NAG Library Function Document

nag_real_sparse_eigensystem_iter (f12abc)

Note: this function uses optional arguments to define choices in the problem specification. If you wish to use default settings for all of the optional arguments, then the option setting function nag_real_sparse_eigensystem_option (f12adc) need not be called. If, however, you wish to reset some or all of the settings please refer to Section 11 in nag_real_sparse_eigensystem_option (f12adc) for a detailed description of the specification of the optional arguments.

1 Purpose

nag_real_sparse_eigensystem_iter (f12abc) is an iterative solver used to find some of the eigenvalues (and optionally the corresponding eigenvectors) of a standard or generalized eigenvalue problem defined by real nonsymmetric matrices. This is part of a suite of functions that also includes nag_real_sparse_eigensystem_init (f12aac), nag_real_sparse_eigensystem_sol (f12acc), nag_real_sparse_eigensystem_option (f12adc) and nag_real_sparse_eigensystem_monit (f12aec). It is

2 Specification

#include <nag.h>
#include <nagf12.h>

void nag_real_sparse_eigensystem_iter (Integer *irevcm, double resid[], double v[], double **x, double **y, double **mx, Integer *nshift, double comm[], Integer icomm[], NagError *fail)

3 Description

The suite of functions is designed to calculate some of the eigenvalues, \( \lambda \), (and optionally the corresponding eigenvectors, \( x \)) of a standard eigenvalue problem \( Ax = \lambda x \), or of a generalized eigenvalue problem \( Ax = \lambda Bx \) of order \( n \), where \( n \) is large and the coefficient matrices \( A \) and \( B \) are sparse, real and nonsymmetric. The suite can also be used to find selected eigenvalues/eigenvectors of smaller scale dense, real and nonsymmetric problems.

nag_real_sparse_eigensystem_iter (f12abc) is a reverse communication function, based on the ARPACK routine dnaupd, using the Implicitly Restarted Arnoldi iteration method. The method is described in Lehoucq and Sorensen (1996) and Lehoucq (2001) while its use within the ARPACK software is described in great detail in Lehoucq et al. (1998). An evaluation of software for computing eigenvalues of sparse nonsymmetric matrices is provided in Lehoucq and Scott (1996). This suite of functions offers the same functionality as the ARPACK software for real nonsymmetric problems, but the interface design is quite different in order to make the option setting clearer and to simplify the interface of nag_real_sparse_eigensystem_iter (f12abc).

The setup function nag_real_sparse_eigensystem_init (f12aac) must be called before nag_real_sparse_eigensystem_iter (f12abc), the reverse communication iterative solver. Options may be set for nag_real_sparse_eigensystem_iter (f12abc) by prior calls to the option setting function nag_real_sparse_eigensystem_option (f12adc) and a post-processing function nag_real_sparse_eigensystem_sol (f12acc) must be called following a successful final exit from nag_real_sparse_eigensystem_iter (f12abc). nag_real_sparse_eigensystem_monit (f12aec), may be called following certain flagged, intermediate exits from nag_real_sparse_eigensystem_iter (f12abc) to provide additional monitoring information about the computation.

nag_real_sparse_eigensystem_iter (f12abc) uses reverse communication, i.e., it returns repeatedly to the calling program with the argument irevcm (see Section 5) set to specified values which require the calling program to carry out one of the following tasks:

- compute the matrix-vector product \( y = OPx \), where \( OP \) is defined by the computational mode;
- compute the matrix-vector product \( y = Bx \);
notify the completion of the computation;

– allow the calling program to monitor the solution.

The problem type to be solved (standard or generalized), the spectrum of eigenvalues of interest, the mode used (regular, regular inverse, shifted inverse, shifted real or shifted imaginary) and other options can all be set using the option setting function nag_real_sparse_eigensystem_option (f12adc) (see Section 11.1 in nag_real_sparse_eigensystem_option (f12adc) for details on setting options and of the default settings).

