SPEIGS

Generated by Doxygen 1.9.4

Chapter 1

SPEIGS

1.0.0.1 An efficient preprocessor for VERY SParse EIgen-decomposition problem.

SPEIGS implements an efficient eigen-decomposition pre-processor for *Extremely* sparse matrices (arising from SDPs). Given a real symmetric matrix, it computes the full eigen-decomposition $A = V\Lambda V^T$. It internally uses Lapack *deyevr* as the subroutine and detects the possible special structures within the matrix to factorize. It is **recommended** to try the package if A satisfies either of the conditions below

1. A is EXTREMELY sparse. e.g.., there may exist empty row/columns
2. A is possibly low-rank. e.g. Some matrices may be approximately rank-one.

If neither of the two cases is satisfied, we recommend the use of other state-of-the-art sparse eigen-decomposition libraries such as ARPACK and MKL FEAST.

1.0.0.2 Origin

SPEIGS originates from **DSDP5.8** (https://www.mcs.anl.gov/hs/software/DSDP/), a semi-definite programming solver by Steve Benson and is formalized as a library in its successor **HDSDP** (https://github.com/COPT-Public/HDSDP).

1.0.0.3 Current release

The current version of SPEIGS is 1.0.0 and can be called from C and MATLAB Mex interface.

1.0.0.4 Installation

SPEIGS is built via CMAKE system and is linked to MKL library for the implementation of *dsyevr*. The user can also switch to other implementations by modifying the paths and linked libraries from the **CMakeLists.txt**.

```
# Option 1. Link with MKL
set(ENV{MKL_LIB_PATH} YOUR_MKL_PATH
set(ENV{MKL_OMP_PATH} YOUR_OMP_PATH
target_link_libraries(speigs $ENV{MKL_LIB_PATH}/libmkl_core.a)
target_link_libraries(speigs $ENV{MKL_LIB_PATH}/libmkl_intel_lp64.a)
target_link_libraries(speigs $ENV{MKL_LIB_PATH}/libmkl_intel_thread.a)
target_link_libraries(speigs $ENV{MKL_OMP_PATH}/libiomp5.dylib)
# Option 2. Modify the paths to link with other implementations
# set(ENV{LAPACK_BLAS_PATH} YOUR_LAPACK_BLAS_PATH)
# target_link_libraries(speigs $ENV{LAPACK_BLAS_PATH}/liblapack.a)
# target_link_libraries(speigs $ENV{LAPACK_BLAS_PATH}/liblas.a)
```

After configuring the installation paths. Users can execute

```
mkdir build
cmake ..
make
```

in the command line and build the SPEIG library.

1.0.0.5 Documentation

The interface of SPDEIGS is well-documented using **doxygen** system and the users can run

```
cd doc
doxygen .
```

in the command line to generate HTML or LaTex documents to the interface.

1.0.0.6 Examples

The examples for SPEIGS are available at

```
src/example.h
src/example.c
matlab/mex_speigs.c
```

and in a word, SPEIGS runs in a two-phase fashion

- An analysis phase that detects special structure within the matrix
- A factorization phase that extracts the decomposition exploiting the structures from analysis phase

The users can flexibly decide whether to use SPEIGS to factorize based on the result from the analysis phase.

1.0.0.7 Use in MATLAB

SPEIGS is callable from MATLAB by the MEX interface. Users can either build the mexfile using **CMakeLists.txt** from `matlab` directory or download the pre-built mex files and run

```
install_mex
```

in Matlab. On successful installation, SPEIGS can be uses as `eig` function by

```
[V, e] = speigs(A, opts);
```

and `test_speigs.m`, `help speigs` would provide help on how to use the routine.

1.0.0.8 Performance

Since SPEIGS serves as a pre-processor that targets special structures of matrices. If there does exist structures to exploit, SPEIG might be 1000x faster than conventional eigen solvers. SPEIG would be less efficient without structure and users can decide after the analysis phase.

1.0.0.9 Maintainer

SPEIGS is a by-product of the **HDSDP** solver, which is maintained by Wenzhi Gao from Shanghai University of Finance and Economics.

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

File Documentation

3.1 /Users/gaowenzhi/Desktop/public/SPEIGS/matlab/mex_speigs.c File Reference

Mexfile entry function for speigs.

```
#include <stdio.h>
#include "speigs.h"
```

Macros

- #define **V** plhs[0]
- #define **e** plhs[1]
- #define **A** prhs[0]
- #define **opts** prhs[1]

Functions

- static void print_mtype (spint mtype)
  Print type of matrix.
- void mexFunction (int nlhs, mxArray ∗plhs[ ], int nrhs, const mxArray ∗prhs[ ])
  Matlab entry function.

3.1.1 Detailed Description

Mexfile entry function for speigs.

Author

Wenzhi Gao, Shanghai University of Finance and Economics

Date

Aug, 24th, 2022

3.1.2 Function Documentation

3.1.2.1 mexFunction()

```
void mexFunction (
            int nlhs,
            mxArray * plhs[],
            int nrhs,
            const mxArray * prhs[] )
```

Matlab entry function.

