LAPACK / LAPACKE
Eigen

Δρ. Γ. Χαμόδρακας

LAPACK library (1)

- Fortran 90 library
- Provides routines for:
  - solving systems of simultaneous linear equations,
  - least-squares solutions of linear systems of equations,
  - eigenvalue problems,
  - singular value problems,
  - LU, Cholesky, QR, SVD, Schur, generalized Schur matrix factorizations

LAPACK library (2)

- Handles dense and banded but not general sparse matrices
- Supports real and complex matrices in single and double precision
  - i.e. routines with float and double arguments
- http://www.netlib.org/lapack/

LAPACKE C interface

- A 2-level C interface to LAPACK Fortran 90 routines
- **High level interface**:
  - handles all workspace memory allocation internally
- Middle level interface:
  - requires the user to provide workspace arrays as in the original Fortran 90 library
- Supports both column-major and row-major matrices

Column-major / Row-major matrices

- 2d-array example

$$\begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \end{bmatrix}$$

Column-major (Fortran)

Address	Coords	Value
0	[1,1]	11
1	[2,1]	21
2	[1,2]	12
3	[2,2]	22
4	[1,3]	13
5	[2,3]	23

Row-major (C)

Address	Coords	Value
0	[1,1]	11
1	[1,2]	12
2	[1,3]	13
3	[2,1]	21
4	[2,2]	22
5	[2,3]	23

LAPACKE (cont.d)

- Naming Scheme of the C interface:
  - e.g. dgeev: LAPACK routine for eigenvalue and eigenvector computation of a general matrix
  - LAPACKE_dgeev: high-level interface, LAPACKE_dgeev_work: middle-level interface
- 2d-array arguments:
  - Passed as pointers and not as pointers to pointers
  - Due to this pass an extra int parameter for row-major or col-major storage
  - LAPACK_ROW_MAJOR / LAPACK_COL_MAJOR symbolic constants
  - Row-major layout may require more memory and time

LAPACKE (cont.d)

- Extra argument for leading dimension in matrices:
  - row-major, the number of rows
  - col-major, the number of columns
- Prototypes, symbolic constants, macros, type definitions declared in lapacke.h header file
- lapack_int: integer type defined in lapacke.h

LAPACKE installation

1. Download source from: http://www.netlib.org/lapack/
2. tar xvf lapack-3.6.0.tar
3. nano CMakeLists.txt (within lapack-3.6.0 root dir)
4. Find and edit the following lines (replace OFF with ON):

```
option(LAPACKE "Build LAPACKE" ON)
# LAPACKE has also the interface to some routines from tmglib,
# if LAPACKE_WITH_TMG is selected, we need to add those routines to LAPACKE
option(LAPACKE_WITH_TMG "Build LAPACKE with tmglib routines" ON)
if (LAPACKE_WITH_TMG)
  set(LAPACKE ON)
  if(NOT BUILD_TESTING)
    add_subdirectory(TESTING/MATGEN)
   endif(NOT BUILD_TESTING)
endif(LAPACKE_WITH_TMG)
```

5. cmake -DBUILD_SHARED_LIBS=ON (within lapack-3.6.0 root dir)
6. Run make and copy lib/* to /usr/lib; include/* to /usr/include
7. Run ldconfig

- Ubuntu, use apt-get (version 3.5.0):
  - sudo apt-get install liblapacke-dev checkinstall / sudo apt-get install liblapack-doc

LAPACKE inverse

- Computation of the inverse of a matrix using the LU factorization
  - First, call `LAPACKE_dgetrf` to fill the ipiv pivot indices array.
  - `lapack_int LAPACKE_dgetri ( int  matrix_layout, lapack_int n, double * a, lapack_int lda, const lapack_int * ipiv)`
- Documentation:

  http://www.netlib.org/lapack/explore-html/da/d0e/lapacke__dgetri_8c.html

  http://www.netlib.org/lapack/explore-html/df/da4/dgetri_8f.html

  INPUT /OUTPUT PARAMETERS

  matrix_layout: LAPACK_ROW_MAJOR or LAPACK_COL_MAJOR

  n: order of the matrix (num of rows/cols)

  a: the matrix, size n*n (inverse is stored here)