4 References


5 Arguments

Note: this function uses reverse communication. Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the argument irevcm. Between intermediate exits and re-entries, all arguments other than x and y must remain unchanged.

1: irevcm – Integer *

Input/Output

On initial entry: irevcm = 0, otherwise an error condition will be raised.

On intermediate re-entry: must be unchanged from its previous exit value. Changing irevcm to any other value between calls will result in an error.

On intermediate exit: has the following meanings.

irevcm = −1

The calling program must compute the matrix-vector product y = OPx, where x is stored in x and the result y is placed in y. If B is not symmetric semi-definite then the precomputed values in mx should not be used (see the explanation under irevcm = 2).

irevcm = 1

The calling program must compute the matrix-vector product y = OPx. This is similar to the case irevcm = −1 except that the result of the matrix-vector product Bx (as required in some computational modes) has already been computed and is available in mx.

irevcm = 2

The calling program must compute the matrix-vector product y = Bx, where x is stored as described in the case irevcm = −1 and y is placed in y. This computation is requested when solving the Generalized problem using either Shifted Inverse Imaginary or Shifted Inverse Real; in these cases B is used as an inner-product space and requires that B be symmetric semi-definite. If neither A nor B is symmetric semi-definite then the problem should be reformulated in a Standard form.

irevcm = 3

Compute the nshift real and imaginary parts of the shifts where the real parts are to be placed in the first nshift locations of the array y and the imaginary parts are to be placed in the first nshift locations of the array mx. Only complex conjugate pairs of shifts may be applied and the pairs must be placed in consecutive locations. This value of irevcm will only arise if the optional argument Supplied Shifts is set in a prior call to nag_real_sparse_eigensystem_option (f12adc) which is intended for experienced users...
only; the default and recommended option is to use exact shifts (see Lehoucq et al. (1998) for details).

\( \texttt{irevcm} = 4 \)

Monitoring step: a call to \texttt{nag_real_sparse_eigensystem_monit (f12aec)} can now be made to return the number of Arnoldi iterations, the number of converged Ritz values, their real and imaginary parts, and the corresponding Ritz estimates.

On final exit: \( \texttt{irevcm} = 5 \): \texttt{nag_real_sparse_eigensystem_iter (f12abc)} has completed its tasks. The value of \texttt{fail} determines whether the iteration has been successfully completed, or whether errors have been detected. On successful completion \texttt{nag_real_sparse_eigensystem_sol (f12acc)} must be called to return the requested eigenvalues and eigenvectors (and/or Schur vectors).

Constraint: on initial entry, \( \texttt{irevcm} = 0 \); on re-entry \( \texttt{irevcm} \) must remain unchanged.

2: \( \texttt{resid}[\text{dim}] \) – double

Input/Output

Note: the dimension, \( \text{dim} \), of the array \( \text{resid} \) must be at least \( n \) (see \texttt{nag_real_sparse_eigensystem_init (f12aac)}).

On initial entry: need not be set unless the option Initial Residual has been set in a prior call to \texttt{nag_real_sparse_eigensystem_option (f12adc)} in which case \( \text{resid} \) should contain an initial residual vector, possibly from a previous run.

On intermediate re-entry: must be unchanged from its previous exit. Changing \( \text{resid} \) to any other value between calls may result in an error exit.

On intermediate exit: contains the current residual vector.

On final exit: contains the final residual vector.

3: \( \texttt{v}[n \times \text{ncv}] \) – double

Input/Output

The \( i \)th element of the \( j \)th basis vector is stored in location \( \texttt{v}[n \times (i - 1) + j - 1] \), for \( i = 1,2,\ldots,n \) and \( j = 1,2,\ldots,\text{ncv} \).

On initial entry: need not be set.

On intermediate re-entry: must be unchanged from its previous exit.

On intermediate exit: contains the current set of Arnoldi basis vectors.

On final exit: contains the final set of Arnoldi basis vectors.

4: \( \texttt{x} \) – double **

Input/Output

On initial entry: need not be set, it is used as a convenient mechanism for accessing elements of \texttt{comm}.

