15 Eigensystems

This chapter describes functions for computing eigenvalues and eigenvectors of matrices. There are routines for real symmetric, real nonsymmetric, complex hermitian, real generalized symmetric-definite, complex generalized hermitian-definite, and real generalized nonsymmetric eigensystems. Eigenvalues can be computed with or without eigenvectors. The hermitian and real symmetric matrix algorithms are symmetric bidiagonalization followed by QR reduction. The nonsymmetric algorithm is the Francis QR double-shift. The generalized nonsymmetric algorithm is the QZ method due to Moler and Stewart.

The functions described in this chapter are declared in the header file '`gsl_eigen.h`'.

15.1 Real Symmetric Matrices

For real symmetric matrices, the library uses the symmetric bidiagonalization and QR reduction method. This is described in Golub & van Loan, section 8.3. The computed eigenvalues are accurate to an absolute accuracy of $\epsilon ||A||_2$, where ϵ is the machine precision.

`gsl_eigen_symm_workspace * gsl_eigen_symm_alloc` (*const size_t* `n`) [Function]

This function allocates a workspace for computing eigenvalues of n-by-n real symmetric matrices. The size of the workspace is $O(2n)$.

`void gsl_eigen_symm_free` (*gsl_eigen_symm_workspace * * `w`) [Function]

This function frees the memory associated with the workspace *w*.

`int gsl_eigen_symm` (*gsl_matrix * * `A`, *gsl_vector * * `eval`, *gsl_eigen_symm_workspace * * `w`) [Function]

This function computes the eigenvalues of the real symmetric matrix A. Additional workspace of the appropriate size must be provided in *w*. The diagonal and lower triangular part of A are destroyed during the computation, but the strict upper triangular part is not referenced. The eigenvalues are stored in the vector *eval* and are unordered.

`gsl_eigen_symmv_workspace * gsl_eigen_symmv_alloc` (*const size_t* `n`) [Function]

This function allocates a workspace for computing eigenvalues and eigenvectors of n-by-n real symmetric matrices. The size of the workspace is $O(4n)$.

`void gsl_eigen_symmv_free` (*gsl_eigen_symmv_workspace * * `w`) [Function]

This function frees the memory associated with the workspace *w*.

`int gsl_eigen_symmv` (*gsl_matrix * * `A`, *gsl_vector * * `eval`, *gsl_matrix * * `evec`, *gsl_eigen_symmv_workspace * * `w`) [Function]

This function computes the eigenvalues and eigenvectors of the real symmetric matrix A. Additional workspace of the appropriate size must be provided in *w*. The diagonal and lower triangular part of A are destroyed during the computation, but the strict upper triangular part is not referenced. The eigenvalues are stored in the vector *eval* and are unordered. The corresponding eigenvectors are stored in the columns of the matrix *evec*. For example, the eigenvector in the first column corresponds to the first eigenvalue. The eigenvectors are guaranteed to be mutually orthogonal and normalised to unit magnitude.

15.2 Complex Hermitian Matrices

For hermitian matrices, the library uses the complex form of the symmetric bidiagonalization and QR reduction method.

`gsl_eigen_herm_workspace * gsl_eigen_herm_alloc` (*const size_t* `n`) [Function]

This function allocates a workspace for computing eigenvalues of n-by-n complex hermitian matrices. The size of the workspace is $O(3n)$.

`void gsl_eigen_herm_free` (*gsl_eigen_herm_workspace ** `w`) [Function]

This function frees the memory associated with the workspace *w*.

`int gsl_eigen_herm` (*gsl_matrix_complex ** `A`, *gsl_vector ** `eval`, *gsl_eigen_herm_workspace ** `w`) [Function]

This function computes the eigenvalues of the complex hermitian matrix A. Additional workspace of the appropriate size must be provided in *w*. The diagonal and lower triangular part of A are destroyed during the computation, but the strict upper triangular part is not referenced. The imaginary parts of the diagonal are assumed to be zero and are not referenced. The eigenvalues are stored in the vector *eval* and are unordered.

