NAG Library Function Document

nag_complex_sparse_eigensystem_init (f12anc)

1 Purpose

nag_complex_sparse_eigensystem_init (f12anc) is a setup function in a suite of functions consisting of
nag_complex_sparse_eigensystem_init (f12anc), nag_complex_sparse_eigensystem_iter (f12apc),
nag_complex_sparse_eigensystem_sol (f12aqc), nag_complex_sparse_eigensystem_option (f12arc) and
nag_complex_sparse_eigensystem_monit (f12asc). It is used to find some of the eigenvalues (and
optionally the corresponding eigenvectors) of a standard or generalized eigenvalue problem defined by
complex nonsymmetric matrices.

The suite of functions is suitable for the solution of large sparse, standard or generalized, nonsymmetric
complex eigenproblems where only a few eigenvalues from a selected range of the spectrum are
required.

2 Specification

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

void nag_complex_sparse_eigensystem_init (Integer n, Integer nev,
                                       Integer ncv, Integer icomm[], Integer licomm, Complex comm[],
                                       Integer lcomm, 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 complex eigenvalue problem \( Ax = \lambda x \), or of a generalized
complex eigenvalue problem \( Ax = \lambda Bx \) of order \( n \), where \( n \) is large and the coefficient matrices \( A \) and
\( B \) are sparse, complex and nonsymmetric. The suite can also be used to find selected eigenvalues/
eigenvectors of smaller scale dense, complex and nonsymmetric problems.

nag_complex_sparse_eigensystem_init (f12anc) is a setup function which must be called before
nag_complex_sparse_eigensystem_iter (f12apc), the reverse communication iterative solver, and before
nag_complex_sparse_eigensystem_option (f12arc), the options setting function.

nag_complex_sparse_eigensystem_sol (f12aqc) is a post-processing function that must be called
following a successful final exit from nag_complex_sparse_eigensystem_iter (f12apc), while
nag_complex_sparse_eigensystem_monit (f12asc) can be used to return additional monitoring
information during the computation.

This setup function initializes the communication arrays, sets (to their default values) all options that can
be set by you via the option setting function nag_complex_sparse_eigensystem_option (f12arc), and
checks that the lengths of the communication arrays as passed by you are of sufficient length. For details
of the options available and how to set them see Section 11.1 in
nag_complex_sparse_eigensystem_option (f12arc).

4 References

Analysis and Applications 23 551–562

nonsymmetric matrices Preprint MCS-P547-1195 Argonne National Laboratory

Mark 25
5 Arguments

1: \( n \) – Integer \( \text{Input} \)

\( On\ entry: \) the order of the matrix \( A \) (and the order of the matrix \( B \) for the generalized problem) that defines the eigenvalue problem.

\( Constraint: \ n > 0. \)

2: \( nev \) – Integer \( \text{Input} \)

\( On\ entry: \) the number of eigenvalues to be computed.

\( Constraint: \ 0 < nev < n - 1. \)

3: \( ncv \) – Integer \( \text{Input} \)

\( On\ entry: \) the number of Arnoldi basis vectors to use during the computation.

At present there is no \( a\ priori \) analysis to guide the selection of \( ncv \) relative to \( nev \). However, it is recommended that \( ncv \geq 2 \times nev + 1. \) If many problems of the same type are to be solved, you should experiment with increasing \( ncv \) while keeping \( nev \) fixed for a given test problem. This will usually decrease the required number of matrix-vector operations but it also increases the work and storage required to maintain the orthogonal basis vectors. The optimal ‘cross-over’ with respect to CPU time is problem dependent and must be determined empirically.

\( Constraint: \ nev + 1 < ncv \leq n. \)

4: \( \text{icomm}[\max(1,\text{licomm})] \) – Integer \( \text{Communication Array} \)

\( On\ exit: \) contains data to be communicated to the other functions in the suite.