On intermediate re-entry: is not normally changed.

On intermediate exit: contains the vector \( x \) when \( \texttt{irevcm} \) returns the value \(-1\), \(+1\) or \(2\).

On final exit: does not contain useful data.

5: \( \texttt{y} \) – double **

Input/Output

On initial entry: need not be set, it is used as a convenient mechanism for accessing elements of \texttt{comm}.

On intermediate re-entry: must contain the result of \( y = OPx \) when \( \texttt{irevcm} \) returns the value \(-1\) or \(+1\). It must contain the real parts of the computed shifts when \( \texttt{irevcm} \) returns the value \(3\).

On intermediate exit: does not contain useful data.

On final exit: does not contain useful data.
6:  \textit{mx} – double \*
\hspace{1em} \textit{Input/Output}

\textit{On initial entry:} need not be set, it is used as a convenient mechanism for accessing elements of \textit{comm}.

\textit{On intermediate re-entry:} must contain the result of \( y = Bx \) when \textit{irevcm} returns the value 2. It must contain the imaginary parts of the computed shifts when \textit{irevcm} returns the value 3.

\textit{On intermediate exit:} contains the vector \( Bx \) when \textit{irevcm} returns the value +1.

\textit{On final exit:} does not contain any useful data.

7:  \textit{nshift} – Integer *
\hspace{1em} \textit{Output}

\textit{On intermediate exit:} if the option \textbf{Supplied Shifts} is set and \textit{irevcm} returns a value of 3, \textit{nshift} returns the number of complex shifts required.

8:  \textit{comm}[\textit{dim}] – double
\hspace{1em} \textit{Communication Array}

\textbf{Note:} the dimension, \textit{dim}, of the array \textit{comm} must be at least \( \max(1, lcomm) \) (see \textit{nag\_real\_sparse\_eigensystem\_init (f12aac)}).

\textit{On initial entry:} must remain unchanged following a call to the setup function \textit{nag\_real\_sparse\_eigensystem\_init (f12aac)}.

\textit{On exit:} contains data defining the current state of the iterative process.

9:  \textit{icomm}[\textit{dim}] – Integer
\hspace{1em} \textit{Communication Array}

\textbf{Note:} the dimension, \textit{dim}, of the array \textit{icomm} must be at least \( \max(1, lcomm) \) (see \textit{nag\_real\_sparse\_eigensystem\_init (f12aac)}).

\textit{On initial entry:} must remain unchanged following a call to the setup function \textit{nag\_real\_sparse\_eigensystem\_init (f12aac)}.

\textit{On exit:} contains data defining the current state of the iterative process.

10: \textit{fail} – NagError *
\hspace{1em} \textit{Input/Output}

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 \textbf{Error Indicators and Warnings}

\textbf{NE_ALLOC_FAIL}

Dynamic memory allocation failed.

See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE_BAD_PARAM}

On entry, argument \textit{\langle value\rangle} had an illegal value.

\textbf{NE_INITIALIZATION}

Either the initialization function has not been called prior to the first call of this function or a communication array has become corrupted.

\textbf{NE_INT}

The maximum number of iterations \( \leq 0 \), the option \textit{Iteration Limit} has been set to \textit{\langle value\rangle}.

\textbf{NE_INTERNAL_EIGVAL_FAIL}

Error in internal call to compute eigenvalues and corresponding error bounds of the current upper Hessenberg matrix. Please contact NAG.
NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the
call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

NE_MAX_ITER

The maximum number of iterations has been reached. The maximum number of
iterations = \langle value\rangle. The number of converged eigenvalues = \langle value\rangle. The post-processing
function nag_real_sparse_eigensystem_sol (f12acc) may be called to recover the converged
eigenvalues at this point. Alternatively, the maximum number of iterations may be increased by a
call to the option setting function nag_real_sparse_eigensystem_option (f12adc) and the reverse
communication loop restarted. A large number of iterations may indicate a poor choice for the
values of nev and ncv; it is advisable to experiment with these values to reduce the number of
iterations (see nag_real_sparse_eigensystem_init (f12aac)).