`gsl_eigen_hermv_workspace * gsl_eigen_hermv_alloc` (*const size_t* `n`) [Function]

This function allocates a workspace for computing eigenvalues and eigenvectors of n-by-n complex hermitian matrices. The size of the workspace is $O(5n)$.

`void gsl_eigen_hermv_free` (*gsl_eigen_hermv_workspace ** `w`) [Function]

This function frees the memory associated with the workspace *w*.

`int gsl_eigen_hermv` (*gsl_matrix_complex ** `A`, *gsl_vector ** `eval`, *gsl_matrix_complex ** `evec`, *gsl_eigen_hermv_workspace ** `w`) [Function]

This function computes the eigenvalues and eigenvectors of the complex hermitian matrix A. Additional workspace of the appropriate size must be provided in *w*. The diagonal and lower triangular part of A are destroyed during the computation, but the strict upper triangular part is not referenced. The imaginary parts of the diagonal are assumed to be zero and are not referenced. The eigenvalues are stored in the vector *eval* and are unordered. The corresponding complex eigenvectors are stored in the columns of the matrix *evec*. For example, the eigenvector in the first column corresponds to the first eigenvalue. The eigenvectors are guaranteed to be mutually orthogonal and normalised to unit magnitude.

15.3 Real Nonsymmetric Matrices

The solution of the real nonsymmetric eigensystem problem for a matrix A involves computing the Schur decomposition

$$A = ZTZ^T$$

where Z is an orthogonal matrix of Schur vectors and T, the Schur form, is quasi upper triangular with diagonal 1-by-1 blocks which are real eigenvalues of A, and diagonal 2-by-2 blocks whose eigenvalues are complex conjugate eigenvalues of A. The algorithm used is the double-shift Francis method.

`gsl_eigen_nonsymm_workspace * gsl_eigen_nonsymm_alloc` (*const size_t* `n`) [Function]

This function allocates a workspace for computing eigenvalues of n-by-n real nonsymmetric matrices. The size of the workspace is $O(2n)$.

`void gsl_eigen_nonsymm_free` (*gsl_eigen_nonsymm_workspace ** `w`) [Function]

This function frees the memory associated with the workspace *w*.

`void gsl_eigen_nonsymm_params` (*const int* `compute_t`, *const int* `balance`, *gsl_eigen_nonsymm_workspace ** `w`) [Function]

This function sets some parameters which determine how the eigenvalue problem is solved in subsequent calls to `gsl_eigen_nonsymm`.

If *compute_t* is set to 1, the full Schur form T will be computed by `gsl_eigen_nonsymm`. If it is set to 0, T will not be computed (this is the default setting). Computing the full Schur form T requires approximately 1.5–2 times the number of flops.

If *balance* is set to 1, a balancing transformation is applied to the matrix prior to computing eigenvalues. This transformation is designed to make the rows and columns of the matrix have comparable norms, and can result in more accurate eigenvalues for matrices whose entries vary widely in magnitude. See Section 14.14 [Balancing], page 146 for more information. Note that the balancing transformation does not preserve the orthogonality of the Schur vectors, so if you wish to compute the Schur vectors with `gsl_eigen_nonsymm_Z` you will obtain the Schur vectors of the balanced matrix instead of the original matrix. The relationship will be

$$T = Q^t D^{-1} A D Q$$

where Q is the matrix of Schur vectors for the balanced matrix, and D is the balancing transformation. Then `gsl_eigen_nonsymm_Z` will compute a matrix Z which satisfies

$$T = Z^{-1} A Z$$

with $Z = DQ$. Note that Z will not be orthogonal. For this reason, balancing is not performed by default.

`int gsl_eigen_nonsymm` (*gsl_matrix ** `A`, *gsl_vector_complex ** `eval`, *gsl_eigen_nonsymm_workspace ** `w`) [Function]

This function computes the eigenvalues of the real nonsymmetric matrix A and stores them in the vector *eval*. If T is desired, it is stored in the upper portion of A on output. Otherwise, on output, the diagonal of A will contain the 1-by-1 real eigenvalues and 2-by-2 complex conjugate eigenvalue systems, and the rest of A is destroyed. In rare cases, this function may fail to find all eigenvalues. If this happens, an error code is returned and the number of converged eigenvalues is stored in `w->n_evals`. The converged eigenvalues are stored in the beginning of *eval*.