  lda: the leading dimension of a == n

  ipiv: int array, size == n

Matrix Multiplication

- LAPACKE incorporates CBLAS (the C interface for Basic Linear Algebra Subprograms)
  - `void cblas_dgemm(const enum CBLAS_ORDER Order, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_TRANSPOSE TransB, const int M, const int N, const int K, const double alpha, const double *A, const int lda, const double *B, const int ldb, const double beta, double *C, const int ldc);`
- Documentation:
  - http://www.netlib.org/lapack/explore-html/d7/d2b/dgemm_8f.html
  - http://www.netlib.org/clapack/CLAPACK-3.1.1.1/BLAS/WRAP/cblas.h
  - Note the differences for row and col-major arrays (enum and not symbolic constant: CblasRowMajor, CblasColMajor). It is possible to use `LAPACK_COL_MAJOR, LAPACK_ROW_MAJOR`

CBLAS Use

1. Download CBLAS code: http://www.netlib.org/blas/blast-forum/cblas.tgz
2. $ tar xvf cblas.tgz
3. Copy cblas*.h to /usr/include
4. Create symlink in /usr/lib and run ldconfig

   $ sudo ln -sf libblas.so.3 libblas.so

   $ sudo ldconfig
5. Compile with –lblas gcc parameter

LAPACKE eigenproblems

- Eigenvalue and eigenvector computation:
  - `lapack_int LAPACKE_dgeev( int matrix_layout, char jobvl, char jobvr, lapack_int n, double* a, lapack_int lda, double* wr, double* wi, double* vl, lapack_int ldvl, double* vr, lapack_int ldvr );`
- Documentation:

  http://www.netlib.org/lapack/explore-html/d9/d28/dgeev_8f.html

  http://www.netlib.org/lapack/explore-html/de/ddd/lapacke__dgeev_8c.html
- Generalised eigenproblem solution:
  - `lapack_int LAPACKE_dggev( int matrix_layout, char jobvl, char jobvr, lapack_int n, double* a, lapack_int lda, double* b, lapack_int ldb, double* alphar, double* alphai, double* beta, double* vl, lapack_int ldvl, double* vr, lapack_int ldvr );`
- Documentation:

  http://www.netlib.org/lapack/explore-html/d9/d52/dggev_8f.html

  http://www.netlib.org/lapack/explore-html/de/d27/lapacke__dggev_8c.html

LAPACKE eigenproblem example (1)

```
lapack_int LAPACKE_dgeev( int matrix_layout, char jobvl, char
jobvr, lapack_int n, double* a, lapack_int lda, double* wr,
double* wi, double* vl, lapack_int ldvl, double* vr,
lapack_int ldvr );
```

INPUT PARAMETERS

matrix_layout: LAPACK_ROW_MAJOR or LAPACK_COL_MAJOR

jobvl: **'N'** (do not compute left eigenvectors) / 'V' compute

jobvr: 'N' (do not compute right eigenvectors) / **'V' compute**

n: order of the matrix (num of rows/cols)

a: the matrix, size n*n

lda: the leading dimension of a == n

ldvl, ldvr: the leading dimension of left and right eigenvector matrices == n

LAPACKE eigenproblem example (2)

OUTPUT PARAMETERS

wr, wi: arrays with size at least n, containing the real and imaginary parts of the eigenvalues respectively

vr: array containing the right eigenvectors, size at least n*n

If wi[j]== 0 (note that fabs function must be used because wi[j] is double), corresponding eigenvector is the j column of vr.

Compilation/Linking example:

```
$gcc -o eigenprob eigenprob.c -llapacke
```

LAPACKE eigenproblem example (3)

```c
#include <stdio.h>
#include <lapacke.h>
#include <math.h>

int main (int argc, const char * argv[])
{
        double a[3][3] = {6,3,-8,0,-2,0,1,0,-3};
        int i,j;
        lapack_int info,n,lda,ldvl,ldvr;
        double *real, *imaginary, *rvectors, *lvectors;
        n = lda = ldvl = ldvr= 3;
        real = LAPACKE_malloc(sizeof(double)*n);
        imaginary = LAPACKE_malloc(sizeof(double)*n);
        rvectors = LAPACKE_malloc(sizeof(double)*n*n);

        info = LAPACKE_dgeev(LAPACK_ROW_MAJOR,'N','V',n,*a,lda,real,imaginary,lvectors,ldvl,rvectors, ldvr);
        for(i=0;i<n;i++)
        {
                printf("Eigenvalue: %lf + %lfi\n",real[i], imaginary[i]);
                printf("EigenvectorTransposed: [" );
                if (fabs(imaginary[i])<10e-7) {
                        for (j=0;j<n;j++){
                                printf("%lf ", rvectors[j*3+i]);
                        }
                        printf(" ]\n");
                }
        }
        LAPACKE_free(real);
        LAPACKE_free(imaginary);
        LAPACKE_free(rvectors);
        return(info);
}
```