NE_NO_ARNO LD I_FAC

Could not build an Arnoldi factorization. The size of the current Arnoldi factorization = \langle value\rangle.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

NE_NO_ShIFTS_APPLIED

No shifts could be applied during a cycle of the implicitly restarted Arnoldi iteration.

NE_OPT_INCOMPAT

The options Generalized and Regular are incompatible.

NE_ZERO_INIT_RESID

The option Initial Residual was selected but the starting vector held in resid is zero.

7 Accuracy

The relative accuracy of a Ritz value, \( \lambda \), is considered acceptable if its Ritz estimate \( \leq \text{Tolerance} \times |\lambda| \).
The default Tolerance used is the machine precision given by nag_machine_precision (X02AJC).

8 Parallelism and Performance

nag_real_sparse_eigensystem_iter (f12abc) is threaded by NAG for parallel execution in multithreaded
implementations of the NAG Library.

nag_real_sparse_eigensystem_iter (f12abc) makes calls to BLAS and/or LAPACK routines, which may
be threaded within the vendor library used by this implementation. Consult the documentation for the
vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the
OpenMP environment used within this function. Please also consult the Users’ Note for your
implementation for any additional implementation-specific information.

9 Further Comments

None.
10 Example

This example solves $Ax = \lambda x$ in shift-invert mode, where $A$ is obtained from the standard central difference discretization of the convection-diffusion operator $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \rho \frac{\partial u}{\partial x}$ on the unit square, with zero Dirichlet boundary conditions. The shift used is a real number.

10.1 Program Text

```c
/* nag_real_sparse_eigensystem_iter (f12abc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 8, 2005. */

#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <stdio.h>
#include <nagf12.h>
#include <nagf16.h>

static void my_dgttrf(Integer, double *, double *, double *, double *, Integer *, Integer *);
static void my_dgttrs(Integer, double *, double *, double *, double *, Integer *, double *, double *);

int main(void)
{
    /* Constants */
    Integer licomm = 140, imon = 0;
    /* Scalars */
    double h, rho, s, s1, s2, s3, estnrm, sigmai, sigmar;
    Integer exit_status, info, irevcm, j, lcomm, n, nconv, ncv;
    Integer nev, niter, nshift, nx;
    /* Nag types */
    NagError fail;
    /* Arrays */
    double *comm = 0, *dd = 0, *dl = 0, *du = 0, *du2 = 0, *eigvr = 0;
    double *eigvi = 0, *eigest = 0, *resid = 0, *v = 0;
    Integer *icomm = 0, *ipiv = 0;
    /* Pointers */
    double *mx = 0, *x = 0, *y = 0;
    exit_status = 0;
    INIT_FAIL(fail);
    printf("nag_real_sparse_eigensystem_iter (f12abc) Example Program Results\n");
    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%*[\n] ");
    #else
    scanf("%*[\n] ");
    #endif
    /* Read problem parameter values from data file. */
    #ifdef _WIN32
    scanf_s("%NAG_IFMT%NAG_IFMT%NAG_IFMT%lf%lf%lf%*[\n] ", &nx, &nev, &ncv, &rho, &sigmar, &sigmai);
    #else
    scanf("%NAG_IFMT%NAG_IFMT%NAG_IFMT%lf%lf%lf%*[\n] ", &nx, &nev, &ncv, &rho, &sigmar, &sigmai);
    #endif
    n = nx * nx;
    lcomm = 3*n + 3*ncv*ncv + 6*ncv + 60;
    /* Allocate memory */
    if (!(comm = NAG_ALLOC(lcomm, double))) |
```
!(eigvr = NAG_ALLOC(ncv, double)) ||
!(eigvi = NAG_ALLOC(ncv, double)) ||
!(eigest = NAG_ALLOC(ncv, double)) ||
!(dd = NAG_ALLOC(n, double)) ||
!(dl = NAG_ALLOC(n, double)) ||
!(du = NAG_ALLOC(n, double)) ||
!(du2 = NAG_ALLOC(n, double)) ||
!(resid = NAG_ALLOC(n, double)) ||
!(v = NAG_ALLOC(n * ncv, double)) ||
!(icomm = NAG_ALLOC(licomm, Integer)) ||
!(ipiv = NAG_ALLOC(n, Integer)))