`int gsl_eigen_nonsymm_Z` (*gsl_matrix * `A`, gsl_vector_complex * `eval`, gsl_matrix * `Z`, gsl_eigen_nonsymm_workspace * `w`*) [Function]

This function is identical to `gsl_eigen_nonsymm` except that it also computes the Schur vectors and stores them into *Z*.

`gsl_eigen_nonsymmv_workspace * gsl_eigen_nonsymmv_alloc` (*const size_t `n`*) [Function]

This function allocates a workspace for computing eigenvalues and eigenvectors of *n*-by-*n* real nonsymmetric matrices. The size of the workspace is $O(5n)$.

`void gsl_eigen_nonsymmv_free` (*gsl_eigen_nonsymmv_workspace * `w`*) [Function]

This function frees the memory associated with the workspace *w*.

`void gsl_eigen_nonsymmv_params` (*const int `balance`, gsl_eigen_nonsymm_workspace * `w`*) [Function]

This function sets parameters which determine how the eigenvalue problem is solved in subsequent calls to `gsl_eigen_nonsymmv`. If *balance* is set to 1, a balancing transformation is applied to the matrix. See `gsl_eigen_nonsymm_params` for more information. Balancing is turned off by default since it does not preserve the orthogonality of the Schur vectors.

`int gsl_eigen_nonsymmv` (*gsl_matrix * `A`, gsl_vector_complex * `eval`, gsl_matrix_complex * `evec`, gsl_eigen_nonsymmv_workspace * `w`*) [Function]

This function computes eigenvalues and right eigenvectors of the *n*-by-*n* real nonsymmetric matrix A. It first calls `gsl_eigen_nonsymm` to compute the eigenvalues, Schur form T, and Schur vectors. Then it finds eigenvectors of T and backtransforms them using the Schur vectors. The Schur vectors are destroyed in the process, but can be saved by using `gsl_eigen_nonsymmv_Z`. The computed eigenvectors are normalized to have unit magnitude. On output, the upper portion of A contains the Schur form T. If `gsl_eigen_nonsymm` fails, no eigenvectors are computed, and an error code is returned.

`int gsl_eigen_nonsymmv_Z` (*gsl_matrix * `A`, gsl_vector_complex * `eval`, gsl_matrix_complex * `evec`, gsl_matrix * `Z`, gsl_eigen_nonsymmv_workspace * `w`*) [Function]

This function is identical to `gsl_eigen_nonsymmv` except that it also saves the Schur vectors into *Z*.

15.4 Real Generalized Symmetric-Definite Eigensystems

The real generalized symmetric-definite eigenvalue problem is to find eigenvalues λ and eigenvectors x such that

$$Ax = \lambda Bx$$

where A and B are symmetric matrices, and B is positive-definite. This problem reduces to the standard symmetric eigenvalue problem by applying the Cholesky decomposition to

B:

$$\begin{aligned} Ax &= \lambda Bx \\ Ax &= \lambda LL^t x \\ \left(L^{-1}AL^{-t}\right) L^t x &= \lambda L^t x \end{aligned}$$

Therefore, the problem becomes $Cy = \lambda y$ where $C = L^{-1}AL^{-t}$ is symmetric, and $y = L^t x$. The standard symmetric eigensolver can be applied to the matrix C. The resulting eigenvectors are backtransformed to find the vectors of the original problem. The eigenvalues and eigenvectors of the generalized symmetric-definite eigenproblem are always real.

`gsl_eigen_gensymm_workspace * gsl_eigen_gensymm_alloc` (*const size_t* `n`) [Function]

This function allocates a workspace for computing eigenvalues of *n*-by-*n* real generalized symmetric-definite eigensystems. The size of the workspace is $O(2n)$.

`void gsl_eigen_gensymm_free` (*gsl_eigen_gensymm_workspace ** `w`) [Function]

This function frees the memory associated with the workspace *w*.

