with $\mathbf{C}$ in the following), which describes the correlation across traits induced by the random effect

Different forms of trait designs for fixed effects and trait covariances for random effects reflect different hypothesis on how these contributions affect phenotypes (e.g., shared and trait-specific effects). In the following, we will show how LIMIX can be used for different analysis of multiple traits wisely choosing trait designs and covariances.

We decided to start with multi-trait variance components models before going to multi-trait GWAS analysis, including single-locus and multi-locus analysis. In the next section we will be discussing more subtle problems as regularization and model selection.

4.1 Variance decomposition

4.1.1 Genetic Model

The general multi-trait linear mixed models considered in the LIMIX variance decomposition module can be written as

$$\mathbf{Y} = \sum_i \mathbf{F}_i \mathbf{W}_i \mathbf{A}_i^{(\mathrm{cov})} + \sum_x \mathbf{U}^{(x)} + \mathbf{\Psi}, \quad \mathbf{U}^{(x)} \sim \mathrm{MVN}\left(\mathbf{0}, \mathbf{R}^{(x)}, \mathbf{C}^{(x)}\right),\ \mathbf{\Psi} \sim \mathrm{MVN}\left(\mathbf{0}, \mathbf{I}_N, \mathbf{C}^{(\mathrm{noise})}\right) \tag{17}$$

where

$$\begin{aligned}
\mathbf{Y} &= \text{phenotype vector} \in \mathcal{R}^{N,P} && (18)\\
\mathbf{F}_i &= \text{sample designs for fixed effects} \in \mathcal{R}^{N,K} && (19)\\
\mathbf{W}_i &= \text{effect size matrix of covariates} \in \mathcal{R}^{K,L} && (20)\\
\mathbf{A}^{(\mathrm{cov})} &= \text{trait design for the fixed effect} \in \mathcal{R}^{L,P} && (21)\\
\mathbf{R}^{(x)} &= \text{sample covariance matrix for contribution } x \in \mathcal{R}^{N,N} && (22)\\
\mathbf{C}^{(x)} &= \text{trait covariance matrix for contribution } x \in \mathcal{R}^{P,P} && (23)\\
\mathbf{C}^{(\mathrm{noise})} &= \text{residual trait covariance matrix} \in \mathcal{R}^{P,P} && (24)\\
& && (25)
\end{aligned}$$

The variance decomposition module in LIMIX allows the user to build the model by adding any number of fixed and random effects, specify different designs for fixed effects and choose covariance models for random effects. In case of only two random effects LIMIX uses speedups beaking the $O(N^3P^3)$ complexity in $O(N^3 + P^3)$.

In the following we show two different example. In the first example we construct interpretable trait covariances for random effects and build a model that dissects gene expression variability across environmental and cis and trans genetic effects, and their interactions (cis GxE, trans GxE). In the second example we show how to use the fast implementation for a two-random-effect model, one for genetic effects and the other for residuals. Such a model is the natural extention to multi-trait analysis of the null model in single-trait GWAS and it is the first step of any multi-trait GWAS analysis performed in LIMIX.

4.1.2 Example 1: GxE variance decomposition for Lysine Biosynthesis

Indicating with N and $E = 2$ respectively the number of samples and the number of environments, we considered for each gene a model consisting of an envoronment-specific intercept term, a random effect for cis genetic effects, a random effect for trans genetic effects, and a noise random effect term.

The model can be written

$$\mathbf{Y} = \mathbf{FWA} + \mathbf{U}^{(\mathrm{cis})} + \mathbf{U}^{(\mathrm{trans})}\boldsymbol{\Psi} \tag{26}$$

$$\mathbf{U}^{(\mathrm{cis})} \sim \mathrm{MVN}(\mathbf{0}, \mathbf{R}^{(\mathrm{cis})}, \mathbf{C}^{(\mathrm{cis})}), \mathbf{U}^{(\mathrm{trans})} \sim \mathrm{MVN}(\mathbf{0}, \mathbf{R}^{(\mathrm{trans})}, \mathbf{C}^{(\mathrm{trans})}),\ \boldsymbol{\Psi} \sim \mathrm{MVN}(\mathbf{0}, \mathbf{I}_N, \mathbf{C}_n)$$

where $\mathbf{R}^{(\mathrm{cis})}$ and $\mathbf{R}^{(\mathrm{trans})}$ are the local and distal kinships, built considering all SNPs in cis and trans (i.e., not in cis) respectively. As in the single-trait analysis, we considered as cis region the region of 500kb downstream and upstream the gene.

To dissect persistent and specific variance compoents we considered a 'block+diagonal' form for $C^{(x)}$:

$$C^{(x)} = {a^{(x)}}^2 \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} {c_1^{(x)}}^2 & 0 \\ 0 & {c_2^{(x)}}^2 \end{bmatrix} = \begin{bmatrix} {a^{(x)}}^2 + {c_1^{(x)}}^2 & {a^{(x)}}^2 \\ {a^{(x)}}^2 & {a^{(x)}}^2 + {c_2^{(x)}}^2 \end{bmatrix} \tag{27}$$

Variance of persistent and specific (GxE) effects from term x are ${a^{(x)}}^2$ and $\frac{1}{2}\left({c_1^{(x)}}^2 + {c_2^{(x)}}^2\right)$ while the variance explained by pure environmental shift can be calculated as $\mathrm{var}\left(\mathrm{vec}(\mathbf{FWA})\right)$.

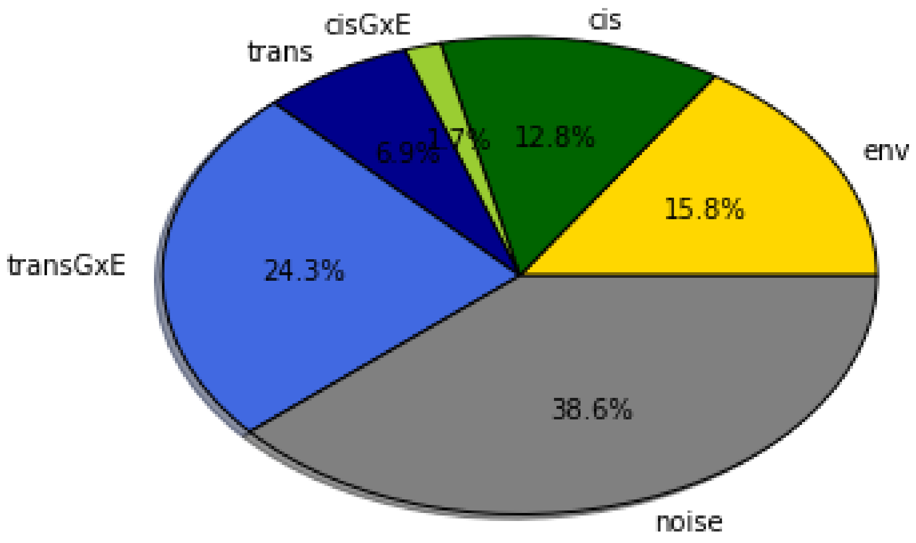
```
In [16]: # loop over genes and fit variance component model
         var = []
         genes_analyzed = []
         K_all = data.getCovariance(normalize=False)

         for gene_idx,gene in enumerate(lysine_group):
             #create a complex query on the gene_ID and environment:
             # select (all environments for) the current gene in lysine_group
             phenotype_query = "(gene_ID=='%s')" % gene
             print "running variance decomposition on %s" % phenotype_query
             data_subsample = data.subsample_phenotypes(phenotype_query=phenotype_query,
                                                        intersection=True)

             #get variables we need from data
             phenotypes,sample_idx = data_subsample.getPhenotypes(phenotype_query=phenotype_query,
                                                                  intersection=True,center=False)
             assert sample_idx.all()
             phenotypes-=phenotypes.values.mean()
             phenotypes/=phenotypes.values.std()
             if phenotypes.shape[1]==0:#gene not found
                 continue
             #snps = data_subsample.getGenotypes(impute_missing=True)
             pos_gene = data_subsample.pheno_reader.get_pos(phenotype_query=phenotype_query)
             K_cis = data_subsample.getCovariance(normalize=False,pos_start=pos_gene["start"][0],
                                                  pos_end=pos_gene["end"][0],windowsize=5e5)
             K_trans = K_all-K_cis
             K_cis /= K_cis.diagonal().mean()
             K_trans /= K_trans.diagonal().mean()

             # variance component model
             vc = VAR.VarianceDecomposition(phenotypes.values)
             vc.addFixedEffect()
```

```
            vc.addRandomEffect(K=K_cis,trait_covar_type='block_diag')
            vc.addRandomEffect(K=K_trans,trait_covar_type='block_diag')
            vc.addRandomEffect(is_noise=True,trait_covar_type='block_diag')
            vc.optimize()

            # environmental variance component
            vEnv = vc.getWeights().var()
            # cis and cisGxE variance components
            Ccis  = vc.getTraitCovar(0)
            a2cis = Ccis[0,1]
            c2cis = Ccis.diagonal().mean()-a2cis
            # trans and cisGxE variance components
            Ctrans  = vc.getTraitCovar(1)
            a2trans = Ctrans[0,1]
            c2trans = Ctrans.diagonal().mean()-a2trans
            # noise variance components
            Cnois = vc.getTraitCovar(2)
            vNois = Cnois.diagonal().mean()

            # Append
            var.append(SP.array([[vEnv,a2cis,c2cis,a2trans,c2trans,vNois]]))
            genes_analyzed.append(gene)
        # concatenate in a unique matrix
        var = SP.concatenate(var)
```

```
running variance decomposition on (gene_ID=='YIL094C')
running variance decomposition on (gene_ID=='YDL182W')
running variance decomposition on (gene_ID=='YDL131W')
running variance decomposition on (gene_ID=='YER052C')
running variance decomposition on (gene_ID=='YBR115C')
running variance decomposition on (gene_ID=='YDR158W')
running variance decomposition on (gene_ID=='YNR050C')
running variance decomposition on (gene_ID=='YJR139C')
running variance decomposition on (gene_ID=='YIR034C')
running variance decomposition on (gene_ID=='YGL202W')
running variance decomposition on (gene_ID=='YDR234W')
```

```
In [17]: #normalize variance component and average
         var/=var.sum(1)[:,SP.newaxis]
         var_mean = var.mean(0)
         labels = ['env','cis','cisGxE','trans','transGxE','noise']
         colors = ['Gold','DarkGreen','YellowGreen','DarkBlue','RoyalBlue','Gray']
         PL.pie(var_mean,labels=labels,autopct='%1.1f%%',colors=colors,shadow=True, startangle=0)
```

```
Out[17]: ([<matplotlib.patches.Wedge at 0x114b54650>,
           <matplotlib.patches.Wedge at 0x115747a50>,
           <matplotlib.patches.Wedge at 0x114bd3ed0>,
           <matplotlib.patches.Wedge at 0x115899b90>,
           <matplotlib.patches.Wedge at 0x1158999d0>,
           <matplotlib.patches.Wedge at 0x114b5e110>],
          [<matplotlib.text.Text at 0x115783a90>,
           <matplotlib.text.Text at 0x114bd3450>,
           <matplotlib.text.Text at 0x115899790>,
           <matplotlib.text.Text at 0x115899f90>,
           <matplotlib.text.Text at 0x114b5efd0>,
           <matplotlib.text.Text at 0x113efab10>],
          [<matplotlib.text.Text at 0x115783350>,
           <matplotlib.text.Text at 0x114bd34d0>,
           <matplotlib.text.Text at 0x115899b10>,
           <matplotlib.text.Text at 0x115899a90>,
           <matplotlib.text.Text at 0x114b5e6d0>,
           <matplotlib.text.Text at 0x113efa210>])
```