LAPACKE solutions (1)

$$A = \begin{bmatrix} 6 & 3 & -8 \\ 0 & -2 & 0 \\ 1 & 0 & -3 \end{bmatrix}$$

- Eigenvalues:

  [5, -2, -2] (geometric multiplicity = 1)

- Eigenvectors:

$$\begin{bmatrix} 0{,}992278 \\ 0 \\ 0{,}124035 \end{bmatrix}, \begin{bmatrix} 0{,}707107 \\ 0 \\ 0{,}707107 \end{bmatrix}, \begin{bmatrix} 0{,}707107 \\ 0 \\ 0{,}707107 \end{bmatrix}$$

LAPACKE solutions (2)

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

- Eigenvalues:

  [-1, 2, -1] (geometric multiplicity = 2)
- Eigenvectors:

$$\begin{bmatrix} -0{,}816497 \\ 0{,}408248 \\ 0{,}408248 \end{bmatrix}, \begin{bmatrix} 0{,}577350 \\ 0{,}577350 \\ 0{,}577350 \end{bmatrix}, \begin{bmatrix} 0{,}226455 \\ -0.792594 \\ 0{,}566139 \end{bmatrix}$$

LAPACKE Jacobi SVD

- Jacobi Singular Value Decomposition:
  - `lapack_int LAPACKE_dgesvj (int matrix_order, char joba, char jobu, char jobv, lapack_int m, lapack_int n, double *a, lapack_int lda, double *sva, lapack_int mv, double *v, lapack_int ldv, double *stat);`
- Documentation:

  http://www.netlib.org/lapack/explore-html/d1/d5e/dgesvj_8f.html

  http://www.netlib.org/lapack/explore-html/d3/d01/lapacke__dgesvj_8c.html

LAPACKE Jacobi SVD example (1)

INPUT PARAMETERS

```
matrix_layout: LAPACK_ROW_MAJOR or LAPACK_COL_MAJOR
joba: 'G' for a general MxN matrix
jobu: 'N' U matrix is not computed
jobv: 'N' V matrix is not computed
m, n: rows, columns of the matrix
a: the matrix, size m*n
lda: the leading dimension of a
mv: relevant only if V is computed
stat: double array for internal work, size MAX(6,m+n)
```

OUTPUT PARAMETERS

```
sva: double array, size n for storing singular values
v: double array for storing matrix V, not referenced in the
example
```

LAPACKE Jacobi SVD example (2)

```c
#include <stdio.h>
#include <lapacke.h>
#include <math.h>

int main (int argc, const char * argv[])
{
    double input[3][3] = {1,1,0,0,1,1,1,2,1};
    int i,j;
    lapack_int info,n,ldinput,ldv;
    double *singular, *v, *stat;

    n = ldinput = ldv =  3;
    singular = LAPACKE_malloc(sizeof(double)*n);
    stat = LAPACKE_malloc(sizeof(double)*2*n);

    info = LAPACKE_dgesvj(LAPACK_ROW_MAJOR,'G','N','N', n, n, *input, ldinput, singular, 0,  v, ldv, stat);
    for(i=0;i<n;i++)
        printf("Singular Value: %.6e\n",singular[i]);

    LAPACKE_free(singular);
    return(info);
}
```

Eigen Library

- Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.
- http://eigen.tuxfamily.org/dox/
- Eigen supports dense and sparse matrices
- Installation:
  - Download and extract the source code: Eigen is a template library (subdirectory Eigen within the extracted directory)
    - Compilation/Linking: g++ -I /path/to/eigen/ my_program.cpp -o my_program
  - (Optional) Copy or symlink the header files to /usr/local/include
    - Compilation/Linking: g++ my_program.cpp -o my_program

Eigen Library Objects

- Eigen provides two kinds of dense objects: mathematical matrices and vectors which are both represented by the template class Matrix, and general 1D and 2D arrays represented by the template class Array:

```
typedef Matrix<Scalar, RowsAtCompileTime, ColsAtCompileTime,
Options> MyMatrixType;

typedef Array<Scalar, RowsAtCompileTime, ColsAtCompileTime,
Options> MyArrayType;
```