Parameters

in	*nlhs*	Number of left-hand-side parameters
out	*plhs*	Pointers for left-hand-side parameters
in	*nrhs*	Number of right-hand-side parameters
out	*prhs*	Pointers for left-hand-side parameters

Matab entry for [V, e] = mex_speigs(A, opts); V is a n by r array that gives r eigenvectors and e is all the nonzero eigen-values. opts.gthresh specifies when submatrix permutation is used opts.tol specifies the criterion to decide if an eigen-value is 0 opts.quiet hides logs during factorization

3.1.2.2 print_mtype()

```
static void print_mtype (
            spint mtype )  [static]
```

Print type of matrix.

Parameters

in	*mtype*	Type of the matrix

3.2 /Users/gaowenzhi/Desktop/public/SPEIGS/src/example.c File Reference

The example for SPEIGS package.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "example.h"
#include "speigs.h"
```

Macros

- #define **sperr**(x) printf(x); return SP_EIGS_ERR;

Functions

- static void **print_mtype** (spint mtype)
- static void **print_evd** (spint n, double ∗e, double ∗V)
- spint test_matrix (spint n, spint ∗Ap, spint ∗Ai, double ∗Ax)

  Test an example of matrix.
- int main ()

  Main function.

3.2.1 Detailed Description

The example for SPEIGS package.

A little example that demonstrates how to use SPEIGS

Author

Wenzhi Gao, Shanghai University of Finance and Economics

Date

Aug, 25th, 2022

3.2.2 Function Documentation

3.2.2.1 main()

```
int main ( )
```

Main function.

Test the SPEIG routines

3.2.2.2 test_matrix()

```
spint test_matrix (
            spint n,
            spint * Ap,
            spint * Ai,
            double * Ax )
```

Test an example of matrix.

Sample usage of SPEIGS routine

3.3 example.h

```
#ifndef example_h
#define example_h

#include "speigs.h"

spint n = 5;

/* Zero matrix */
spint Ap0[] = {0, 0, 0, 0, 0, 0};
spint Ai0[0];
double Ax0[0];

/* Diagonal matrix */
spint Ap1[6] = {0, 1, 2, 3, 3, 4};
spint Ai1[5] = {0, 1, 2, 4};
double Ax1[5] = {1.0, 2.0, 3.0, 4.0};

/* Two-two matrix */
spint Ap2[6] = {0, 1, 1, 1, 1, 1};
spint Ai2[1] = {3};
double Ax2[5] = {10.0};

/* Rank-one matrix */
spint Ap3[6] = {0, 0, 0, 2, 3, 3};
spint Ai3[3] = {2, 3, 3};
double Ax3[3] = {1.0, -2.0, 4.0};

/* Sparse submatrix */
spint Ap4[6] = {0, 0, 0, 2, 3, 3};
spint Ai4[3] = {2, 3, 3};
double Ax4[3] = {-1.0, -2.0, 100.0};

/* General matrix*/
spint Ap5[6] = {0, 5, 9, 12, 14, 15};
spint Ai5[15] = {0, 1, 2, 3, 4,
                    1, 2, 3, 4,
                       2, 3, 4,
                          3, 4,
                             4};
double Ax5[15] = {-1.0, -2.0, 1.0, 3.5, 912,
                  12.5,  -1.0 , -1.3, 50.0, 10.1,
                   13.1, 0.01, 0.5, 9.5, 20.3 };

/* Eigen value and vector */

double e[5] = {0.0};
double V[25] = {0.0};

#endif /* example_h */
```

3.4 /Users/gaowenzhi/Desktop/public/SPEIGS/src/speigs.c File Reference

The implementation of sparse eigen decomposition routine for HDSDP.

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "speigs.h"
#include "spinfo.h"
```

Functions

- static void speig_get_factorize_space (spint ∗n, spint ∗sn, spint ∗type, spint ∗liwork, spint ∗lwork)

  Compute lwork and iwork.
- static void speigs_is_diag (spint ∗p, spint ∗i, spint n, spint ∗is_diag)

  Check if a matrix is diagonal.