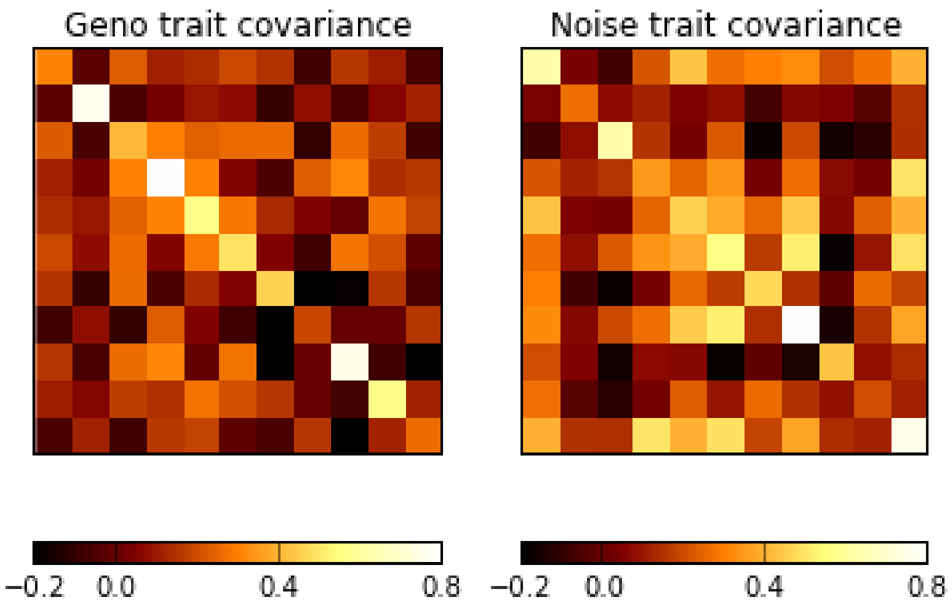


4.1.3 Example 2: fast variance decomposition

In this example we consider a model with only two random effects: one genetic and a noise term and show how to perform fast inference to learn the trait covariance matrices. The mathematical tricks used in LIMIX have also been proposed in [5] and [6]. We apply the model to the expression in the glucose condition and for 11 genes of the lysine biosynthesis pathway.

```
In [18]: #create a complex query on the gene_ID and environment:
         # select environment 0 for all genes in lysine_group
         phenotype_query = "(gene_ID in %s) & (environment==0)" % str(lysine_group)

         data_subsample = data.subsample_phenotypes(phenotype_query=phenotype_query,
                                                    intersection=True)

         #get variables we need from data
         phenotypes,sample_idx = data_subsample.getPhenotypes(phenotype_query=phenotype_query,
                                                              intersection=True)
         assert sample_idx.all()

         sample_relatedness = data_subsample.getCovariance()

         #set parameters for the analysis
         N, G = phenotypes.shape
```

```
        # variance component model
        vc = VAR.VarianceDecomposition(phenotypes.values)
        vc.addFixedEffect()
        vc.addRandomEffect(K=sample_relatedness,trait_covar_type='lowrank_diag',rank=4)
        vc.addRandomEffect(is_noise=True,trait_covar_type='lowrank_diag',rank=4)
        vc.optimize()
        # retrieve geno and noise covariance matrix
        Cg = vc.getTraitCovar(0)
        Cn = vc.getTraitCovar(1)

In [19]: # plot trait covariance matrices
        plt = PL.subplot(1,2,1)
        PL.title('Geno trait covariance')
        PL.imshow(Cg,vmin=-0.2,vmax=0.8,interpolation='none',cmap=cm.afmhot)
        PL.colorbar(ticks=[-0.2,0.,0.4,0.8],orientation='horizontal')
        plt.set_xticks([])
        plt.set_yticks([])
        plt = PL.subplot(1,2,2)
        PL.title('Noise trait covariance')
        PL.imshow(Cn,vmin=-0.2,vmax=0.8,interpolation='none',cmap=cm.afmhot)
        PL.colorbar(ticks=[-0.2,0.,0.4,0.8],orientation='horizontal')
        plt.set_xticks([])
        plt.set_yticks([])

Out[19]: []
```



4.2 Multi Trait GWAS

The genetic model considered by LIMIX for multivariate GWAS consists of fixed effects for covariates, a fixed effect for the SNP being tested, a random effect for the polygenic effect and a noisy random effect term.

The model can be written

$$\mathbf{Y} = \sum_i \mathbf{F}_i \mathbf{W}_i \mathbf{A}_i^{(\mathrm{cov})} + \mathbf{X}\mathbf{B}\mathbf{A}_1^{(\mathrm{snp})} + \mathbf{G} + \mathbf{\Psi}, \quad \mathbf{G} \sim \mathrm{MVN}(\mathbf{0}, \mathbf{K}, \mathbf{C}_g),\ \mathbf{\Psi} \sim \mathrm{MVN}(\mathbf{0}, \mathbf{I}_N, \mathbf{C}_n), \tag{28}$$

where

$$\begin{aligned}
\mathbf{Y} &= \text{matrix-variate phenotype} \in \mathcal{R}^{N,P} && (29)\\
\mathbf{F_i} &= \text{sample design for covariates} \in \mathcal{R}^{N,K} && (30)\\
\mathbf{W_i} &= \text{effect size matrix of covariates} \in \mathcal{R}^{K,L} && (31)\\
\mathbf{A_i}^{(\mathrm{cov})} &= \text{trait design for the fixed effect} \in \mathcal{R}^{L,P} && (32)\\
\mathbf{X} &= \text{genetic profile of the SNP} \in \mathcal{R}^{N,1} && (33)\\
\mathbf{B} &= \text{effect sizes of the SNP} \in \mathcal{R}^{1,M} && (34)\\
\mathbf{A}_1^{(\mathrm{snp})} &= \text{trait design for the fixed effect} \in \mathcal{R}^{M,P} && (35)\\
\mathbf{K} &= \text{sample relatedeness matrix} \in \mathcal{R}^{N,N} && (36)\\
\mathbf{C}_g &= \text{genetic trait covariance matrix} \in \mathcal{R}^{P,P} && (37)\\
\mathbf{I}_N &= \text{sample noise matrix} \in \mathcal{R}^{N,N} && (38)\\
\mathbf{C}_n &= \text{noise trait covariance matrix} \in \mathcal{R}^{P,P} && (39)
\end{aligned}$$

Prior to any multi-trait GWAS analysis, LIMIX learns the trait covariates matrices on the mixed model without the fixed effect from the SNP. However, LIMIX allows the user to set refitting of the signal-to-noise ratio δ SNP-by-SNP during the GWAS as in single-trait analysis.

In general, LIMIX tests for particular trait designs of the fixed effect $\mathbf{A}_1^{(\mathrm{snp})} \neq \mathbf{A}_0^{(\mathrm{snp})}$. As shown extensively below, specifying opportunately $\mathbf{A}_1^{(\mathrm{snp})}$ and $\mathbf{A}_0^{(\mathrm{snp})}$ different biological hypothesis can be tested.