5: \( \text{licomm} \) – Integer \( \text{Input} \)

\( On\ entry: \) the dimension of the array \( \text{icomm} \).

If \( \text{licomm} = -1 \), a workspace query is assumed and the function only calculates the required dimensions of \( \text{icomm} \) and \( \text{comm} \), which it returns in \( \text{icomm}[0] \) and \( \text{comm}[0] \) respectively.

\( Constraint: \ \text{licomm} \geq 140 \) or \( \text{licomm} = -1. \)

6: \( \text{comm}[\max(1,\text{licomm})] \) – Complex \( \text{Communication Array} \)

\( On\ exit: \) contains data to be communicated to the other functions in the suite.

7: \( \text{lcomm} \) – Integer \( \text{Input} \)

\( On\ entry: \) the dimension of the array \( \text{comm} \).

If \( \text{lcomm} = -1 \), a workspace query is assumed and the function only calculates the dimensions of \( \text{icomm} \) and \( \text{comm} \) required by \( \text{nag_complex_sparse_eigensystem_iter} \) (f12apc), which it returns in \( \text{icomm}[0] \) and \( \text{comm}[0] \) respectively.

\( Constraint: \ \text{lcomm} \geq 3 \times n + 3 \times ncv \times ncv + 5 \times ncv + 60 \) or \( \text{lcomm} = -1. \)

8: \( \text{fail} \) – NagError \( * \) \( \text{Input/Output} \)

The NAG error argument (see Section 3.6 in the Essential Introduction).
6 Error Indicators and Warnings

NE_ALLOC_FAIL
Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM
On entry, argument \(\text{value}\) had an illegal value.

NE_INT
On entry, \(n = \langle \text{value} \rangle\).
Constraint: \(n > 0\).

On entry, \(nev = \langle \text{value} \rangle\).
Constraint: \(nev > 0\).

NE_INT_2
The length of the integer array \(icomm\) is too small \(licomm = \langle \text{value} \rangle\), but must be at least \(\langle \text{value} \rangle\).

NE_INT_3
On entry, \(icomm = \langle \text{value} \rangle\), \(n = \langle \text{value} \rangle\) and \(ncv = \langle \text{value} \rangle\).
Constraint: \(icomm \geq 3 \times n + 3 \times ncv \times ncv + 5 \times ncv + 60\).

On entry, \(ncv = \langle \text{value} \rangle\), \(nev = \langle \text{value} \rangle\) and \(n = \langle \text{value} \rangle\).
Constraint: \(ncv \geq nev + 1\) and \(ncv \leq n\).

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_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.

7 Accuracy
Not applicable.

8 Parallelism and Performance
Not applicable.

9 Further Comments
None.

10 Example
This example solves \(Ax = \lambda x\) in regular 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 eigenvalues of largest magnitude are found.
10.1 Program Text

/* nag_complex_sparse_eigensystem_init (f12anc) Example Program. */
* Copyright 2014 Numerical Algorithms Group.
* Mark 8, 2005.
*/
#include <nag.h>
#include <nag_stdlib.h>
#include <nag_string.h>
#include <stdio.h>
#include <naga02.h>
#include <nagf12.h>
#include <nagf16.h>

static void av(Integer, Complex *, Complex *);
static void tv(Integer, Complex *, Complex *);

int main(void)
{
    /* Constants */
    Integer imon = 0;
    /* Scalars */
    Complex sigma;
    double estnrm;
    Integer exit_status, i, irevcm, lcomm, licomm, n, nconv, ncv;
    Integer nev, niter, nshift, nx;
    /* Nag types */
    NagError fail;

    /* Arrays */
    Complex *comm = 0, *eigest = 0, *eigv = 0, *resid = 0, *v = 0;
    Integer *icomm = 0;
    /* Poneters */
    Complex *mx = 0, *x = 0, *y = 0;