- static void speigs_is_rankone (spint ∗p, spint ∗i, double ∗x, spint n, spint ∗is_rankone, double ∗work, double tol)

    Find out if a matrix is rank-one. $A = \alpha a a^T$.
- static void speigs_compute_submat (spint ∗p, spint ∗i, spint n, spint ∗sn, spint ∗nnzs, spint ∗perm, spint ∗iperm)

    Compute the dense submatrix of a large sparse matrix.
- static spint speigs_factorize_zero (spint ∗p, spint ∗i, double ∗x, spint n, spint ∗aiwork, double ∗awork, spint ∗sn, spint ∗iwork, spint ∗liwork, double ∗work, spint ∗lwork, double ∗evals, double ∗evecs, spint ∗rank, double tol)

    Compute the eigen factorization of an all-zero matrix.
- static spint speigs_factorize_diag (spint ∗p, spint ∗i, double ∗x, spint n, spint ∗aiwork, double ∗awork, spint ∗sn, spint ∗iwork, spint ∗liwork, double ∗work, spint ∗lwork, double ∗evals, double ∗evecs, spint ∗rank, double tol)

    Compute the eigen factorization of a diagonal matrix.
- static spint speigs_factorize_two (spint ∗p, spint ∗i, double ∗x, spint n, spint ∗aiwork, double ∗awork, spint ∗sn, spint ∗iwork, spint ∗liwork, double ∗work, spint ∗lwork, double ∗evals, double ∗evecs, spint ∗rank, double tol)

    Compute the eigen factorization of a two-two matrix.
- static spint speigs_factorize_rankone (spint ∗p, spint ∗i, double ∗x, spint n, spint ∗aiwork, double ∗awork, spint ∗sn, spint ∗iwork, spint ∗liwork, double ∗work, spint ∗lwork, double ∗evals, double ∗evecs, spint ∗rank, double tol)

    Compute the eigen factorization of a rank-one matrix.
- static spint speigs_factorize_dense (double ∗a, double ∗evals, double ∗evecs, spint ∗n, spint ∗liwork, spint ∗iwork, spint ∗lwork, double ∗work, spint ∗isuppz)

    Compute the eigen factorization of a general full matrix.
- static spint speigs_factorize_sparse (spint ∗p, spint ∗i, double ∗x, spint n, spint ∗aiwork, double ∗awork, spint ∗sn, spint ∗iwork, spint ∗liwork, double ∗work, spint ∗lwork, double ∗evals, double ∗evecs, spint ∗rank, double tol)

    Compute the eigen factorization of a sparse matrix admitting an easier submatrix representation.
- static spint speigs_factorize_general (spint ∗p, spint ∗i, double ∗x, spint n, spint ∗aiwork, double ∗awork, spint ∗sn, spint ∗iwork, spint ∗liwork, double ∗work, spint ∗lwork, double ∗evals, double ∗evecs, spint ∗rank, double tol)

    Compute the eigen factorization of a general dense matrix.
- spint speigs_analyze (spint ∗Ap, spint ∗Ai, double ∗Ax, spint ∗dim, spint ∗iwork, spint ∗liwork, double ∗work, spint ∗lwork, spint ∗type, spint ∗sn, double tol, double gthresh)

    Perform the analysis phase of sparse eigen-value factorization.
- spint speigs_factorize (spint ∗Ap, spint ∗Ai, double ∗Ax, spint ∗dim, spint ∗aiwork, double ∗awork, spint ∗type, spint ∗sn, spint ∗iwork, spint ∗liwork, double ∗work, spint ∗lwork, double ∗evals, double ∗evecs, spint ∗rank, double tol)

    Perform the analysis phase of sparse eigen-value factorization.

Variables

- static char **jobz** = 'V'
- static char **range** = 'A'
- static char **uplolow** = 'L'
- static double **abstol** = 0.0
- static spint(∗ speig_routines [6])(spint ∗, spint ∗, double ∗, spint, spint ∗, double ∗, spint ∗, spint ∗, spint ∗, double ∗, spint ∗, double ∗, double ∗, spint ∗, double)

    The jump table for eigen routines.

3.4.1 Detailed Description

The implementation of sparse eigen decomposition routine for HDSDP.

A set of routines that factorize very sparse matrices that typically arise from semi-definite programming problems. The routines detect the special structures of the matrix and accelerate the factorization procedure.

Author

Wenzhi Gao, Shanghai University of Finance and Economics

Date

Aug, 24th, 2022

3.4.2 Function Documentation

3.4.2.1 speig_get_factorize_space()

```
static void speig_get_factorize_space (
            spint * n,
            spint * sn,
            spint * type,
            spint * liwork,
            spint * lwork )  [static]
```

Compute lwork and iwork.

Parameters

in	*n*	Dimension of the matrix
in	*sn*	Dimension of the submatrix
in	*type*	Type of the matrix
out	*liwork*	Estimated integer workspace length
out	*lwork*	Estimated double workspace length

The function computes the space requirement for the subroutines that will be invoked in the factorization phase

3.4.2.2 speigs_analyze()

```
spint speigs_analyze (
            spint * Ap,
            spint * Ai,
            double * Ax,
            spint * dim,
            spint * iwork,
```

```
            spint * liwork,
            double * work,
            spint * lwork,
            spint * type,
            spint * sn,
            double tol,
            double gthresh )
```

Perform the analysis phase of sparse eigen-value factorization.