An alternative efficient implement of multi trait GWAS is available in GEMMA [6], however LIMIX permits more flexible models to be constructed and enables automated regularization of trait covariances (see below).

4.2.1 Any effect test

Association between any of the phenotypes and the genetic marker can be tested by setting

$$\mathbf{A}_1^{(\mathrm{snp})} = \mathbf{I}_P, \quad \mathbf{A}_0^{(\mathrm{snp})} = \mathbf{0} \tag{40}$$

This is a P degrees of freedom test.

In this context, multi-trait modelling is considered just to empower detection of genetic loci while there is no interest in the specific design of the association.

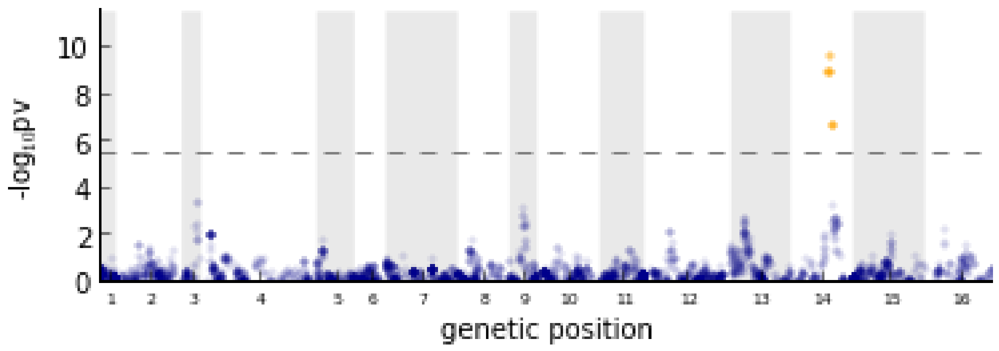
Example: any effect test for gene YJR139C across environments Here we test for genetic effects on expression of one gene either in environment 0 or in environment 1.

```
In [20]: # Get data
         # select all environmens for gene YJR139C in lysine_group
         phenotype_query = "(gene_ID == 'YJR139C')"

         data_subsample = data.subsample_phenotypes(phenotype_query=phenotype_query,
                                                    intersection=True)

         #get variables we need from data
         snps = data_subsample.getGenotypes(impute_missing=True)
         phenotypes,sample_idx = data_subsample.getPhenotypes(phenotype_query=phenotype_query,
                                                              intersection=True)
         assert sample_idx.all()

         sample_relatedness = data_subsample.getCovariance()

         #set parameters for the analysis
         N, P = phenotypes.shape

         covs = None                     #covariates
         Acovs = None                    #the design matrix for the covariates
         Asnps = SP.eye(P)               #the design matrix for the SNPs
         K1r = sample_relatedness        #the first sample-sample covariance matrix (non-noise)
         K2r = SP.eye(N)                 #the second sample-sample covariance matrix (noise)
         K1c = None                      #the first phenotype-phenotype covariance matrix (non-noise)
         K2c = None                      #the second phenotype-phenotype covariance matrix (noise)
         covar_type = 'freeform'         #the type of the trait/trait covariance to be estimated
         searchDelta = False             #specify if delta should be optimized for each SNP
         test="lrt"                      #specify type of statistical test

         # Running the analysis
         # when cov are not set (None), LIMIX considers an intercept (covs=SP.ones((N,1)))
         lmm, pvalues = QTL.test_lmm_kronecker(snps,phenotypes.values,covs=covs,Acovs=Acovs,
                                               Asnps=Asnps,K1r=K1r,trait_covar_type=covar_type)

         #convert P-values to a DataFrame for nice output writing:
         pvalues = pd.DataFrame(data=pvalues.T,index=data_subsample.geno_ID,columns=['YJR139C'])
         pvalues = pd.concat([position,pvalues],join="outer",axis=1)
```

```
.. Training the backgrond covariance with a GP model
Background model trained in 0.06 s
```

REMARKS: - kronecker_lmm always considers Asnp0=0 thus only Asnp1 needs to be specified - if trait covariance matrices (arguments K1c and K2c are not specified) kronecker_lmm implements the variance decomposition module to estimate them. In this case, the user can specify the trait covariance model to considered (covar_type and rank).

```
In [21]: plt = PL.subplot(2,1,1)
         plot_manhattan(plt,pvalues['pos_cum'],pvalues['YJR139C'],chromBounds)
```



4.2.2 Common effect test

A common effect test is a 1 degree of freedom test and can be done by setting

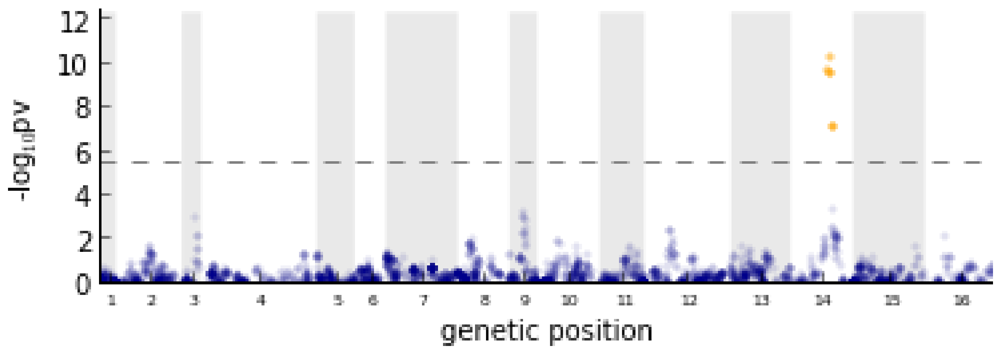
$$\mathbf{A}_1^{(\mathrm{snp})} = \mathbf{1}_{1,P}, \quad \mathbf{A}_0^{(\mathrm{snp})} = \mathbf{0} \tag{41}$$

Example: common effect test for gene YJR139C across environments

```
In [22]: # Get data
         # select all environmens for gene YJR139C in lysine_group
         phenotype_query = "(gene_ID == 'YJR139C')"

         data_subsample = data.subsample_phenotypes(phenotype_query=phenotype_query,
                                                    intersection=True)

         #get variables we need from data
         snps = data_subsample.getGenotypes(impute_missing=True)
         phenotypes,sample_idx = data_subsample.getPhenotypes(phenotype_query=phenotype_query,
                                                              intersection=True)
         assert sample_idx.all()

         sample_relatedness = data_subsample.getCovariance()

         #set parameters for the analysis
         N, P = phenotypes.shape

         covs = None                  #covariates
         Acovs = None                 #the design matrix for the covariates
         Asnps = SP.ones((1,P))       #the design matrix for the SNPs
         K1r = sample_relatedness     #the first sample-sample covariance matrix (non-noise)
         K2r = SP.eye(N)              #the second sample-sample covariance matrix (noise)
         K1c = None                   #the first phenotype-phenotype covariance matrix (non-noise)
         K2c = None                   #the second phenotype-phenotype covariance matrix (noise)
         covar_type = 'freeform'      #the type of the trait/trait covariance to be estimated
         searchDelta = False          #specify if delta should be optimized for each SNP
         test="lrt"                   #specify type of statistical test

         # Running the analysis
         # when cov are not set (None), LIMIX considers an intercept (covs=SP.ones((N,1)))
         lmm, pvalues = QTL.test_lmm_kronecker(snps,phenotypes.values,covs=covs,Acovs=Acovs,
                                               Asnps=Asnps,K1r=K1r,trait_covar_type=covar_type)

         #convert P-values to a DataFrame for nice output writing:
         pvalues = pd.DataFrame(data=pvalues.T,index=data_subsample.geno_ID,columns=['YJR139C'])
         pvalues = pd.concat([position,pvalues],join="outer",axis=1)
```

```
.. Training the backgrond covariance with a GP model
Background model trained in 0.05 s
```

```
In [23]: plt = PL.subplot(2,1,1)
         plot_manhattan(plt,pvalues['pos_cum'],pvalues['YJR139C'],chromBounds)
```



4.2.3 Specific effect test

For a specifc effect test for trait p the alternative model is set to have both a common and a specific effect for transcript p from the SNP while the null model has only a common effect.