    /* Assign to Complex type using nag_complex (a02bac) */
    sigma = nag_complex(0.0, 0.0);
    exit_status = 0;
    INIT_FAIL(fail);
    printf("nag_complex_sparse_eigensystem_init (f12anc) Example "
           "Program Results\n");
    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%*[^\n] ");
    #else
    scanf("%[^\n] ");
    #endif
    #ifdef _WIN32
    scanf_s("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%[^\n] ", &nx, &nev, &ncv);
    #else
    scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%[^\n] ", &nx, &nev, &ncv);
    #endif
    n = nx * nx;
    /* Allocate memory */
    if (((eigv = NAG_ALLOC(ncv, Complex)) ||
         (eigest = NAG_ALLOC(ncv, Complex)) ||
         (resid = NAG_ALLOC(n, Complex)) ||
         (v = NAG_ALLOC(n * ncv, Complex)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    /* Initialise communication arrays for problem using
     nag_complex_sparse_eigensystem_init (f12anc).
     */
The first call sets lcomm = licomm = -1 to perform a workspace query. */
lcomm = licomm = -1;
if (!(comm = NAG_ALLOC(1, Complex)) ||
   !(icomm = NAG_ALLOC(1, Integer)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
nag_complex_sparse_eigensystem_init(n, nev, ncv, icomm, licomm,
   comm, lcomm, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_complex_sparse_eigensystem_init (f12anc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
lcomm = (Integer)comm[0].re;
licomm = icomm[0];
NAG_FREE(comm);
NAG_FREE(icomm);
if (!(comm = NAG_ALLOC(lcomm, Complex)) ||
   !(icomm = NAG_ALLOC(licomm, Integer)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
nag_complex_sparse_eigensystem_init(n, nev, ncv, icomm, licomm,
   comm, lcomm, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_complex_sparse_eigensystem_init (f12anc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
irevcm = 0;
REVCOMLOOP:
    /* repeated calls to reverse communication routine
    nag_complex_sparse_eigensystem_iter (f12apc). */
nag_complex_sparse_eigensystem_iter(&irevcm, resid, v, &x, &y, &mx,
   &nshift, comm, icomm, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_complex_sparse_eigensystem_iter (f12apc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
if (irevcm != 5 && irevcm != 0)
{
    if (irevcm == -1 || irevcm == 1)
    {
        /* Perform matrix vector multiplication y <- Op*x */
av(nx, x, y);
    }
    else if (irevcm == 4 && imon == 1)
    {
        /* If imon=1, get monitoring information using
        nag_complex_sparse_eigensystem_monit (f12asc). */
nag_complex_sparse_eigensystem_monit(&niter, &nconv, eigv,
   eigest, icomm, comm);
        /* Compute 2-norm of Ritz estimates using
        nag_zge_norm (f16uac). */
nag_zge_norm(Nag_ColMajor, Nag_FrobeniusNorm, nev, 1, eigest,
   eigv, &estnrm, &fail);
    }
}
printf("Error from nag_complex_sparse_eigensystem_monit"  
"(f12asc).\n\n", fail.message);  
exit_status = 1;  
goto END;  
}  
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_complex_sparse_eigensystem_sol  
(f12aqc) to compute eigenvalues/vectors. */  
nag_complex_sparse_eigensystem_sol(&nconv, eigv, v, sigma,  
resid, v, comm, icomm, &fail);  
if (fail.code != NE_NOERROR)  
{  
printf("Error from nag_complex_sparse_eigensystem_sol "  
"(f12aqc).\n\n", fail.message);  
exit_status = 1;  
goto END;  
}  
printf("\n The \%"NAG_IFMT" Ritz values", nconv);  
printf(" of largest magnitude are:\n\n");  
for (i = 0; i <= nconv-1; ++i)  
{  
printf("\%8"NAG_IFMT"(%12.4f, %12.4f)\n", i+1,  
eigv[i].re, eigv[i].im);  
}  
else  
{  
printf("Error from nag_complex_sparse_eigensystem_iter "  
"(f12apc).\n\n", fail.message);  
exit_status = 1;  
goto END;  
}  
END:  
NAG_FREE(comm);  
NAG_FREE(eigv);  
NAG_FREE(eigest);  
NAG_FREE(resid);  
NAG_FREE(v);  
NAG_FREE(icomm);  
return exit_status;  
}  
static void av(Integer nx, Complex *x, Complex *y)  
{  
/* Scalars */  
double hr;  
Integer i, j, lo;  
/* Function Body */  
/* Allocate memory */  
hr = (double) -(nx + 1) * (nx + 1);  
tv(nx, x, y);  
for (j = 0; j <= nx - 1; ++j)  
{  
y[j].re = y[j].re + hr*x[nx+j].re;  
y[j].im = y[j].im + hr*x[nx+j].im;  
}  
for (j = 2; j <= nx - 1; ++j)  
{  
lo = (j - 1) * nx;  
tv(nx, &x[lo], &y[lo]);  
for (i = 0; i <= nx - 1; ++i)  
{  
}
\begin{verbatim}
static void tv(Integer nx, Complex *x, Complex *y)
{
    /* Compute the matrix vector multiplication y<---T*x where T is a */
    /* nx by nx tridiagonal matrix. */