Parameters

in	*Ap*	CSC format column pointer
in	*Ai*	CSC format row index
in	*Ax*	CSC format matrix nonzero entries
in	*dim*	Dimension of the matrix
out	*iwork*	Integer working array for the analysis phase
in	*liwork*	Length of "iwork" or the expected length of integer working array
out	*work*	Double working array for the analysis phase
in	*lwork*	Length of "lwork" or the expected length of double working array
out	*type*	Type of the matrix
out	*sn*	Size of the submatrix
in	*tol*	Tolerance to classify if a matrix is rank-one by $\|\|A - aa^T\|\|_F \leq tol$
in	*gthresh*	Threshold of (submatrix size / dim) classifying a matrix as general or sparse

Returns

retcode Status of the analysis phase

Perform the analysis phase of the sparse eigen-value factorization.

If all the necessary memories are allocated, on exit, "work" and "iwork" are filled by the intermediate information which can be used in the factorization phase; "type" is filled by one of the five types; "sn" is filled by size of the submatrix.

If "dim" is supplied and the rest of the working array is incomplete, "lwork" and "work" will be respectively filled by the expected length of the double and integer working arrays

3.4.2.3 speigs_compute_submat()

```
static void speigs_compute_submat (
            spint * p,
            spint * i,
            spint n,
            spint * sn,
            spint * nnzs,
            spint * perm,
            spint * iperm )  [static]
```

Compute the dense submatrix of a large sparse matrix.

Parameters

in	*p*	CSC format column pointer
in	*i*	CSC format row index
in	*n*	Dimension of the matrix
out	*sn*	Dimension of the submatrix
in	*nnzs*	Number of nonzeros in each column
out	*perm*	Permutation that gathers nonzero elements
out	*iperm*	Inverse permutation

On exit, "perm" and "iperm" will be filled by the permutation and its inverse respectively

3.4.2.4 speigs_factorize()

```
spint speigs_factorize (
            spint * Ap,
            spint * Ai,
            double * Ax,
            spint * dim,
            spint * aiwork,
            double * awork,
            spint * type,
            spint * sn,
            spint * iwork,
            spint * liwork,
            double * work,
            spint * lwork,
            double * evals,
            double * evecs,
            spint * rank,
            double tol )
```

Perform the analysis phase of sparse eigen-value factorization.

Parameters

in	*Ap*	CSC format column pointer
in	*Ai*	CSC format row index
in	*Ax*	CSC format matrix nonzero entries
in	*dim*	Dimension of the matrix
in	*aiwork*	Integer working array from the analysis phase
in	*awork*	Double working array from the analysis phase
in	*type*	"type" from the analysis phase
in	*sn*	"sn" from the analysis phase
in	*iwork*	Integer working array for the factorization phase
in	*liwork*	Length of "iwork" or the expected length of integer working array
in	*work*	Double working array for the factorization phase
in	*lwork*	Length of "lwork" or the expected length of double working array
out	*evals*	Eigen-values after factorization
out	*evecs*	Eigen-vectors after factorization
out	*rank*	Rank of the factorized matrix
in	*tol*	Tolerance to tell if an eigen-value is 0

Returns

retcode Status of the factorization phase

Perform the analysis phase of the sparse eigen-value factorization.

If all the necessary memories are allocated, on exit, "work" and "iwork" are filled by the intermediate information which can be used in the factorization phase; "type" is filled by one of the five types; "sn" is filled by size of the submatrix.

If "dim" is supplied and the rest of the working array is incomplete, "lwork" and "work" will be respectively filled by the expected length of the double and integer working arrays

3.4.2.5 speigs_factorize_dense()

```
static spint speigs_factorize_dense (
            double * a,
            double * evals,
            double * evecs,
            spint * n,
            spint * liwork,
            spint * iwork,
            spint * lwork,
            double * work,
            spint * isuppz )  [static]
```

Compute the eigen factorization of a general full matrix.

Parameters

in	*a*	Dense array that contains the matrix to factorize
out	*evals*	Eigen-values after factorization
out	*evecs*	Eigen-vectors after factorization
in	*n*	Dimension of the dense matrix
in	*liwork*	Length of the integer working array for Lapack
in	*iwork*	Integer working array for Lapack
in	*lwork*	Length of double working array for Lapack
in	*work*	Double working array for Lapack
in	*isuppz*	Auxiliary placeholder for Lapack parameter

Returns

retcode Status of the factorization

On exit, "evals" and "evecs" will be overwritten by the eigen-decomposition of the matrix. "rank" is the rank of the matrix The routine is a wrapper of the Lapack dsyevr function

3.4.2.6 speigs_factorize_diag()

```
static spint speigs_factorize_diag (
            spint * p,
```

```
            spint * i,
            double * x,
            spint n,
            spint * aiwork,
            double * awork,
            spint * sn,
            spint * iwork,
            spint * liwork,
            double * work,
            spint * lwork,
            double * evals,
            double * evecs,
            spint * rank,
            double tol )  [static]
```

Compute the eigen factorization of a diagonal matrix.