Eigen Library Type Parameters

- Scalar is the scalar type of the coefficients (e.g., float, double, bool, int, etc.).
- RowsAtCompileTime and ColsAtCompileTime are the number of rows and columns of the matrix as known at compile-time or Dynamic.
- Options can be ColMajor or RowMajor, default is ColMajor.

```
Matrix<double, 6, Dynamic> // Dynamic number of columns (heap allocation)
Matrix<double, Dynamic, 2> // Dynamic number of rows (heap allocation)
Matrix<double, Dynamic, Dynamic, RowMajor> // Fully dynamic, row major (heap)
Matrix<double, 13, 3> // Fully fixed (usually allocated on stack)
```

Eigen Convenience Typedefs

```
Matrix<float,Dynamic,Dynamic> <=> MatrixXf
Matrix<double,Dynamic,1> <=> VectorXd
Matrix<int,1,Dynamic> <=> RowVectorXi
Matrix<float,3,3> <=> Matrix3f
Matrix<float,4,1> <=> Vector4f

Array<float,Dynamic,Dynamic> <=> ArrayXXf
Array<double,Dynamic,1> <=> ArrayXd
Array<int,1,Dynamic> <=> RowArrayXi
Array<float,3,3> <=> Array33f
Array<float,4,1> <=> Array4f
```

Eigen Constructors

- **1D**

```
Vector4d  v4;
Vector2f  v1(x, y);
Array3i   v2(x, y, z);
Vector4d  v3(x, y, z, w);
VectorXf  v5; // empty object
ArrayXf   v6(size);

Vector3f  v1;     v1 << x, y, z;
ArrayXf   v2(4);  v2 << 1, 2, 3, 4
```

- **2D**

```
Matrix4f  m1;
MatrixXf  m5; // empty object
MatrixXf  m6(nb_rows, nb_columns);

Matrix3f  m1;   m1 << 1, 2, 3,
                      4, 5, 6,
                      7, 8, 9;
```

Eigen Constructors (external arrays)

```
float data[] = {1,2,3,4};
Map<Vector3f> v1(data);       // uses v1 as a Vector3f object
Map<ArrayXf>  v2(data,3);     // uses v2 as a ArrayXf object
Map<Array22f> m1(data);       // uses m1 as a Array22f object
Map<MatrixXf> m2(data,2,2);   // uses m2 as a MatrixXf object
```

```
rows() function returns the number of rows of a Matrix object
cols() function returns the number of columns of a Matrix
object
size() function returns the total number of coefficients of a
Matrix object
```

Eigen inverse / determinant

- Inverse of a matrix is computed by the inherited member function `inverse()` of the Matrix class template (LU decomposition)
- Determinant of a matrix is computed by the member function `determinant()` of the Matrix class template (LU decomposition)
- Matrix-matrix multiplication is performed with the overloaded `operator*`
- Scalar-matrix multiplication is performed with the operator *
- Check the documentation

Eigenproblem example with Eigen

```cpp
#include <iostream>
#include <Eigen/Dense>
#include <Eigen/Eigenvalues>

using std::endl;
using std::cout;

using Eigen::MatrixXd;
using Eigen::EigenSolver;
using Eigen::Map;


int main()
{
  double data[] = {6,0,1,3,-2,0,-8,0,-3};
  Map<MatrixXd> m(data, 3,3);

  EigenSolver<MatrixXd> es(m);

  cout << "The eigenvalues of A are:" << endl << es.eigenvalues() << endl;
  cout << "The matrix of eigenvectors, V, is:" << endl << es.eigenvectors() << endl << endl;


  return 0;
}
```

Note that matrix data are given with col-major layout

Eigen solutions (1)

$$A = \begin{bmatrix} 6 & 3 & -8 \\ 0 & -2 & 0 \\ 1 & 0 & -3 \end{bmatrix}$$

- Eigenvalues:

  [5, -2, -2] (geometric multiplicity = 1)

- Eigenvectors:

$$\begin{bmatrix} 0{,}992278 \\ 0 \\ 0{,}124035 \end{bmatrix}, \begin{bmatrix} -0{,}707107 \\ 0 \\ -0{,}707107 \end{bmatrix}, \begin{bmatrix} 0{,}707107 \\ 0 \\ 0{,}707107 \end{bmatrix}$$