It is a 1 degree of freedom test and, in the particular case of $P = 3$ traits and for $p = 0$, it can be done by setting

$$\mathbf{A}_1^{(\text{snp})} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \quad , \mathbf{A}_0^{(\text{snp})} = \mathbf{1}_{1,3} \tag{42}$$

Example: specific effect test for gene YJR139C across environmentsExample

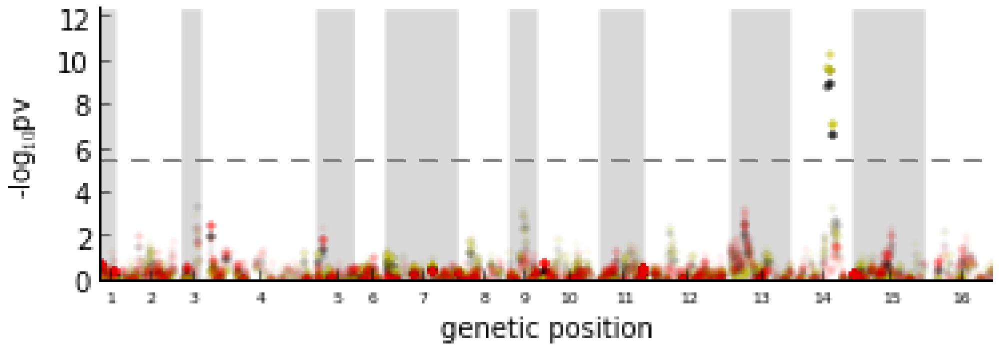
```
In [24]: # select all environmens for gene YJR139C in lysine_group
         phenotype_query = "(gene_ID == 'YJR139C')"

         data_subsample = data.subsample_phenotypes(phenotype_query=phenotype_query,
                                                    intersection=True)

         #get variables we need from data
         snps = data_subsample.getGenotypes(impute_missing=True)
         phenotypes,sample_idx = data_subsample.getPhenotypes(phenotype_query=phenotype_query,
                                                              intersection=True)
         assert sample_idx.all()

         sample_relatedness = data_subsample.getCovariance()

         #set parameters for the analysis
         N, P = phenotypes.shape

         covs = None                #covariates
         Acovs = None               #the design matrix for the covariates
         Asnps0 = SP.ones((1,P))    #the null model design matrix for the SNPs
         Asnps1 = SP.zeros((2,P))   #the alternative model design matrix for the SNPs
         Asnps1[0,:] = 1.0
         Asnps1[1,0] = 1.0
         K1r = sample_relatedness   #the first sample-sample covariance matrix (non-noise)
         K2r = SP.eye(N)            #the second sample-sample covariance matrix (noise)
         K1c = None                 #the first phenotype-phenotype covariance matrix (non-noise)
         K2c = None                 #the second phenotype-phenotype covariance matrix (noise)
```

```
covar_type = 'freeform'     #the type of the trait/trait covariance to be estimated
searchDelta = False         #specify if delta should be optimized for each SNP
test="lrt"                  #specify type of statistical test

# Running the analysis
# when cov are not set (None), LIMIX considers an intercept (covs=SP.ones((N,1)))
pvalues = QTL.test_interaction_lmm_kronecker(snps=snps,phenos=phenotypes.values,
    covs=covs,Acovs=Acovs,Asnps1=Asnps1,Asnps0=Asnps0,K1r=K1r,K2r=K2r,K1c=K1c,K2c=K2c,
    trait_covar_type=covar_type,searchDelta=searchDelta)


#convert P-values to a DataFrame for nice output writing:
pvalues = pd.DataFrame(data=SP.concatenate(pvalues).T,index=data_subsample.geno_ID,
                       columns=["specific","null_common","alternative_any"])
pvalues = pd.concat([position,pvalues],join="outer",axis=1)
```

```
.. Training the backgrond covariance with a GP model
Background model trained in 0.06 s
```

Simple_interaction_kronecker not only compares the alternative model (where the trait design of the SNP is Asnp=Asnp1) versus the null model (where the trait design of the SNP is Asnp=Asnp0) but also compares the alternative model and the null models versus the no association model (Asnp=0). Three pvalues are then retured: - pv for alternative vs null (specific effect test) - pv for null vs noAssociation (common effect test) - pv for alternative vs noAssociation (any effect test)

```
In [25]: tests = ['Any effect test','Interaction effect test']
         lim = -1.2*SP.log10(SP.array([pvalues['alternative_any'].min(),
                 pvalues['null_common'].min(),pvalues['specific'].min()]).min())
         plt = PL.subplot(2,1,1)
         plot_manhattan(plt,pvalues['pos_cum'],pvalues['alternative_any'],
                        chromBounds,colorS='k',colorNS='k',lim=lim,alphaNS=0.05)
         plot_manhattan(plt,pvalues['pos_cum'],pvalues['null_common'],
                        chromBounds,colorS='y',colorNS='y',lim=lim,alphaNS=0.05)
         plot_manhattan(plt,pvalues['pos_cum'],pvalues['specific'],
                        chromBounds,colorS='r',colorNS='r',lim=lim,alphaNS=0.05)
```



4.2.4 Example: any, common and specific effect tests across environments for all genes in the pathway

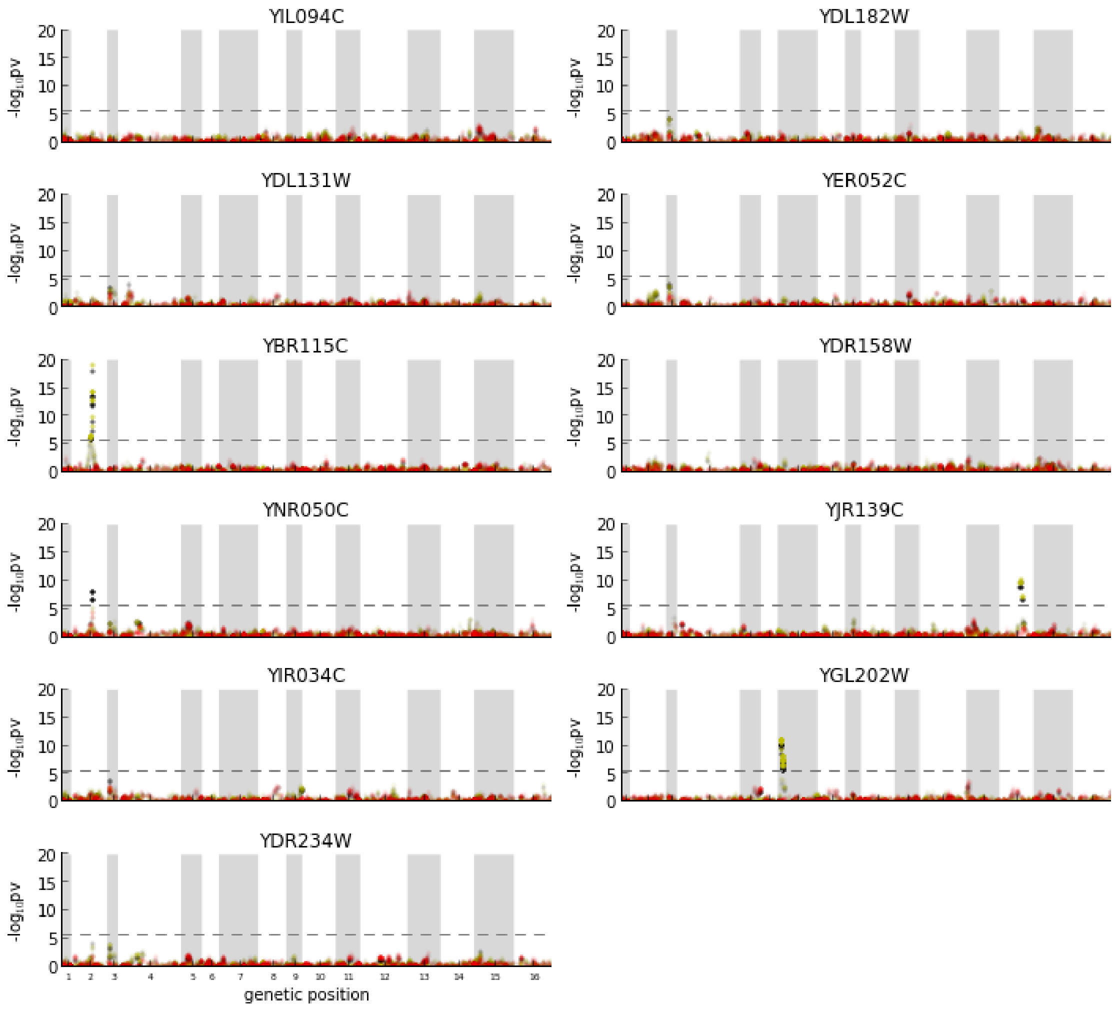
```
In [26]: pvalues_dict={}
```

```
    for gene_idx,gene in enumerate(lysine_group):
        # select all environmens for gene YJR139C in lysine_group
        phenotype_query = "(gene_ID == '%s')" % gene
        print "running GxE GWAS tests on %s" % phenotype_query
        data_subsample = data.subsample_phenotypes(phenotype_query=phenotype_query,
                                                   intersection=True)

        #get variables we need from data
        phenotypes,sample_idx = data_subsample.getPhenotypes(phenotype_query=phenotype_query,
                                                             intersection=True)
        assert sample_idx.all()
        if phenotypes.shape[1]==0:
            continue

        snps = data_subsample.getGenotypes(impute_missing=True)

        sample_relatedness = data_subsample.getCovariance()

        #set parameters for the analysis
        N, P = phenotypes.shape

        covs = None                    #covariates
        Acovs = None                   #the design matrix for the covariates
        Asnps0 = SP.ones((1,P))        #the null model design matrix for the SNPs
        Asnps1 = SP.zeros((2,P))       #the alternative model design matrix for the SNPs
        Asnps1[0,:] = 1.0
        Asnps1[1,0] = 1.0
        K1r = sample_relatedness       #the first sample-sample covariance matrix (non-noise)
        K2r = SP.eye(N)                #the second sample-sample covariance matrix (noise)
        K1c = None                     #the first phenotype-phenotype covariance matrix (non-noise)
        K2c = None                     #the second phenotype-phenotype covariance matrix (noise)
        covar_type = 'freeform'        #the type of the trait/trait covariance to be estimated
        searchDelta = False            #specify if delta should be optimized for each SNP
        test="lrt"                     #specify type of statistical test

        # Running the analysis
        # when cov are not set (None), LIMIX considers an intercept (covs=SP.ones((N,1)))
        pvalues = QTL.test_interaction_lmm_kronecker(snps=snps,phenos=phenotypes.values,
            covs=covs,Acovs=Acovs,Asnps1=Asnps1,Asnps0=Asnps0,K1r=K1r,K2r=K2r,K1c=K1c,
            K2c=K2c,trait_covar_type=covar_type,searchDelta=searchDelta)

        #convert P-values to a DataFrame for nice output writing:
        pvalues = pd.DataFrame(data=SP.concatenate(pvalues).T,index=data_subsample.geno_ID,
                               columns=["specific","null_common","alternative_any"])
        pvalues = pd.concat([position,pvalues],join="outer",axis=1)
        key = "pvalues_%s" % gene
        pvalues_dict[key] = pvalues
```

```
running GxE GWAS tests on (gene_ID == 'YIL094C')
.. Training the backgrond covariance with a GP model
Background model trained in 0.05 s
running GxE GWAS tests on (gene_ID == 'YDL182W')
.. Training the backgrond covariance with a GP model
Background model trained in 0.05 s
```

```
running GxE GWAS tests on (gene_ID == 'YDL131W')
.. Training the backgrond covariance with a GP model
Background model trained in 0.06 s
running GxE GWAS tests on (gene_ID == 'YER052C')
.. Training the backgrond covariance with a GP model
Background model trained in 0.05 s
running GxE GWAS tests on (gene_ID == 'YBR115C')
.. Training the backgrond covariance with a GP model
Background model trained in 0.09 s
running GxE GWAS tests on (gene_ID == 'YDR158W')
.. Training the backgrond covariance with a GP model
Background model trained in 0.05 s
running GxE GWAS tests on (gene_ID == 'YNR050C')
.. Training the backgrond covariance with a GP model
Background model trained in 0.05 s
running GxE GWAS tests on (gene_ID == 'YJR139C')
.. Training the backgrond covariance with a GP model
Background model trained in 0.05 s
running GxE GWAS tests on (gene_ID == 'YIR034C')
.. Training the backgrond covariance with a GP model
Background model trained in 0.05 s
running GxE GWAS tests on (gene_ID == 'YGL202W')
.. Training the backgrond covariance with a GP model
Background model trained in 0.05 s
running GxE GWAS tests on (gene_ID == 'YDR234W')
.. Training the backgrond covariance with a GP model
Background model trained in 0.06 s
```

```
In [27]: # plotting all manhattan plots
         PL.figure(1,figsize=(10,16))
         lim = 20
         for g in range(G):
             plt = PL.subplot(G,2,g+1)
             PL.title(lysine_group[g])
             if g!=G-1:   xticklabels = False
             else:        xticklabels = True
             pv = pvalues_dict['pvalues_%s'%lysine_group[g]]
             plot_manhattan(plt,pv['pos_cum'],pv['alternative_any'],
                 chromBounds,colorS='k',colorNS='k',lim=lim,alphaNS=0.05,xticklabels=xticklabels)
             plot_manhattan(plt,pv['pos_cum'],pv['null_common'],
                 chromBounds,colorS='y',colorNS='y',lim=lim,alphaNS=0.05,xticklabels=xticklabels)
             plot_manhattan(plt,pv['pos_cum'],pv['specific'],
                 chromBounds,colorS='r',colorNS='r',lim=lim,alphaNS=0.05,xticklabels=xticklabels)
         PL.tight_layout()
```