Parameters

in	*p*	CSC format column pointer
in	*i*	CSC format row index
in	*x*	CSC format matrix nonzero entries
in	*n*	Dimension of the matrix
in	*aiwork*	Integer working array from the analysis phase
in	*awork*	Double working array from the analysis phase
in	*sn*	Dimension of the submatrix
in	*iwork*	Integer working array for the factorization phase
in	*liwork*	Length of "iwork"
in	*work*	Double working array for the factorization phase
in	*lwork*	Length of "work"
out	*evals*	Eigen-values after factorization
out	*evecs*	Eigen-vectors after factorization
out	*rank*	Rank of the factorized matrix
in	*tol*	Tolerance to tell if an eigen-value is 0

Returns

retcode Status of the factorization

On exit, "evals" and "evecs" will be overwritten by the eigen-decomposition of the matrix. "rank" is the rank of the matrix Since the matrix is diagonal, all the eigen-vectors are unit vectors and eigen-values are determined by the elements in "x"

3.4.2.7 speigs_factorize_general()

```
static spint speigs_factorize_general (
            spint * p,
            spint * i,
            double * x,
            spint n,
            spint * aiwork,
            double * awork,
```

```
            spint * sn,
            spint * iwork,
            spint * liwork,
            double * work,
            spint * lwork,
            double * evals,
            double * evecs,
            spint * rank,
            double tol )  [static]
```

Compute the eigen factorization of a general dense matrix.

Parameters

in	*p*	CSC format column pointer
in	*i*	CSC format row index
in	*x*	CSC format matrix nonzero entries
in	*n*	Dimension of the matrix
in	*aiwork*	Integer working array from the analysis phase
in	*awork*	Double working array from the analysis phase
in	*sn*	Dimension of the submatrix
in	*iwork*	Integer working array for the factorization phase
in	*liwork*	Length of "iwork"
in	*work*	Double working array for the factorization phase
in	*lwork*	Length of "work"
out	*evals*	Eigen-values after factorization
out	*evecs*	Eigen-vectors after factorization
out	*rank*	Rank of the factorized matrix
in	*tol*	Tolerance to tell if an eigen-value is 0

Returns

retcode Status of the factorization

On exit, "evals" and "evecs" will be overwritten by the eigen-decomposition of the matrix. "rank" is the rank of the matrix The routine converts the sparse matrix into a dense array and calls Lapack directly. Slow in general

3.4.2.8 speigs_factorize_rankone()

```
static spint speigs_factorize_rankone (
            spint * p,
            spint * i,
            double * x,
            spint n,
            spint * aiwork,
            double * awork,
            spint * sn,
            spint * iwork,
            spint * liwork,
            double * work,
            spint * lwork,
            double * evals,
```

```
            double * evecs,
            spint * rank,
            double tol )  [static]
```

Compute the eigen factorization of a rank-one matrix.

Parameters

in	*p*	CSC format column pointer
in	*i*	CSC format row index
in	*x*	CSC format matrix nonzero entries
in	*n*	Dimension of the matrix
in	*aiwork*	Integer working array from the analysis phase
in	*awork*	Double working array from the analysis phase
in	*sn*	Dimension of the submatrix
in	*iwork*	Integer working array for the factorization phase
in	*liwork*	Length of "iwork"
in	*work*	Double working array for the factorization phase
in	*lwork*	Length of "work"
out	*evals*	Eigen-values after factorization
out	*evecs*	Eigen-vectors after factorization
out	*rank*	Rank of the factorized matrix
in	*tol*	Tolerance to tell if an eigen-value is 0

Returns

retcode Status of the factorization

On exit, "evals" and "evecs" will be overwritten by the eigen-decomposition of the matrix. "rank" is the rank of the matrix Since the matrix is rank-one, the "awork" array from the analysis phase contains the eigen-decomposition

3.4.2.9 speigs_factorize_sparse()

```
static spint speigs_factorize_sparse (
            spint * p,
            spint * i,
            double * x,
            spint n,
            spint * aiwork,
            double * awork,
            spint * sn,
            spint * iwork,
            spint * liwork,
            double * work,
            spint * lwork,
            double * evals,
            double * evecs,
            spint * rank,
            double tol )  [static]
```

Compute the eigen factorization of a sparse matrix admitting an easier submatrix representation.

Parameters

in	*p*	CSC format column pointer
in	*i*	CSC format row index
in	*x*	CSC format matrix nonzero entries
in	*n*	Dimension of the matrix
in	*aiwork*	Integer working array from the analysis phase
in	*awork*	Double working array from the analysis phase
in	*sn*	Dimension of the submatrix
in	*iwork*	Integer working array for the factorization phase
in	*liwork*	Length of "iwork"
in	*work*	Double working array for the factorization phase
in	*lwork*	Length of "work"
out	*evals*	Eigen-values after factorization
out	*evecs*	Eigen-vectors after factorization
out	*rank*	Rank of the factorized matrix
in	*tol*	Tolerance to tell if an eigen-value is 0

Returns

retcode Status of the factorization

On exit, "evals" and "evecs" will be overwritten by the eigen-decomposition of the matrix. "rank" is the rank of the matrix The routine uses the permutation and inverse permutation information collected in the analysis phase to formulate the submatrix, factorizes the submatrix and finally recovers the decomposition using the inverse permutation

3.4.2.10 speigs_factorize_two()

```
static spint speigs_factorize_two (
            spint * p,
            spint * i,
            double * x,
            spint n,
            spint * aiwork,
            double * awork,
            spint * sn,
            spint * iwork,
            spint * liwork,
            double * work,
            spint * lwork,
            double * evals,
            double * evecs,
            spint * rank,
            double tol )  [static]
```

Compute the eigen factorization of a two-two matrix.