{ printf("Allocation failure\n");
  exit_status = -1;
  goto END;
}

/* Initialise communication arrays for problem using
nag_real_sparse_eigensystem_init (f12aac). */

nag_real_sparse_eigensystem_init(n, nev, ncv, icomm, licomm,
                                 comm, lcomm, &fail);
if (fail.code != NE_NOERROR)
  { printf(
          "Error from nag_real_sparse_eigensystem_init (f12aac).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
  }

/* Select the required mode using
nag_real_sparse_eigensystem_option (f12adc). */
nag_real_sparse_eigensystem_option("SHIFTED INVERSE REAL", icomm,
                                 comm, &fail);

/* Construct C = A - sigma*I, and factorize using my_dgttrf. */

h = 1.0 / (double)(n + 1);
s = rho * h / 2.0;
s1 = -1.0 - s;
s2 = 2.0 - sigmar;
s3 = s - 1.0;
for (j = 0; j <= n - 2; ++j)
  {
    dl[j] = s1;
    dd[j] = s2;
    du[j] = s3;
  }

my_dgttrf(n, dl, dd, du, du2, ipiv, &info);
irevcm = 0;

/* Repeated calls to reverse communication routine
nag_real_sparse_eigensystem_iter (f12abc). */
nag_real_sparse_eigensystem_iter(&irevcm, resid, v, &x, &y, &mx,
                                 &nshift, comm, icomm, &fail);
if (irevcm != 5)
  {
    if (irevcm == -1 || irevcm == 1)
      {
        /* Perform y <-- OP*x = inv[A-SIGMA*I]*x. */
        /* Use my_dgttrs, a cut down C version of Lapack’s dgttrs. */
        my_dgttrs(n, dl, dd, du, du2, ipiv, x, y);
      }
    else if (irevcm == 4 && imon == 1)
      {
        /* If imon=1, get monitoring information using
         nag_real_sparse_eigensystem_monit (f12aec). */
        nag_real_sparse_eigensystem_monit(&niter, &nconv, eigvr,
                                          eigvi, eigest, icomm, comm);
        /* Compute 2-norm of Ritz estimates using
         nag_dge_norm (f16rac). */
        nag_dge_norm(Nag_ColMajor, Nag_FrobeniusNorm, nev, 1, eigest,
                     eigvi, eigest, icomm, comm);
      }
  }

Mark 25
nev, &estnrm, &fail);
printf("Iteration %3"NAG_IFMT", niter);
printf(" No. converged = %3"NAG_IFMT", nconv);
printf(" norm of estimates = %17.8e\n", estnrm);
} goto REVCOMLOOP;
}
if (fail.code == NE_NOERROR)
{
    /* Post-Process using nag_real_sparse_eigensystem_sol
       (f12acc) to compute eigenvalues/vectors. */
    nag_real_sparse_eigensystem_sol(&nconv, eigvr, eigvi, v, sigmar,
                                   sigmai, resid, v, comm, icomm,
                                   &fail);
    /* Print computed eigenvalues. */
    printf("\n");
    printf(" The %4"NAG_IFMT" Ritz values of closest",
           nconv);
    printf(" to unity are:\n\n");
    for (j = 0; j <= nconv-1; ++j)
    {
        printf("%8"NAG_IFMT"%5s( %12.4f ,%12.4f )\n", j+1,",
                eigvr[j], eigvi[j]);
    }
} else
{
    printf(" Error from nag_real_sparse_eigensystem_iter (f12abc).\\n\n", fail.message);
    exit_status = 1;
    goto END;
}
END:
NAG_FREE(comm);
NAG_FREE(eigvr);
NAG_FREE(eigvi);
NAG_FREE(eigest);
NAG_FREE(dd);
NAG_FREE(dl);
NAG_FREE(du);
NAG_FREE(du2);
NAG_FREE(resid);
NAG_FREE(v);
NAG_FREE(icomm);
NAG_FREE(ipiv);
return exit_status;
}
static void my_dgttrf(Integer n, double dl[], double d[],
                        double du[], double du2[], Integer ipiv[],
                        Integer *info)
{ /* A simple C version of the Lapack routine dgttrf with argument
    checking removed */
    /* Scalars */
    double temp, fact;
    Integer i;
    /* Function Body */
    *info = 0;
    for (i = 0; i < n; ++i)
    {
        ipiv[i] = i;
    }
    for (i = 0; i < n - 2; ++i)
    {
        du2[i] = 0.0;
    }
    for (i = 0; i < n - 2; i++)
    {
        if (fabs(d[i]) >= fabs(dl[i]))
        {  