All any-effect eQTL are explained by environmental persistent signal with the exception of the peak on chromosome 2 for gene YNR050C that has an almost significant specific component. These results together with those obtained from the GxE variance decomposition analysis suggest that the strong transGxE signal of the genes in the pathway is due to interaction of the poligenic effect with the environment.

4.3 Multi trait Multi locus models

```
In [27]:
```

5 Predictions and Model Selection

In this section we discuss how to use the phenotype prediction tool in the LIMIX variance decomposition module to monitor overfitting and perform model selection. We start by discussing how to impose regularization on the trait covariance matrices and how to make phenotype predicitons. Finally we show how to use cross validation to select the extent of the penalization.

5.1 Regularized maximum likelihood

When considering high number of traits, flexible models for trait covariance matrices are characterized by a large number of parameters and may lead to overfitting. For example, general semi-definite positive matrices (*freeform* matrices) are parameterized by $\frac{1}{2}P(P+1)$ real values, with the number of parameters increasing quadratically in the number of traits while the data only linearly.

In order to prevent this, LIMIX makes available to the user less complex covariance models (e.g. block and low rank models) as well as the possibility to add a regularization term over the trait covariance matrices

$$\min_{\boldsymbol{\theta}} \underbrace{-\log \mathcal{N}\left(\mathrm{vec}\mathbf{Y} \mid \sum_i \mathbf{F}_i \mathbf{W}_i \mathbf{A}_i,\ \mathbf{C}_g(\boldsymbol{\theta}) \otimes \mathbf{R} + \mathbf{C}_n(\boldsymbol{\theta}) \otimes \mathbf{I}\right)}_{\text{data fit}} + \underbrace{\lambda \sum_{i \neq j} \left(\mathbf{C}_g(\boldsymbol{\theta})_{ij}{}^2 + \mathbf{C}_n(\boldsymbol{\theta})_{ij}{}^2\right)}_{\text{regularizer}} \tag{43}$$

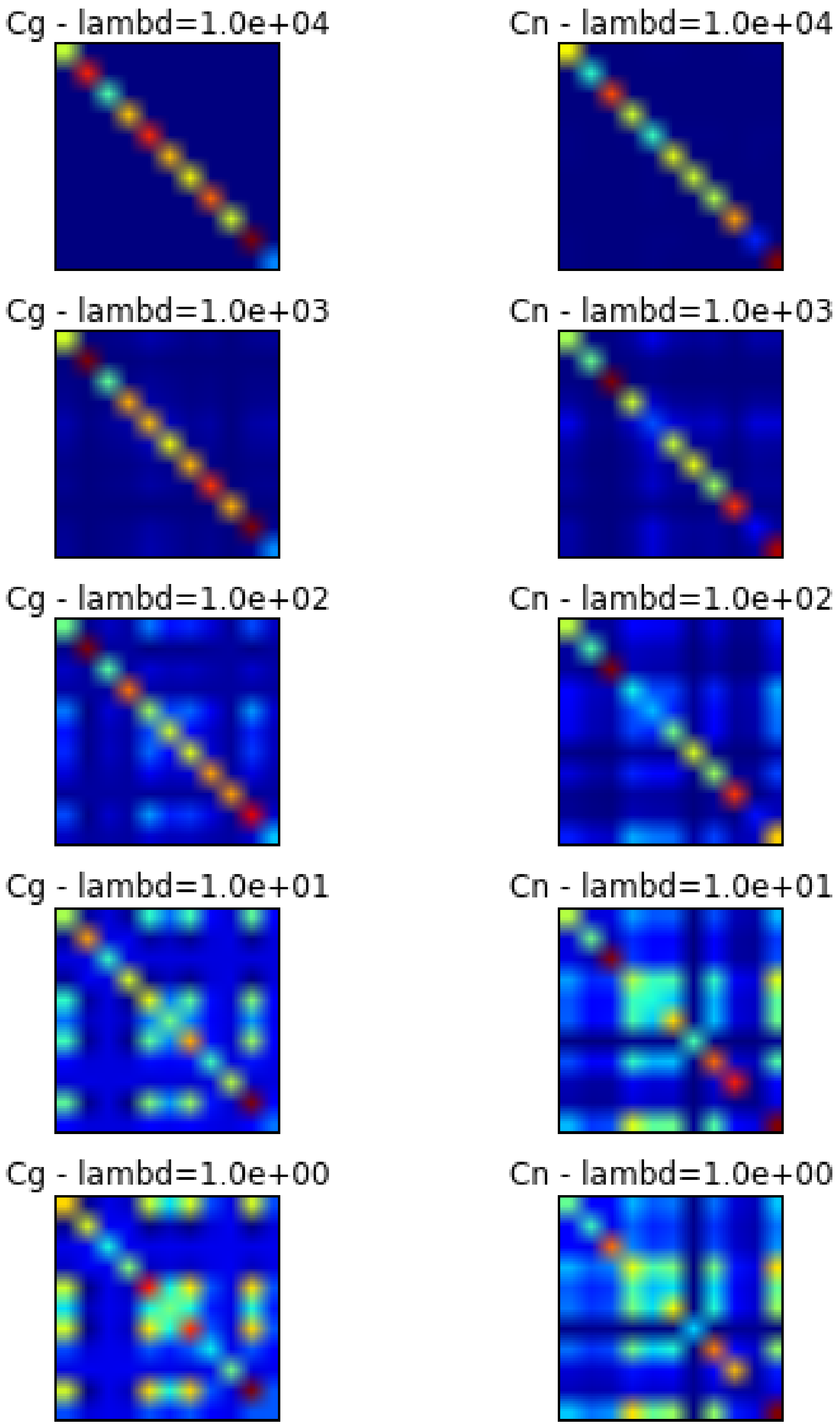
where $\otimes$ denotes the Kronecker product and λ is the extent of the penalization. Notice that the penalization is only applied to off-diagonal elements to preserve unbiased marginal heritability estimates.