`int gsl_eigen_gensymm` (*gsl_matrix ** `A`, *gsl_matrix ** `B`, *gsl_vector ** `eval`, *gsl_eigen_gensymm_workspace ** `w`) [Function]

This function computes the eigenvalues of the real generalized symmetric-definite matrix pair (A, B), and stores them in *eval*, using the method outlined above. On output, B contains its Cholesky decomposition and A is destroyed.

`gsl_eigen_gensymmv_workspace * gsl_eigen_gensymmv_alloc` (*const size_t* `n`) [Function]

This function allocates a workspace for computing eigenvalues and eigenvectors of *n*-by-*n* real generalized symmetric-definite eigensystems. The size of the workspace is $O(4n)$.

`void gsl_eigen_gensymmv_free` (*gsl_eigen_gensymmv_workspace ** `w`) [Function]

This function frees the memory associated with the workspace *w*.

`int gsl_eigen_gensymmv` (*gsl_matrix ** `A`, *gsl_matrix ** `B`, *gsl_vector ** `eval`, *gsl_matrix ** `evec`, *gsl_eigen_gensymmv_workspace ** `w`) [Function]

This function computes the eigenvalues and eigenvectors of the real generalized symmetric-definite matrix pair (A, B), and stores them in *eval* and *evec* respectively. The computed eigenvectors are normalized to have unit magnitude. On output, B contains its Cholesky decomposition and A is destroyed.

15.5 Complex Generalized Hermitian-Definite Eigensystems

The complex generalized hermitian-definite eigenvalue problem is to find eigenvalues λ and eigenvectors x such that

$$Ax = \lambda Bx$$

where A and B are hermitian matrices, and B is positive-definite. Similarly to the real case, this can be reduced to $Cy = \lambda y$ where $C = L^{-1}AL^{-\dagger}$ is hermitian, and $y = L^\dagger x$. The standard hermitian eigensolver can be applied to the matrix C. The resulting eigenvectors are backtransformed to find the vectors of the original problem. The eigenvalues of the generalized hermitian-definite eigenproblem are always real.

`gsl_eigen_genherm_workspace * gsl_eigen_genherm_alloc` (*const size_t* `n`) [Function]

This function allocates a workspace for computing eigenvalues of n-by-n complex generalized hermitian-definite eigensystems. The size of the workspace is $O(3n)$.

`void gsl_eigen_genherm_free` (*gsl_eigen_genherm_workspace ** `w`) [Function]

This function frees the memory associated with the workspace *w*.

`int gsl_eigen_genherm` (*gsl_matrix_complex ** `A`, *gsl_matrix_complex ** `B`, *gsl_vector ** `eval`, *gsl_eigen_genherm_workspace ** `w`) [Function]

This function computes the eigenvalues of the complex generalized hermitian-definite matrix pair (A, B), and stores them in *eval*, using the method outlined above. On output, B contains its Cholesky decomposition and A is destroyed.

`gsl_eigen_genhermv_workspace * gsl_eigen_genhermv_alloc` (*const size_t* `n`) [Function]

This function allocates a workspace for computing eigenvalues and eigenvectors of n-by-n complex generalized hermitian-definite eigensystems. The size of the workspace is $O(5n)$.

`void gsl_eigen_genhermv_free` (*gsl_eigen_genhermv_workspace ** `w`) [Function]

This function frees the memory associated with the workspace *w*.

`int gsl_eigen_genhermv` (*gsl_matrix_complex ** `A`, *gsl_matrix_complex ** `B`, *gsl_vector ** `eval`, *gsl_matrix_complex ** `evec`, *gsl_eigen_genhermv_workspace ** `w`) [Function]

This function computes the eigenvalues and eigenvectors of the complex generalized hermitian-definite matrix pair (A, B), and stores them in *eval* and *evec* respectively. The computed eigenvectors are normalized to have unit magnitude. On output, B contains its Cholesky decomposition and A is destroyed.