Parameters

in	*p*	CSC format column pointer
in	*i*	CSC format row index

Parameters

in	*x*	CSC format matrix nonzero entries
in	*n*	Dimension of the matrix
in	*aiwork*	Integer working array from the analysis phase
in	*awork*	Double working array from the analysis phase
in	*sn*	Dimension of the submatrix
in	*iwork*	Integer working array for the factorization phase
in	*liwork*	Length of "iwork"
in	*work*	Double working array for the factorization phase
in	*lwork*	Length of "work"
out	*evals*	Eigen-values after factorization
out	*evecs*	Eigen-vectors after factorization
out	*rank*	Rank of the factorized matrix
in	*tol*	Tolerance to tell if an eigen-value is 0

Returns

retcode Status of the factorization

On exit, "evals" and "evecs" will be overwritten by the eigen-decomposition of the matrix. "rank" is the rank of the matrix Since the matrix composes of 2 by 2 submatrices, Givens' rotation is employed to factorize the matrixf

3.4.2.11 speigs_factorize_zero()

```
static spint speigs_factorize_zero (
            spint * p,
            spint * i,
            double * x,
            spint n,
            spint * aiwork,
            double * awork,
            spint * sn,
            spint * iwork,
            spint * liwork,
            double * work,
            spint * lwork,
            double * evals,
            double * evecs,
            spint * rank,
            double tol ) [static]
```

Compute the eigen factorization of an all-zero matrix.

Parameters

in	*p*	CSC format column pointer
in	*i*	CSC format row index
in	*x*	CSC format matrix nonzero entries
in	*n*	Dimension of the matrix
in	*aiwork*	Integer working array from the analysis phase
in	*awork*	Double working array from the analysis phase

Parameters

in	*sn*	Dimension of the submatrix
in	*iwork*	Integer working array for the factorization phase
in	*liwork*	Length of "iwork"
in	*work*	Double working array for the factorization phase
in	*lwork*	Length of "work"
out	*evals*	Eigen-values after factorization
out	*evecs*	Eigen-vectors after factorization
out	*rank*	Rank of the factorized matrix
in	*tol*	Tolerance to tell if an eigen-value is 0

Returns

retcode Status of the factorization

On exit, "evals" and "evecs" will be overwritten by the eigen-decomposition of the matrix. "rank" is the rank of the matrix Since the matrix is all-zero, no operation is needed.

3.4.2.12 speigs_is_diag()

```
static void speigs_is_diag (
            spint * p,
            spint * i,
            spint n,
            spint * is_diag )  [static]
```

Check if a matrix is diagonal.

Parameters

in	*p*	CSC format column pointer
in	*i*	CSC format row index
in	*n*	Dimension of the matrix
out	*is_diag*	Is the matrix diagonal?

3.4.2.13 speigs_is_rankone()

```
static void speigs_is_rankone (
            spint * p,
            spint * i,
            double * x,
            spint n,
            spint * is_rankone,
            double * work,
            double tol )  [static]
```

Find out if a matrix is rank-one. $A = \alpha a a^T$.

Parameters

in	*p*	CSC format column pointer
in	*i*	CSC format row index
in	*x*	CSC format matrix nonzero entries
in	*n*	Dimension of the matrix
out	*is_rankone*	Is the matrix rank-one?
out	*work*	Working array for rank-one detection
in	*tol*	Tolerance for rank-one classification $\|\|A - aa^T\|\|_F \leq tol$

On exit, the array "work" would be filled by the rank-one factor a if A is rank-one

3.4.3 Variable Documentation

3.4.3.1 speig_routines

```
spint(* speig_routines[6])(spint *, spint *, double *, spint, spint *, double *, spint *, spint
*, spint *, double *, spint *, double *, double *, spint *, double) (
            spint * ,
            spint * ,
            double * ,
            spint ,
            spint * ,
            double * ,
            spint * ,
            spint * ,
            spint * ,
            double * ,
            spint * ,
            double * ,
            double * ,
            spint * ,
            double ) [static]
```

Initial value:

```
=
{
    &speigs_factorize_zero,
    &speigs_factorize_sparse,
    &speigs_factorize_general,
    &speigs_factorize_rankone,
    &speigs_factorize_diag,
    &speigs_factorize_two
}
```

The jump table for eigen routines.