5.1.1 Example: setting regularization

```
In [28]: #create a complex query on the gene_ID and environment:
         # select environment 0 for all genes in lysine_group
         phenotype_query = "(gene_ID in %s) & (environment==0)" % str(lysine_group)

         data_subsample = data.subsample_phenotypes(phenotype_query=phenotype_query,
                                                    intersection=True)

         #get variables we need from data
         phenotypes,sample_idx = data_subsample.getPhenotypes(phenotype_query=phenotype_query,
                                                              intersection=True)
         assert sample_idx.all()

         sample_relatedness = data_subsample.getCovariance()

         # variance component model
         vc = VAR.VarianceDecomposition(phenotypes.values)
         vc.addFixedEffect()
         vc.addRandomEffect(K=sample_relatedness,trait_covar_type='lowrank_diag')
         vc.addRandomEffect(is_noise=True,trait_covar_type='lowrank_diag')

         # setting different extent of panalization
         lambds = [1e4, # very highly penalized model
                   1e3,1e2,1e1,
                   1e0 # lowly penalized model
                  ]
         Cg = []
         Cn = []
         for lambd in lambds:
             print "optimizing model with lambda=%.1e" % lambd
             # fit the model with extent of penalization lambd
             vc.optimize(lambd=lambd)
             # store trait covariance matrices
             Cg.append(vc.getTraitCovar(0))
             Cn.append(vc.getTraitCovar(1))
```

```
optimizing model with lambda=1.0e+04
optimizing model with lambda=1.0e+03
optimizing model with lambda=1.0e+02
optimizing model with lambda=1.0e+01
optimizing model with lambda=1.0e+00
```

```
In [29]: PL.figure(1,figsize=(6,8))
         for i in range(len(lambds)):
             plt = PL.subplot(len(lambds),2,2*i+1)
             PL.title('Cg - lambd=%.1e'%lambds[i])
             PL.imshow(Cg[i])
             plt.set_xticks([])
             plt.set_yticks([])
             plt = PL.subplot(len(lambds),2,2*i+2)
             PL.title('Cn - lambd=%.1e'%lambds[i])
             PL.imshow(Cn[i])
             plt.set_xticks([])
             plt.set_yticks([])
         PL.tight_layout()
```


Cg - lambd=1.0e+04
Cn - lambd=1.0e+04
Cg - lambd=1.0e+03
Cn - lambd=1.0e+03
Cg - lambd=1.0e+02
Cn - lambd=1.0e+02
Cg - lambd=1.0e+01
Cn - lambd=1.0e+01
Cg - lambd=1.0e+00
Cn - lambd=1.0e+00

5.2 Phenotype predictions

Let N denote the number of individuals for which we have both phenotypes $\mathbf{Y} \in \mathcal{R}^{N,P}$ and genotypes $\mathbf{X} \in \mathcal{R}^{N,S}$ and $N^\star$ denote the number of individuals for which we have genotypes $\mathbf{X}^\star \in \mathcal{R}^{N^\star,S}$ and we want to predict phenotypic values. Moreover, let us introduce

$$\begin{aligned}
\mathbf{F_i} &= \text{fixed effect sample design for the } N \text{ samples} \in \mathcal{R}^{N,K} && (44)\\
\mathbf{F_i^\star} &= \text{fixed effect sample design for the } N^\star \text{ samples} \in \mathcal{R}^{N^\star,K} && (45)\\
\mathbf{R} &= \text{sample relatedeness matrix} = \mathbf{X}\mathbf{X}^T \in \mathcal{R}^{N,N} && (46)\\
\mathbf{R}^\star &= \text{cross relatedeness matrix} = \mathbf{X}\mathbf{X}^{\star T} \in \mathcal{R}^{N,N^\star} && (47)
\end{aligned}$$

The LIMIX variance decomposition model implements best linear unbiased prediction (BLUP) where $\mathbf{F}_i$, $\mathbf{R}$ and $\mathbf{Y}$ are used to estimate the model parameters while $\mathbf{F}_i^\star$, $\mathbf{R}^\star$ and $\mathbf{Y}^\star$ are used for predicticting:

$$\mathbf{Y}^\star = \sum_i \mathbf{F}_i^\star \mathbf{W}_i \mathbf{A}_i + \mathbf{R}^{\star T}\mathbf{Z}\mathbf{C} \quad (48)$$

where

$$\begin{aligned}
\text{vec}\mathbf{Z} &= \mathbf{K}^{-1}\text{vec}\left(\mathbf{Y} - \sum_i \mathbf{F}_i \mathbf{W}_i \mathbf{A}_i\right) && (49)\\
\mathbf{K} &= \mathbf{C}_g \otimes \mathbf{R} + \mathbf{C}_n \otimes \mathbf{I} && (50)
\end{aligned}$$

While here we show only the case with two random effect, a polygenic random effect and a noise random effect, the implementation in LIMIX can be used for any number of random effects.

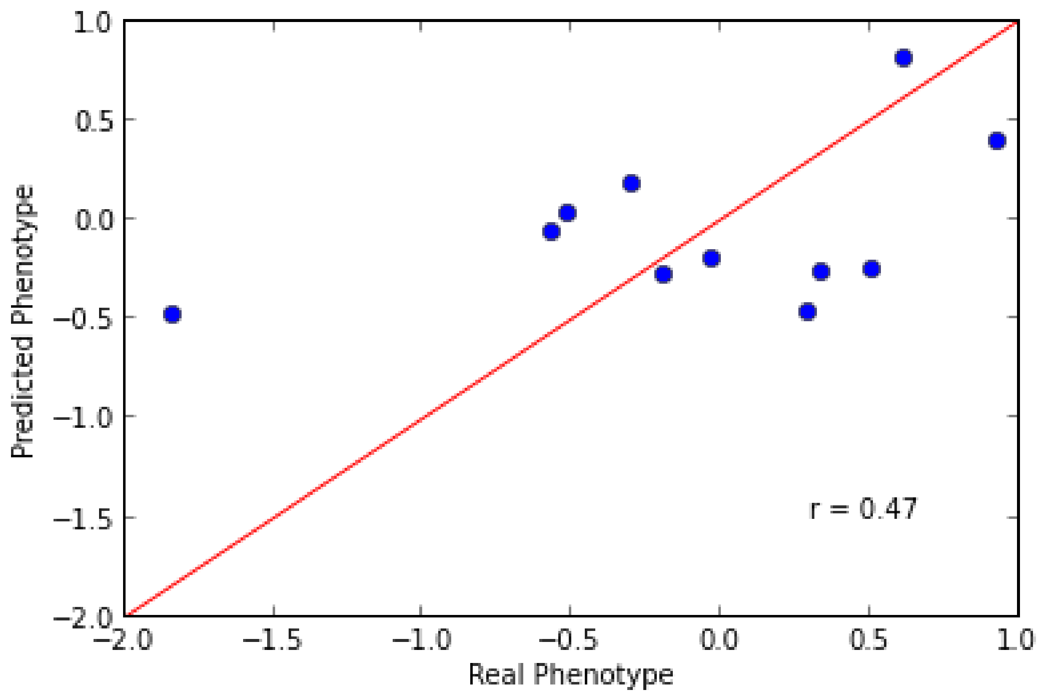
5.2.1 Example: predictions for gene YJR139C

In this example we predict gene expression from genotype for gene YJR139C. We split the samples in two parts: - training samples (˜90% of the samples), whose phenotype and genotype values are used to train the model - test samples (˜10% of the samples), whose genotypes are used for predictions.

Predicted phenotypes for test samples are then compared with the corresponding measured phenotypes. The code for the multi-trait case is identical.

To use the prediction method in the variance decomposition module, it is sufficient to specify for each fixed and random effect to use for predictions the test sample design $\mathbf{F}^\star$ and the cross covariance $\mathbf{R}^\star$ respectively (in addition to $\mathbf{F}$ and $\mathbf{R}$). Predicting using only part of the fixed and random effect terms is also possible.

```
In [30]: #create a complex query on the gene_ID and environment:
         # select environment 0 for gene YJR139C
         phenotype_query = "(gene_ID=='YJR139C') & (environment==0)"

         data_subsample = data.subsample_phenotypes(phenotype_query=phenotype_query,
                                                    intersection=True)

         #get variables we need from data
         phenotypes,sample_idx = data_subsample.getPhenotypes(phenotype_query=phenotype_query,
                                                              intersection=True)
         assert sample_idx.all()

         sample_relatedness = data_subsample.getCovariance()

         # split samples into training and test
         SP.random.seed(0)
         r   = SP.random.permutation(phenotypes.shape[0])
         nfolds = 10
         Icv = SP.floor(((SP.ones((phenotypes.shape[0]))*nfolds)*r)/phenotypes.shape[0])
         Itrain  = Icv!=0
         Itest   = Icv==0
         pheno_train = phenotypes.values[Itrain,:]
         pheno_test  = phenotypes.values[Itest,:]

         # variance component model
         vc = VAR.VarianceDecomposition(pheno_train)
         vc.setTestSampleSize(Itest.sum())
         #1. add intercept
         F     = SP.ones([pheno_train.shape[0],1])
         Ftest = SP.ones([pheno_test.shape[0],1])
         vc.addFixedEffect(F=F,Ftest=Ftest)
         #2. add polygenic random effect
         Rtrain = sample_relatedness[Itrain,:][:,Itrain]
         Rcross = sample_relatedness[Itrain,:][:,Itest]
         vc.addRandomEffect(K=Rtrain,Kcross=Rcross)
         #3. noise
         vc.addRandomEffect(is_noise=True)
         # train the model on the test
         vc.optimize()
         # predict on the training
         predicted_phenos = vc.predictPhenos()

In [31]: # plot predicted versus real phenotypes
         rho = SP.corrcoef(predicted_phenos.T,pheno_test.T)[0,1]
         PL.plot(pheno_test.ravel(),predicted_phenos.ravel(),'o')
         PL.xlabel('Real Phenotype')
         PL.ylabel('Predicted Phenotype')
         PL.plot([-2,1],[-2,1],'r')
         PL.xlim(-2,1)
         PL.ylim(-2,1)
         PL.text(0.3,-1.5,'r = %.2f'%(rho))

Out[31]: <matplotlib.text.Text at 0x11583fa10>
```



5.3 Cross-Validation Model and Selection

LIMIX variance decomposition has a method for cross-validation that builds on the prediction methods we have just showcased. The entire dataset is split into n folds, each fold is predicted independentely after training the model on the remaining $n-1$ so that the whole matrix of predicted phenotypes can be obtained. This can be used to perform model selection across different genetic models.