/* No row interchange required, eliminate dl[i]. */
if (d[i] != 0.0)
{
    fact = dl[i] / d[i];
    dl[i] = fact;
    d[i+1] = d[i+1] - fact * du[i];
}
else
{
    /* Interchange rows I and I+1, eliminate dl[I] */
    fact = d[i] / dl[i];
    d[i] = dl[i];
    dl[i] = fact;
    temp = du[i];
    du[i] = d[i+1];
    d[i+1] = temp - fact*d[i+1];
    du2[i] = du[i+1];
    du[i+1] = -fact * du[i+1];
    ipiv[i] = i + 1;
}
if (n > 1)
{
    i = n - 2;
    if (fabs(d[i]) >= fabs(dl[i]))
    {
        if (d[i] != 0.0)
        {
            fact = dl[i] / d[i];
            dl[i] = fact;
            d[i+1] = d[i+1] - fact * du[i];
        }
    }
    else
    {
        fact = d[i] / dl[i];
        d[i] = dl[i];
        dl[i] = fact;
        temp = du[i];
        du[i] = d[i+1];
        d[i+1] = temp - fact * d[i+1];
        ipiv[i] = i + 1;
    }
}
/* Check for a zero on the diagonal of U. */
for (i = 0; i < n; ++i)
{
    if (d[i] == 0.0)
    {
        *info = i;
        goto END;
    }
}
END:
return;

static void my_dgttrs(Integer n, double dl[], double d[],
        double du[], double du2[], Integer ipiv[],
        double b[], double y[])
{
    /* A simple C version of the Lapack routine dgttrs with argument
       checking removed, the number of right-hand-sides=1, Trans='N' */
    /* Scalars */
    Integer i, ip;
    double temp;
    /* Solve L*X = b. */
    for (i = 0; i <= n - 1; ++i)
    {
        y[i] = b[i];
    }

Mark 25

f12abc.9
for (i = 0; i < n - 1; ++i) {
  ip = ipiv[i];
  temp = y[i+1-ip+i] - dl[i]*y[ip];
  y[i] = y[ip];
  y[i+1] = temp;
}
/* Solve U*x = b. */
y[n-1] = y[n-1] / d[n-1];
if (n > 1) {
  y[n-2] = (y[n-2] - du[n-2]*y[n-1])/d[n-2];
}
for (i = n - 3; i >= 0; --i) {
  y[i] = (y[i]-du[i]*y[i+1]-du2[i]*y[i+2])/d[i];
}
return;

10.2 Program Data

nag_real_sparse_eigensystem_iter (f12abc) Example Program Data
10 4 20 10.0 1.0 0.0 : Values for nx, nev, ncv, rho, sigmar, sigmai

10.3 Program Results

nag_real_sparse_eigensystem_iter (f12abc) Example Program Results

The 4 Ritz values of closest to unity are:

1  (  1.0192 ,  0.0000 )
2  (  0.9656 ,  0.0000 )
3  (  1.0738 ,  0.0000 )
4  (  0.9129 ,  0.0000 )