- Note that ev2 = -1*ev3 (linearly dependent)
- Row 2 of ev3 actually computed as: -7.3271e-16

Eigen solutions (2)

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

- Eigenvalues:

  [-1, 2, -1] (geometric multiplicity = 2)

- Eigenvectors:

$$\begin{bmatrix} -0{,}816497 \\ 0{,}408248 \\ 0{,}408248 \end{bmatrix}, \begin{bmatrix} 0{,}577350 \\ 0{,}577350 \\ 0{,}577350 \end{bmatrix}, \begin{bmatrix} 0 \\ -0{,}707107 \\ 0{,}707107 \end{bmatrix}$$

- Note that LAPACKE_ev3 ≅ -0.277350*ev1 + 0.960769*Eigen_ev3
- Linear combination of independent eigenvectors

Produced Eigenvectors

- When the geometric multiplicity of an eigenvalue is greater than 1, eigenvectors produced by LAPACKE and Eigen may be different and not linearly dependent.
- In that case, produced eigenvectors are a linear combination of N independent eigenvectors, where N is the geometric multiplicity of the eigenvalue.

Jacobi SVD example with Eigen

```cpp
#include <iostream>
#include <Eigen/Dense>
#include <Eigen/Core>
#include <Eigen/SVD>

using std::endl;
using std::cout;
using Eigen::MatrixXd;
using Eigen::JacobiSVD;
using Eigen::Map;

int main()
{
  double data[] = {1,0,1,1,1,2,0,1,1};
  Map<MatrixXd> m(data, 3,3);

  JacobiSVD<MatrixXd> svd(m, Eigen::ComputeFullU);
  JacobiSVD<MatrixXd>::SingularValuesType singular = svd.singularValues();

  cout << "The singular values of A are:" << svd.singularValues() << endl;
  for (int i = 0; i < singular.rows(); i++)
    cout << "Singular Value" << i << ":" << singular(i) << endl;

  return 0;
}
```

Eigen Functors

- `eigenvalues()`, `eigenvectors()` and `singularValues()` functions in the previous examples return objects of the `Matrix` template class (`typedef`'d)
- The `Matrix` template class is a functor. Specifically, the matrix coefficient accessors and mutators are provided through the overloaded parenthesis operator:

```
MatrixXd m(2,2);
m(0,0) = 3;
m(1,0) = 2.5;
m(0,1) = -1;
m(1,1) = m(1,0) + m(0,1);
```

Generalized Eigenproblem

- Eigen does not fully support the solution of a Generalized Eigenproblem
- Eigenvectors are not computed
- In order to resolve this, LAPACKE may be interfaced with Eigen
- LAPACKE must be installed

Generalized Eigenproblem example

```
#include <iostream>
#include <Eigen/Dense>
#include <lapacke.h>

using std::endl;
using std::cout;
using Eigen::MatrixXd;

bool GEP(MatrixXd& A, MatrixXd& B, MatrixXd& v, MatrixXd& lambda);

int main()
{
    MatrixXd A = MatrixXd::Random(4,4);
    MatrixXd B = MatrixXd::Random(4,4);
    MatrixXd V(4,4); // Contains N=4 Eigenvectors (4 rows each)

    /* Contains N=4 Eigenvalues. each eigen value has a real and an imaginary part
       (columns 0 and 1) and a denominator (column 2) */
    MatrixXd lambda(4,3);

    GEP(A,B,V,lambda);
    cout << lambda << endl;
    cout << V << endl;
}

bool GEP(MatrixXd& A, MatrixXd& B, MatrixXd& v, MatrixXd& lambda)
{
  int N = A.cols(); // Number of columns of A and B. Number of rows of v.
  if (B.cols() != N  || A.rows()!=N || B.rows()!=N)
    return false;

  v.resize(N,N);
  lambda.resize(N, 3);

  int LDA = A.outerStride(); int LDB = B.outerStride(); int LDV = v.outerStride(); int INFO = 0;

  double * alphar = lambda.col(0).data();
  double * alphai = lambda.col(1).data();
  double * beta = lambda.col(2).data();

  INFO = LAPACKE_dggev(LAPACK_COL_MAJOR,'N', 'V', N, A.data(), LDA, B.data(), LDB, alphar, alphai, beta, 0, LDV, v.data(), LDV);

  return INFO==0;
}
```