In the following we show how the cross-validation method can be used to select the extent of the penalization on the trait covariance matrices.

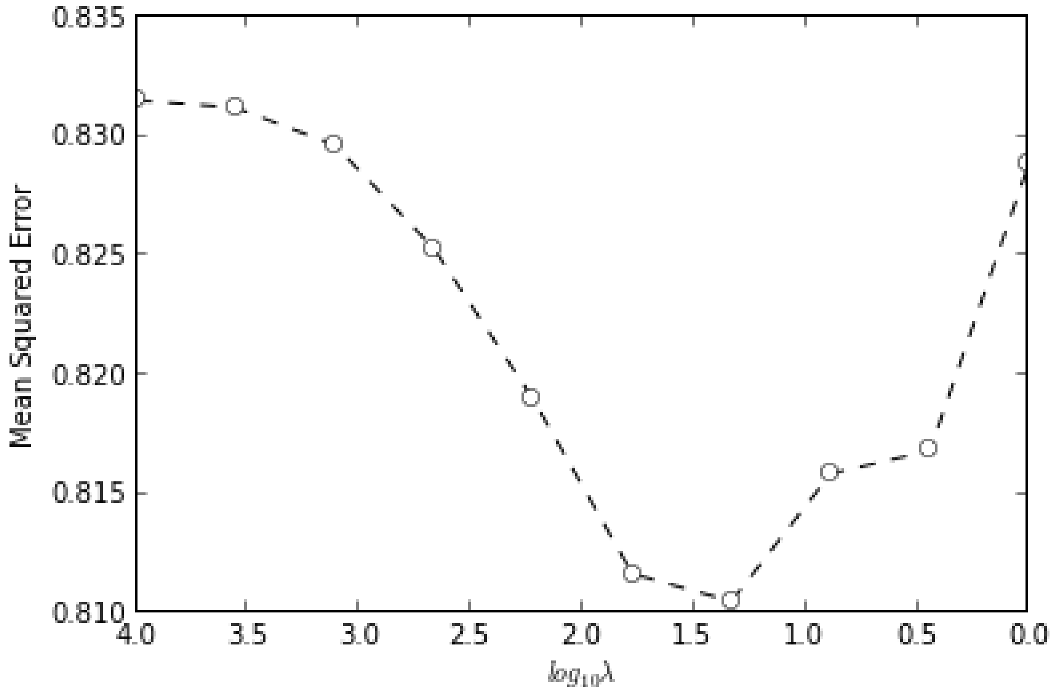
5.3.1 Example: selecting the extend of the penalization

```
In [32]: #create a complex query on the gene_ID and environment:
         # select environment 0 for all genes in lysine_group
         phenotype_query = "(gene_ID in %s) & (environment==0)" % str(lysine_group)

         data_subsample = data.subsample_phenotypes(phenotype_query=phenotype_query,
                                                    intersection=True)

         #get variables we need from data
         phenotypes,sample_idx = data_subsample.getPhenotypes(phenotype_query=phenotype_query,
                                                              intersection=True)
         assert sample_idx.all()
         N,P=phenotypes.shape

         sample_relatedness = data_subsample.getCovariance()

         # variance component model
         vc = VAR.VarianceDecomposition(phenotypes.values)
         vc.addFixedEffect()
         vc.addRandomEffect(K=sample_relatedness,trait_covar_type='lowrank_diag',rank=2)
         vc.addRandomEffect(is_noise=True,trait_covar_type='lowrank_diag',rank=2)

         # values of lambda to be considered
         lambds = 10**SP.linspace(4,0,10)

         # options for cross validation
         seed = 0
         n_folds = 10

         # cross validation for different values of lambda
         MSE = []
         for lambd in lambds:
             print "Optimizing model with lambda:", lambd
             predicted_phenos = vc.crossValidation(seed=seed,n_folds=n_folds,
                                                   lambd=lambd,verbose=False)
             mse = ((predicted_phenos-phenotypes.values)**2).sum()/float(N*P)
             MSE.append(mse)
```

```
Optimizing model with lambda: 10000.0
Optimizing model with lambda: 3593.8136638
Optimizing model with lambda: 1291.54966501
Optimizing model with lambda: 464.158883361
Optimizing model with lambda: 166.81005372
Optimizing model with lambda: 59.9484250319
Optimizing model with lambda: 21.5443469003
Optimizing model with lambda: 7.74263682681
Optimizing model with lambda: 2.78255940221
Optimizing model with lambda: 1.0
```

```
In [33]: plt = PL.subplot(1,1,1)
         PL.plot(SP.log10(lambds),MSE,'--',color='k')
         PL.plot(SP.log10(lambds),MSE,'o',color='white')
         plt.invert_xaxis()
         PL.xlabel('$log_{10}\lambda$')
         PL.ylabel('Mean Squared Error')
```

```
Out[33]: <matplotlib.text.Text at 0x113f0e290>
```



6 Accounting for gene expression heterogeneity

LIMIX provides implementation for preivously proposed mixed models to estimate gene expression heterogeneity due to unobserved confounders (e.g. [7] & [8]).

LIMIX estimates a sample-by-sample covariance matrix that accounts for both genetic and non-genetic confounders. This approach is applicable in studies with high-dimensional molecular phenotypes such as gene expression levels. In the following we briefly introduce a generic model for expresison heterogeneity that is closely related to the PANAMA approach [8] and show how this can be used for eQTL analyses.

6.1 The Genetic Model

Sarting form a generative model that includes both hidden and known covaraites, we start with the a generative model of the form

$$\mathbf{y}_p = \mathbf{u} + \boldsymbol{\eta} + \boldsymbol{\psi}, \quad \forall p \tag{51}$$

$$\mathbf{u} \sim \mathcal{N}\left(\mathbf{0},\ \sigma_g^2 \mathbf{R}_g\right),\ \boldsymbol{\eta} \sim \mathcal{N}\left(\mathbf{0},\ \mathbf{R}_c\left(\boldsymbol{\alpha}\right)\right),\ \boldsymbol{\psi} \sim \mathcal{N}\left(\mathbf{0},\ \sigma_e^2 \mathbf{I}\right)$$

where we assume independence of the traits conditioned on the latent factors and we have introduced

$$\mathbf{R}_g = \text{sample genetic relatedeness matrix} = \mathbf{X}\mathbf{X}^T \in \mathcal{R}^{N,N} \tag{52}$$

$$\mathbf{R}_c = \text{inferred sample relatedeness matrix due to unobserved covariates} \in \mathcal{R}^{N,N} \tag{53}$$

The parameters of the model are $\left\{\boldsymbol{\alpha}, \sigma_g^2, \sigma_e^2\right\}$. As full rank matrices might overfit the data explaining away genetic signal, LIMIX models $\mathbf{R}_c\left(\boldsymbol{\alpha}\right)$ as rank r matrix and allows the user to choose r. A sensible value for r can be chosen by looking at the number of PCAs of the sample-by-sample empirical covariance matrix explaining 70%-90% of the variance; see also discussion in [**?**]