15.6 Real Generalized Nonsymmetric Eigensystems

Given two square matrices (A, B), the generalized nonsymmetric eigenvalue problem is to find eigenvalues λ and eigenvectors x such that

$$Ax = \lambda Bx$$

We may also define the problem as finding eigenvalues μ and eigenvectors y such that

$$\mu Ay = By$$

Note that these two problems are equivalent (with $\lambda = 1/\mu$) if neither λ nor μ is zero. If say, λ is zero, then it is still a well defined eigenproblem, but its alternate problem involving μ is not. Therefore, to allow for zero (and infinite) eigenvalues, the problem which is actually solved is

$$\beta Ax = \alpha Bx$$

The eigensolver routines below will return two values α and β and leave it to the user to perform the divisions $\lambda = \alpha/\beta$ and $\mu = \beta/\alpha$.

If the determinant of the matrix pencil $A - \lambda B$ is zero for all λ, the problem is said to be singular; otherwise it is called regular. Singularity normally leads to some $\alpha = \beta = 0$ which means the eigenproblem is ill-conditioned and generally does not have well defined eigenvalue solutions. The routines below are intended for regular matrix pencils and could yield unpredictable results when applied to singular pencils.

The solution of the real generalized nonsymmetric eigensystem problem for a matrix pair (A, B) involves computing the generalized Schur decomposition

$$A = QSZ^T$$

$$B = QTZ^T$$

where Q and Z are orthogonal matrices of left and right Schur vectors respectively, and (S, T) is the generalized Schur form whose diagonal elements give the α and β values. The algorithm used is the QZ method due to Moler and Stewart (see references).

`gsl_eigen_gen_workspace * gsl_eigen_gen_alloc` (*const size_t* `n`) [Function]

This function allocates a workspace for computing eigenvalues of *n*-by-*n* real generalized nonsymmetric eigensystems. The size of the workspace is $O(n)$.

`void gsl_eigen_gen_free` (*gsl_eigen_gen_workspace ** `w`) [Function]

This function frees the memory associated with the workspace *w*.

`void gsl_eigen_gen_params` (*const int* `compute_s`, *const int* `compute_t`, *const int* `balance`, *gsl_eigen_gen_workspace ** `w`) [Function]

This function sets some parameters which determine how the eigenvalue problem is solved in subsequent calls to `gsl_eigen_gen`.

If *compute_s* is set to 1, the full Schur form S will be computed by `gsl_eigen_gen`. If it is set to 0, S will not be computed (this is the default setting). S is a quasi upper triangular matrix with 1-by-1 and 2-by-2 blocks on its diagonal. 1-by-1 blocks correspond to real eigenvalues, and 2-by-2 blocks correspond to complex eigenvalues.

If *compute_t* is set to 1, the full Schur form T will be computed by `gsl_eigen_gen`. If it is set to 0, T will not be computed (this is the default setting). T is an upper triangular matrix with non-negative elements on its diagonal. Any 2-by-2 blocks in S will correspond to a 2-by-2 diagonal block in T.

The *balance* parameter is currently ignored, since generalized balancing is not yet implemented.

`int gsl_eigen_gen` (*gsl_matrix ** `A`, *gsl_matrix ** `B`, *gsl_vector_complex ** `alpha`, *gsl_vector ** `beta`, *gsl_eigen_gen_workspace ** `w`) [Function]

This function computes the eigenvalues of the real generalized nonsymmetric matrix pair (A, B), and stores them as pairs in (*alpha*, *beta*), where *alpha* is complex and *beta* is real. If β_i is non-zero, then $\lambda = \alpha_i/\beta_i$ is an eigenvalue. Likewise, if α_i is non-zero, then $\mu = \beta_i/\alpha_i$ is an eigenvalue of the alternate problem $\mu Ay = By$. The elements of *beta* are normalized to be non-negative.

If S is desired, it is stored in A on output. If T is desired, it is stored in B on output. The ordering of eigenvalues in (*alpha*, *beta*) follows the ordering of the diagonal blocks in the Schur forms S and T. In rare cases, this function may fail to find all eigenvalues. If this occurs, an error code is returned.