Currently contains six implementations of eigen routines

3.5 /Users/gaowenzhi/Desktop/public/SPEIGS/src/speigs.h File Reference

Header for basic types and routine list.

```
#include <stddef.h>
```

Macros

- #define **id** "%d"
- #define **sperr**(x) printf(x);
- #define **SP_EIGS_OK** (0)
- #define **SP_EIGS_ERR** (1)
- #define **MATRIX_TYPE_ZERO** (0)
- #define **MATRIX_TYPE_SPARSE** (1)
- #define **MATRIX_TYPE_GENERAL** (2)
- #define **MATRIX_TYPE_RANKONE** (3)
- #define **MATRIX_TYPE_DIAG** (4)
- #define **MATRIX_TYPE_TWOTWO** (5)
- #define **SPEIG_VER** (1)

Typedefs

- typedef int32_t **spint**

Functions

- spint speigs_analyze (spint ∗Ap, spint ∗Ai, double ∗Ax, spint ∗dim, spint ∗iwork, spint ∗liwork, double ∗work, spint ∗lwork, spint ∗type, spint ∗sn, double tol, double gthresh)

  Perform the analysis phase of sparse eigen-value factorization.
- spint speigs_factorize (spint ∗Ap, spint ∗Ai, double ∗Ax, spint ∗dim, spint ∗aiwork, double ∗awork, spint ∗type, spint ∗sn, spint ∗iwork, spint ∗liwork, double ∗work, spint ∗lwork, double ∗evals, double ∗evecs, spint ∗rank, double tol)

  Perform the analysis phase of sparse eigen-value factorization.

3.5.1 Detailed Description

Header for basic types and routine list.

Implement the eigen-decomposition algorithm from DSDP5.8 by Steve Benson.

Given a real symmetric matrix A, the routine explores special structures within and computes the full eigen-decomposition of the matrix. In the backend the routine calls Lapack dsyev (or Netlib Eispack) to decompose the pre-processed system.

This routine is also employed in HDSDP solver for SDP.

Author

Wenzhi Gao, Shanghai University of Finance and Economics

Date

Aug, 24th, 2022

3.5.2 Function Documentation

3.5.2.1 speigs_analyze()

```
spint speigs_analyze (
            spint * Ap,
            spint * Ai,
            double * Ax,
            spint * dim,
            spint * iwork,
            spint * liwork,
            double * work,
            spint * lwork,
            spint * type,
            spint * sn,
            double tol,
            double gthresh )
```

Perform the analysis phase of sparse eigen-value factorization.

Parameters

in	*Ap*	CSC format column pointer
in	*Ai*	CSC format row index
in	*Ax*	CSC format matrix nonzero entries
in	*dim*	Dimension of the matrix
out	*iwork*	Integer working array for the analysis phase
in	*liwork*	Length of "iwork" or the expected length of integer working array
out	*work*	Double working array for the analysis phase
in	*lwork*	Length of "lwork" or the expected length of double working array
out	*type*	Type of the matrix
out	*sn*	Size of the submatrix
in	*tol*	Tolerance to classify if a matrix is rank-one by $\|\|A - aa^T\|\|_F \leq tol$
in	*gthresh*	Threshold of (submatrix size / dim) classifying a matrix as general or sparse

Returns

retcode Status of the analysis phase

Perform the analysis phase of the sparse eigen-value factorization.

If all the necessary memories are allocated, on exit, "work" and "iwork" are filled by the intermediate information which can be used in the factorization phase; "type" is filled by one of the five types; "sn" is filled by size of the submatrix.

If "dim" is supplied and the rest of the working array is incomplete, "lwork" and "work" will be respectively filled by the expected length of the double and integer working arrays

3.5.2.2 speigs_factorize()

```
spint speigs_factorize (
            spint * Ap,
            spint * Ai,
            double * Ax,
            spint * dim,
            spint * aiwork,
            double * awork,
            spint * type,
            spint * sn,
            spint * iwork,
            spint * liwork,
            double * work,
            spint * lwork,
            double * evals,
            double * evecs,
            spint * rank,
            double tol )
```

Perform the analysis phase of sparse eigen-value factorization.

Parameters

in	*Ap*	CSC format column pointer
in	*Ai*	CSC format row index
in	*Ax*	CSC format matrix nonzero entries
in	*dim*	Dimension of the matrix
in	*aiwork*	Integer working array from the analysis phase
in	*awork*	Double working array from the analysis phase
in	*type*	"type" from the analysis phase
in	*sn*	"sn" from the analysis phase
in	*iwork*	Integer working array for the factorization phase
in	*liwork*	Length of "iwork" or the expected length of integer working array
in	*work*	Double working array for the factorization phase
in	*lwork*	Length of "lwork" or the expected length of double working array
out	*evals*	Eigen-values after factorization
out	*evecs*	Eigen-vectors after factorization
out	*rank*	Rank of the factorized matrix
in	*tol*	Tolerance to tell if an eigen-value is 0

Returns

retcode Status of the factorization phase

Perform the analysis phase of the sparse eigen-value factorization.