    /* Scalars */
    Complex dd, dl, du, h2, h, rho, z1, z2, z3;
    Integer j;

    /* Function Body */
    /* Assign to Complex type using nag_complex (a02bac) */
    h = nag_complex((double)(nx + 1), 0.);
    /* Compute Complex multiply using nag_complex_multiply (a02ccc). */
    h2 = nag_complex_multiply(h, h);
    dd = nag_complex_multiply(nag_complex(4.0, 0.0), h2);
    z1 = nag_complex_multiply(nag_complex(-1.0, 0.0), h2);
    /* Assign to Complex type using nag_complex (a02bac) */
    rho = nag_complex(1.0e2, 0.0);
    z2 = nag_complex_multiply(rho, h);
    z3 = nag_complex_multiply(nag_complex(5.0e-1, 0.0), z2);
    /* Compute Complex subtraction using nag_complex_subtract (a02cbc). */
    dl = nag_complex_subtract(z1, z3);
    /* Compute Complex addition using nag_complex_add (a02cac). */
    du = nag_complex_add(z1, z3);
    /* Compute Complex multiply using nag_complex_multiply (a02ccc). */
    z1 = nag_complex_multiply(dd, x[0]);
    z2 = nag_complex_multiply(du, x[1]);
    /* Compute Complex addition using nag_complex_add (a02cac). */
    y[0] = nag_complex_add(z1, z2);
    for (j = 1; j <= nx - 2; ++j)
    {
        /* Compute Complex multiply using nag_complex_multiply (a02ccc). */
        z1 = nag_complex_multiply(dl, x[j-1]);
        z2 = nag_complex_multiply(dd, x[j]);
        /* Compute Complex addition using nag_complex_add (a02cac). */
        y[j] = nag_complex_add(z1, z2);
        y[j] = nag_complex_add(y[j], z3);
    }
    /* Compute Complex multiply using nag_complex_multiply (a02ccc). */
    z1 = nag_complex_multiply(dl, x[nx-2]);
    z2 = nag_complex_multiply(dd, x[nx-1]);
    /* Compute Complex addition using nag_complex_add (a02cac). */
    y[nx-1] = nag_complex_add(z1, z2);
    return;
} /* tv */
\end{verbatim}

10.2 Program Data

nag_complex_sparse_eigensystem_init (f12anc) Example Program Data

10 4 20 : Vaues for nx, nev and ncv
10.3 Program Results

nag_complex_sparse_eigensystem_init (f12anc) Example Program Results

The 4 Ritz values of largest magnitude are:

1  (  716.1973,  -1029.5838)  
2  (  687.5834,  -1029.5838)  
3  (  716.1973,   1029.5838)  
4  (  687.5834,   1029.5838)