`int gsl_eigen_gen_QZ` (*gsl_matrix ** `A`, *gsl_matrix ** `B`, *gsl_vector_complex ** `alpha`, *gsl_vector ** `beta`, *gsl_matrix ** `Q`, *gsl_matrix ** `Z`, *gsl_eigen_gen_workspace ** `w`) [Function]

This function is identical to `gsl_eigen_gen` except that it also computes the left and right Schur vectors and stores them into Q and Z respectively.

`gsl_eigen_genv_workspace * gsl_eigen_genv_alloc` (*const size_t* `n`) [Function]

This function allocates a workspace for computing eigenvalues and eigenvectors of n-by-n real generalized nonsymmetric eigensystems. The size of the workspace is $O(7n)$.

`void gsl_eigen_genv_free` (*gsl_eigen_genv_workspace ** `w`) [Function]

This function frees the memory associated with the workspace *w*.

`int gsl_eigen_genv` (*gsl_matrix ** `A`, *gsl_matrix ** `B`, *gsl_vector_complex ** `alpha`, *gsl_vector ** `beta`, *gsl_matrix_complex ** `evec`, *gsl_eigen_genv_workspace ** `w`) [Function]

This function computes eigenvalues and right eigenvectors of the n-by-n real generalized nonsymmetric matrix pair (A, B). The eigenvalues are stored in (*alpha*, *beta*) and the eigenvectors are stored in *evec*. It first calls `gsl_eigen_gen` to compute the eigenvalues, Schur forms, and Schur vectors. Then it finds eigenvectors of the Schur forms and backtransforms them using the Schur vectors. The Schur vectors are destroyed in the process, but can be saved by using `gsl_eigen_genv_QZ`. The computed eigenvectors are normalized to have unit magnitude. On output, (A, B) contains the generalized Schur form (S, T). If `gsl_eigen_gen` fails, no eigenvectors are computed, and an error code is returned.

`int gsl_eigen_genv_QZ` (*gsl_matrix ** `A`, *gsl_matrix ** `B`, *gsl_vector_complex ** `alpha`, *gsl_vector ** `beta`, *gsl_matrix_complex ** `evec`, *gsl_matrix ** `Q`, *gsl_matrix ** `Z`, *gsl_eigen_genv_workspace ** `w`) [Function]

This function is identical to `gsl_eigen_genv` except that it also computes the left and right Schur vectors and stores them into Q and Z respectively.

15.7 Sorting Eigenvalues and Eigenvectors

`int gsl_eigen_symmv_sort` (*gsl_vector ** `eval`, *gsl_matrix ** `evec`, *gsl_eigen_sort_t* `sort_type`) [Function]

This function simultaneously sorts the eigenvalues stored in the vector *eval* and the corresponding real eigenvectors stored in the columns of the matrix *evec* into ascending or descending order according to the value of the parameter *sort_type*,

`GSL_EIGEN_SORT_VAL_ASC`
    ascending order in numerical value

`GSL_EIGEN_SORT_VAL_DESC`
: descending order in numerical value

`GSL_EIGEN_SORT_ABS_ASC`
: ascending order in magnitude

`GSL_EIGEN_SORT_ABS_DESC`
: descending order in magnitude

`int gsl_eigen_hermv_sort` (*gsl_vector * `eval`, gsl_matrix_complex * `evec`, gsl_eigen_sort_t `sort_type`*) [Function]

This function simultaneously sorts the eigenvalues stored in the vector *eval* and the corresponding complex eigenvectors stored in the columns of the matrix *evec* into ascending or descending order according to the value of the parameter *sort_type* as shown above.

`int gsl_eigen_nonsymmv_sort` (*gsl_vector_complex * `eval`, gsl_matrix_complex * `evec`, gsl_eigen_sort_t `sort_type`*) [Function]

This function simultaneously sorts the eigenvalues stored in the vector *eval* and the corresponding complex eigenvectors stored in the columns of the matrix *evec* into ascending or descending order according to the value of the parameter *sort_type* as shown above. Only `GSL_EIGEN_SORT_ABS_ASC` and `GSL_EIGEN_SORT_ABS_DESC` are supported due to the eigenvalues being complex.