If all the necessary memories are allocated, on exit, "work" and "iwork" are filled by the intermediate information which can be used in the factorization phase; "type" is filled by one of the five types; "sn" is filled by size of the submatrix.

If "dim" is supplied and the rest of the working array is incomplete, "lwork" and "work" will be respectively filled by the expected length of the double and integer working arrays

3.6 speigs.h

Go to the documentation of this file.

```
#ifndef speigs_h
#define speigs_h

#include <stddef.h>

#ifdef MATLAB_MEX_FILE
#include "mex.h"
typedef mwSize spint;
#define sperr mexErrMsgTxt
#define id "%lld"
#else
#ifdef SPEIG_64
typedef int64_t spint;
#define id "%lld"
#else
typedef int32_t spint;
#define id "%d"
#endif
#define sperr(x) printf(x);
#endif

/* Return code */
#define SP_EIGS_OK            (0)
#define SP_EIGS_ERR           (1)

/* Matrix type */
#define MATRIX_TYPE_ZERO      (0)
#define MATRIX_TYPE_SPARSE    (1)
#define MATRIX_TYPE_GENERAL   (2)
#define MATRIX_TYPE_RANKONE   (3)
#define MATRIX_TYPE_DIAG      (4)
#define MATRIX_TYPE_TWOTWO    (5)

#ifdef __cplusplus
extern "C" {
#endif
extern spint speigs_analyze( spint *Ap,    spint *Ai,     double *Ax,   spint  *dim,
                             spint *iwork, spint *liwork, double *work, spint  *lwork,
                             spint *type,  spint *sn,     double tol,   double gthresh );

extern spint speigs_factorize( spint  *Ap,     spint  *Ai,    double *Ax,    spint  *dim,   spint *aiwork,
                               double *awork,  spint  *type,  spint  *sn,    spint  *iwork, spint *liwork,
                               double *work,   spint  *lwork, double *evals, double *evecs,
                               spint  *rank,   double tol );
#ifdef __cplusplus
}
#endif

#define SPEIG_VER  (1) // Version number

#endif /* speigs_h */
```

3.7 /Users/gaowenzhi/Desktop/public/SPEIGS/src/spinfo.h File Reference

Header defining internal constants for speigs.

Macros

- #define **TRUE** (1)
- #define **FALSE** (0)
- #define **ROOT** (7.0710678118654757273731092936941422522068e-01)

  $\frac{\sqrt{2}}{2}$
- #define **LAPACK_IWORK** (12)
- #define **LAPACK_LWORK** (30)

Functions

- void dsyevr (const char ∗jobz, const char ∗range, const char ∗uplo, const spint ∗n, double ∗a, const spint ∗lda, const double ∗vl, const double ∗vu, const spint ∗il, const spint ∗iu, const double ∗abstol, spint ∗m, double ∗w, double ∗z, const spint ∗ldz, spint ∗isuppz, double ∗work, const spint ∗lwork, spint ∗iwork, const spint ∗liwork, spint ∗info)

  Lapack dense eigen routine.

3.7.1 Detailed Description

Header defining internal constants for speigs.

Author

Wenzhi Gao, Shanghai University of Finance and Economics

Date

Aug, 24th, 2022

3.7.2 Function Documentation

3.7.2.1 dsyevr()

```
void dsyevr (
            const char * jobz,
            const char * range,
            const char * uplo,
            const spint * n,
            double * a,
            const spint * lda,
            const double * vl,
            const double * vu,
            const spint * il,
            const spint * iu,
            const double * abstol,
            spint * m,
            double * w,
            double * z,
            const spint * ldz,
            spint * isuppz,
            double * work,
            const spint * lwork,
            spint * iwork,
            const spint * liwork,
            spint * info )
```

Lapack dense eigen routine.

The Lapack eigen routine

3.8 spinfo.h

Go to the documentation of this file.

```
#ifndef spinfo_h
#define spinfo_h

/* Boolean */
#ifndef TRUE
#define TRUE                  (1)
#define FALSE                 (0)
#endif

/* Some constants */
#define ROOT                  (7.071067811865475727373109293694142252206 8e-01)
#define LAPACK_IWORK          (12)
#define LAPACK_LWORK          (30)

#ifdef __cplusplus
extern "C" {
#endif

extern void dsyevr( const char   *jobz,   const char   *range,  const char  *uplo,
                    const spint  *n,            double *a,      const spint *lda,
                    const double *vl,     const double *vu,     const spint *il,
                    const spint  *iu,     const double *abstol,       spint *m,
                          double *w,            double *z,      const spint *ldz,
                          spint  *isuppz,       double *work,   const spint *lwork,
                          spint  *iwork,  const spint  *liwork, spint *info         );
#ifdef __cplusplus
}
#endif

#endif /* spinfo_h */
```

Index