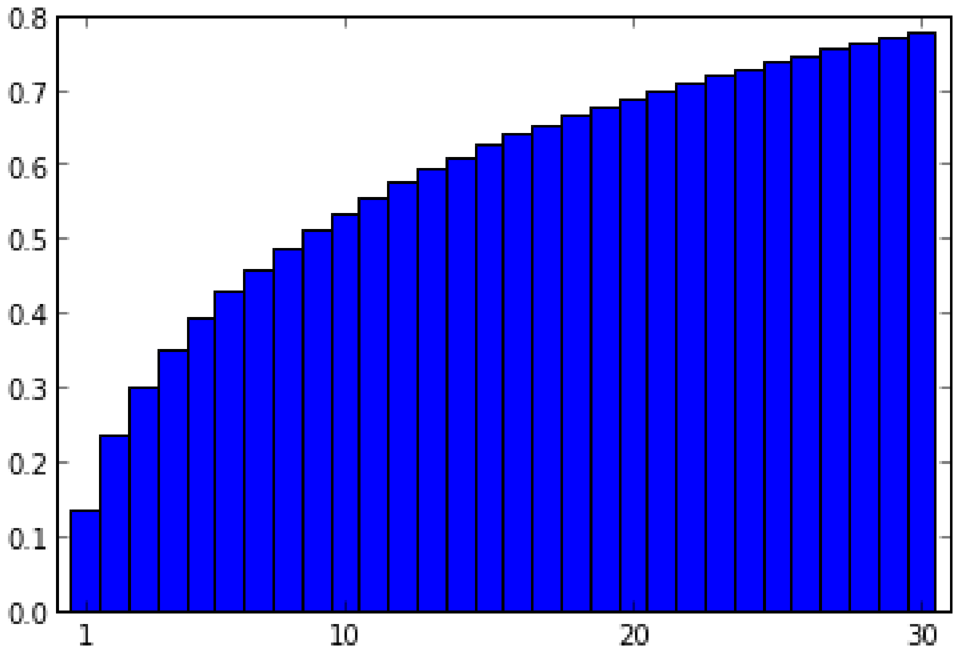
Once $\mathbf{R}_c$ is learned, the new sample relatedeness matrix $\sigma_g^2\mathbf{R}_g + \mathbf{R}_c$ can be plugged in the GWAS and variance decomposition tools in LIMIX both for single and multivariate analysis. In the following we will refer to $\mathbf{R}_c$ (after normalizing) as *hterogeneity matrix* while we will refer to the new sample relatedeness matrix $\sigma_g^2\mathbf{R}_g + \mathbf{R}_c$ (after normalizing) as *total sample covariance matrix.*

6.2 Example: Accounting for gene expression heterogeneity in the yeast dataset

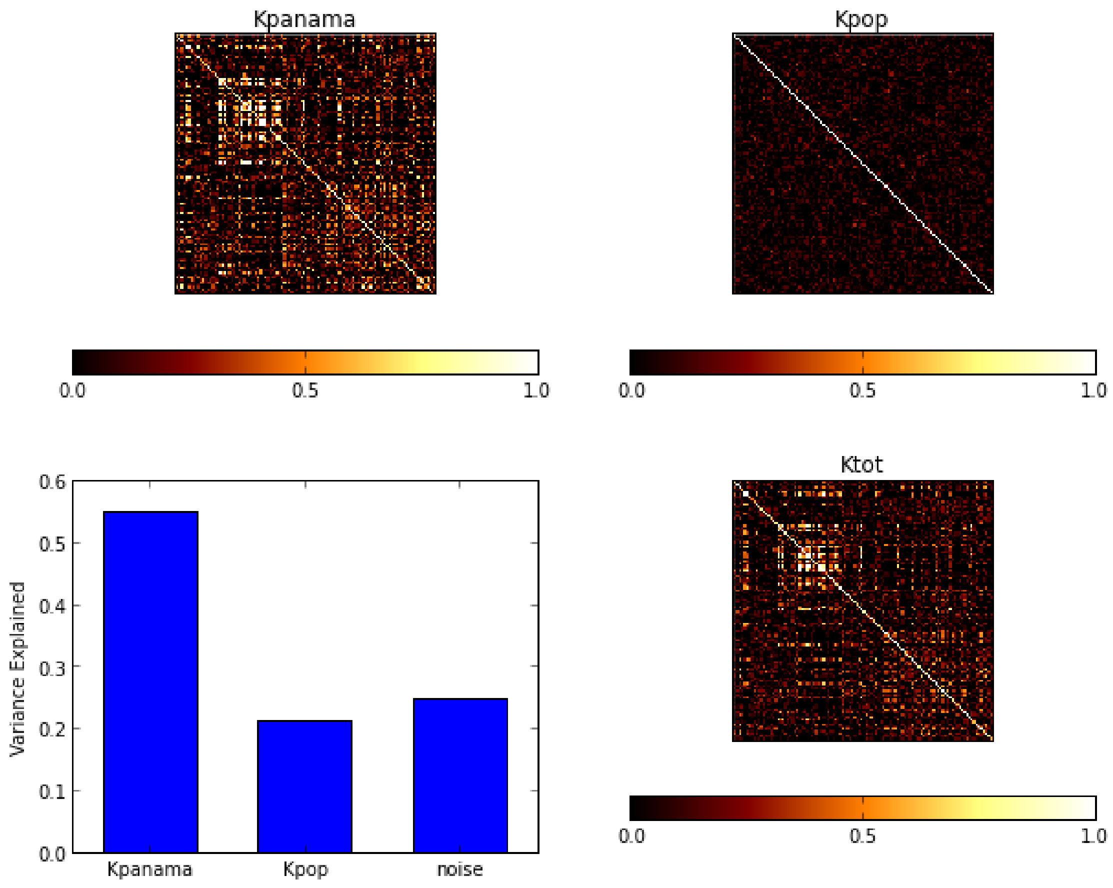
```
In [34]: # import panama
         import limix.modules.panama as PANAMA

         # import data
         phenotypes,sample_idx = data_subsample.getPhenotypes(intersection=True)
         assert sample_idx.all()
         sample_relatedness = data_subsample.getCovariance()
```

```
In [35]: # determine the number of ranks to consider in the PANAMA matrix
         # by looking at the variance explained by PCs
         cum_var = PANAMA.PC_varExplained(phenotypes.values)
         plt = PL.subplot(1,1,1)
         PL.bar(SP.arange(30)+0.5,cum_var[:30],width=1)
         PL.xlim(0,31)
         ticks = SP.linspace(0,30,4); ticks[0] = 1
         plt.set_xticks(ticks)
```

```
Out[35]: [<matplotlib.axis.XTick at 0x115765c50>,
          <matplotlib.axis.XTick at 0x113f16510>,
          <matplotlib.axis.XTick at 0x114ef89d0>,
          <matplotlib.axis.XTick at 0x115887b50>]
```



```
In [36]: # train panama with rank=15 (~60% of the variance)
         p = PANAMA.PANAMA(Y=phenotypes.values,Kpop=sample_relatedness)
         p.train(rank=15)
         # retrieve stuff
         Kpanama = p.get_Kpanama()
         Ktot    = p.get_Ktot()
         vComp   = p.get_varianceComps()

In [37]: # compare Kpanama matrix, Kpop, and the total matrix
         PL.figure(1,figsize=(10,8))
         plt = PL.subplot(2,2,1)
         PL.title('Kpanama')
         PL.imshow(Kpanama,vmin=0,vmax=1,interpolation='none',cmap=cm.afmhot)
         PL.colorbar(ticks=[0,0.5,1],orientation='horizontal')
         plt.set_xticks([])
         plt.set_yticks([])
         plt = PL.subplot(2,2,2)
         PL.title('Kpop')
         PL.imshow(sample_relatedness,vmin=0,vmax=1,interpolation='none',
                   cmap=cm.afmhot)
         PL.colorbar(ticks=[0,0.5,1],orientation='horizontal')
         plt.set_xticks([])
         plt.set_yticks([])
         plt = PL.subplot(2,2,3)
         PL.bar(SP.arange(3)+0.2,vComp,width=0.6)
         plt.set_xticks([0.5,1.5,2.5])
         plt.set_xticklabels(['Kpanama','Kpop','noise'])
         PL.ylabel('Variance Explained')
         plt = PL.subplot(2,2,4)
         PL.title('Ktot')
         PL.imshow(Ktot,vmin=0,vmax=1,interpolation='none',cmap=cm.afmhot)
         PL.colorbar(ticks=[0,0.5,1],orientation='horizontal')
         plt.set_xticks([])
         plt.set_yticks([])

Out[37]: []
```



```
In []:
```

References

[1] Erin N Smith and Leonid Kruglyak. Gene–environment interaction in yeast gene expression. *PLoS biology*, 6(4):e83, 2008.

[2] Christoph Lippert, Jennifer Listgarten, Ying Liu, Carl M Kadie, Robert I Davidson, and David Heckerman. FaST linear mixed models for genome-wide association studies. *Nature Methods*, 8(10):833–835, 2011.

[3] Xiang Zhou and Matthew Stephens. Genome-wide efficient mixed-model analysis for association studies. *Nature genetics*, 44(7):821–824, 2012.

[4] Vincent Segura, Bjarni J Vilhjálmsson, Alexander Platt, Arthur Korte, Ümit Seren, Quan Long, and Magnus Nordborg. An efficient multi-locus mixed-model approach for genome-wide association studies in structured populations. *Nature genetics*, 44(7):825–830, 2012.

[5] Barbara Rakitsch, Christoph Lippert, Karsten Borgwardt, and Oliver Stegle. It is all in the noise: Efficient multi-task gaussian process inference with structured residuals. pages 1466–1474, 2013.

[6] Xiang Zhou and Matthew Stephens. Efficient multivariate linear mixed model algorithms for genome-wide association studies. *Nature Methods, in press*, 2014.

[7] Hyun Min Kang, Chun Ye, and Eleazar Eskin. Accurate discovery of expression quantitative trait loci under confounding from spurious and genuine regulatory hotspots. *Genetics*, 180(4):1909–1925, 2008.

[8] Nicoló Fusi, Oliver Stegle, and Neil D Lawrence. Joint modelling of confounding factors and prominent genetic regulators provides increased accuracy in genetical genomics studies. *PLoS computational biology*, 8(1):e1002330, 2012.