`int gsl_eigen_gensymmv_sort` (*gsl_vector * `eval`, gsl_matrix * `evec`, gsl_eigen_sort_t `sort_type`*) [Function]

This function simultaneously sorts the eigenvalues stored in the vector *eval* and the corresponding real eigenvectors stored in the columns of the matrix *evec* into ascending or descending order according to the value of the parameter *sort_type* as shown above.

`int gsl_eigen_genhermv_sort` (*gsl_vector * `eval`, gsl_matrix_complex * `evec`, gsl_eigen_sort_t `sort_type`*) [Function]

This function simultaneously sorts the eigenvalues stored in the vector *eval* and the corresponding complex eigenvectors stored in the columns of the matrix *evec* into ascending or descending order according to the value of the parameter *sort_type* as shown above.

`int gsl_eigen_genv_sort` (*gsl_vector_complex * `alpha`, gsl_vector * `beta`, gsl_matrix_complex * `evec`, gsl_eigen_sort_t `sort_type`*) [Function]

This function simultaneously sorts the eigenvalues stored in the vectors (*alpha*, *beta*) and the corresponding complex eigenvectors stored in the columns of the matrix *evec* into ascending or descending order according to the value of the parameter *sort_type* as shown above. Only `GSL_EIGEN_SORT_ABS_ASC` and `GSL_EIGEN_SORT_ABS_DESC` are supported due to the eigenvalues being complex.

15.8 Examples

The following program computes the eigenvalues and eigenvectors of the 4-th order Hilbert matrix, $H(i,j) = 1/(i + j + 1)$.

```
#include <stdio.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_eigen.h>

int
main (void)
{
  double data[] = { 1.0  , 1/2.0, 1/3.0, 1/4.0,
                    1/2.0, 1/3.0, 1/4.0, 1/5.0,
                    1/3.0, 1/4.0, 1/5.0, 1/6.0,
                    1/4.0, 1/5.0, 1/6.0, 1/7.0 };

  gsl_matrix_view m
    = gsl_matrix_view_array (data, 4, 4);

  gsl_vector *eval = gsl_vector_alloc (4);
  gsl_matrix *evec = gsl_matrix_alloc (4, 4);

  gsl_eigen_symmv_workspace * w =
    gsl_eigen_symmv_alloc (4);

  gsl_eigen_symmv (&m.matrix, eval, evec, w);

  gsl_eigen_symmv_free (w);

  gsl_eigen_symmv_sort (eval, evec,
                        GSL_EIGEN_SORT_ABS_ASC);

  {
    int i;

    for (i = 0; i < 4; i++)
      {
        double eval_i
           = gsl_vector_get (eval, i);
        gsl_vector_view evec_i
           = gsl_matrix_column (evec, i);

        printf ("eigenvalue = %g\n", eval_i);
        printf ("eigenvector = \n");
        gsl_vector_fprintf (stdout,
                            &evec_i.vector, "%g");
      }
  }

  gsl_vector_free (eval);
  gsl_matrix_free (evec);
  return 0;
}
```

Here is the beginning of the output from the program,

```
$ ./a.out
eigenvalue = 9.67023e-05
eigenvector =
-0.0291933
0.328712
-0.791411
0.514553
...
```

This can be compared with the corresponding output from GNU OCTAVE,

```
octave> [v,d] = eig(hilb(4));
octave> diag(d)
ans =

   9.6702e-05
   6.7383e-03
   1.6914e-01
   1.5002e+00

octave> v
v =

   0.029193   0.179186  -0.582076   0.792608
  -0.328712  -0.741918   0.370502   0.451923
   0.791411   0.100228   0.509579   0.322416
  -0.514553   0.638283   0.514048   0.252161
```

Note that the eigenvectors can differ by a change of sign, since the sign of an eigenvector is arbitrary.

The following program illustrates the use of the nonsymmetric eigensolver, by computing the eigenvalues and eigenvectors of the Vandermonde matrix $V(x;i,j) = x_i^{n-j}$ with $x = (-1,-2,3,4)$.

```
#include <stdio.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_eigen.h>

int
main (void)
{
  double data[] = { -1.0, 1.0, -1.0, 1.0,
                    -8.0, 4.0, -2.0, 1.0,
                    27.0, 9.0, 3.0, 1.0,
                    64.0, 16.0, 4.0, 1.0 };
```

```
    gsl_matrix_view m
      = gsl_matrix_view_array (data, 4, 4);

    gsl_vector_complex *eval = gsl_vector_complex_alloc (4);
    gsl_matrix_complex *evec = gsl_matrix_complex_alloc (4, 4);

    gsl_eigen_nonsymmv_workspace * w =
      gsl_eigen_nonsymmv_alloc (4);

    gsl_eigen_nonsymmv (&m.matrix, eval, evec, w);

    gsl_eigen_nonsymmv_free (w);

    gsl_eigen_nonsymmv_sort (eval, evec,
                             GSL_EIGEN_SORT_ABS_DESC);

    {
      int i, j;

      for (i = 0; i < 4; i++)
        {
          gsl_complex eval_i
             = gsl_vector_complex_get (eval, i);
          gsl_vector_complex_view evec_i
             = gsl_matrix_complex_column (evec, i);

          printf ("eigenvalue = %g + %gi\n",
                  GSL_REAL(eval_i), GSL_IMAG(eval_i));
          printf ("eigenvector = \n");
          for (j = 0; j < 4; ++j)
            {
              gsl_complex z =
                gsl_vector_complex_get(&evec_i.vector, j);
              printf("%g + %gi\n", GSL_REAL(z), GSL_IMAG(z));
            }
        }
    }

    gsl_vector_complex_free(eval);
    gsl_matrix_complex_free(evec);

    return 0;
  }
```

Here is the beginning of the output from the program,

```
  $ ./a.out
```

```
eigenvalue = -6.41391 + 0i
eigenvector =
-0.0998822 + 0i
-0.111251 + 0i
0.292501 + 0i
0.944505 + 0i
eigenvalue = 5.54555 + 3.08545i
eigenvector =
-0.043487 + -0.0076308i
0.0642377 + -0.142127i
-0.515253 + 0.0405118i
-0.840592 + -0.00148565i
...
```

This can be compared with the corresponding output from GNU OCTAVE,

```
octave> [v,d] = eig(vander([-1 -2 3 4]));
octave> diag(d)
ans =

  -6.4139 + 0.0000i
   5.5456 + 3.0854i
   5.5456 - 3.0854i
   2.3228 + 0.0000i

octave> v
v =

 Columns 1 through 3:

  -0.09988 + 0.00000i  -0.04350 - 0.00755i  -0.04350 + 0.00755i
  -0.11125 + 0.00000i   0.06399 - 0.14224i   0.06399 + 0.14224i
   0.29250 + 0.00000i  -0.51518 + 0.04142i  -0.51518 - 0.04142i
   0.94451 + 0.00000i  -0.84059 + 0.00000i  -0.84059 - 0.00000i

 Column 4:

  -0.14493 + 0.00000i
   0.35660 + 0.00000i
   0.91937 + 0.00000i
   0.08118 + 0.00000i
```

Note that the eigenvectors corresponding to the eigenvalue $5.54555 + 3.08545i$ differ by the multiplicative constant $0.9999984 + 0.0017674i$ which is an arbitrary phase factor of magnitude 1.

15.9 References and Further Reading

Further information on the algorithms described in this section can be found in the following book,

> G. H. Golub, C. F. Van Loan, *Matrix Computations* (3rd Ed, 1996), Johns Hopkins University Press, ISBN 0-8018-5414-8.

Further information on the generalized eigensystems QZ algorithm can be found in this paper,

> C. Moler, G. Stewart, "An Algorithm for Generalized Matrix Eigenvalue Problems", SIAM J. Numer. Anal., Vol 10, No 2, 1973.

Eigensystem routines for very large matrices can be found in the Fortran library LAPACK. The LAPACK library is described in,

> *LAPACK Users' Guide* (Third Edition, 1999), Published by SIAM, ISBN 0-89871-447-8.
>
> `http://www.netlib.org/lapack`

The LAPACK source code can be found at the website above along with an online copy of the users guide.