

## NAG Library Function Document

### nag\_real\_sparse\_eigensystem\_init (f12aac)

#### 1 Purpose

nag\_real\_sparse\_eigensystem\_init (f12aac) is a setup function in a suite of functions consisting of nag\_real\_sparse\_eigensystem\_init (f12aac), nag\_real\_sparse\_eigensystem\_iter (f12abc), nag\_real\_sparse\_eigensystem\_sol (f12acc), nag\_real\_sparse\_eigensystem\_option (f12adc) and nag\_real\_sparse\_eigensystem\_monit (f12aec). It is used to find some of the eigenvalues (and optionally the corresponding eigenvectors) of a standard or generalized eigenvalue problem defined by real nonsymmetric matrices.

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

#### 2 Specification

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

void nag_real_sparse_eigensystem_init (Integer n, Integer nev, Integer ncv,
    Integer icomm[], Integer licomm, double 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 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\_init (f12aac) is a setup function which must be called before nag\_real\_sparse\_eigensystem\_iter (f12abc), the reverse communication iterative solver, and before nag\_real\_sparse\_eigensystem\_option (f12adc), the options setting function. nag\_real\_sparse\_eigensystem\_sol (f12acc) is a post-processing function that must be called following a successful final exit from nag\_real\_sparse\_eigensystem\_iter (f12abc), while nag\_real\_sparse\_eigensystem\_monit (f12aec) 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\_real\_sparse\_eigensystem\_option (f12adc), 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\_real\_sparse\_eigensystem\_option (f12adc).

#### 4 References

Lehoucq R B (2001) Implicitly restarted Arnoldi methods and subspace iteration *SIAM Journal on Matrix Analysis and Applications* **23** 551–562

Lehoucq R B and Scott J A (1996) An evaluation of software for computing eigenvalues of sparse nonsymmetric matrices *Preprint MCS-P547-1195* Argonne National Laboratory

Lehoucq R B and Sorensen D C (1996) Deflation techniques for an implicitly restarted Arnoldi iteration *SIAM Journal on Matrix Analysis and Applications* **17** 789–821

Lehoucq R B, Sorensen D C and Yang C (1998) *ARPACK Users' Guide: Solution of Large-scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods* SIAM, Philadelphia

## 5 Arguments

- 1: **n** – Integer *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 *Input*  
*On entry:* the number of eigenvalues to be computed.  
*Constraint:*  $0 < nev < n - 1$ .
- 3: **ncv** – Integer *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: **icomm**[**max(1, licomm)**] – Integer *Communication Array*  
*On exit:* contains data to be communicated to the other functions in the suite.
- 5: **licomm** – Integer *Input*  
*On entry:* the dimension of the array **icomm**.  
 If **licomm** = -1, a workspace query is assumed and the function only calculates the required dimensions of **icomm** and **comm**, which it returns in **icomm**[0] and **comm**[0] respectively.  
*Constraint:*  $licomm \geq 140$  or **licomm** = -1.
- 6: **comm**[**max(1, licomm)**] – double *Communication Array*  
*On exit:* contains data to be communicated to the other functions in the suite.
- 7: **lcomm** – Integer *Input*  
*On entry:* the dimension of the array **comm**.  
 If **lcomm** = -1, a workspace query is assumed and the function only calculates the dimensions of **icomm** and **comm** required by `nag_real_sparse_eigensystem_iter` (f12abc), which it returns in **icomm**[0] and **comm**[0] respectively.  
*Constraint:*  $lcomm \geq 3 \times n + 3 \times nev \times ncv + 6 \times ncv + 60$  or **lcomm** = -1.
- 8: **fail** – NagError \* *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  $\langle value \rangle$  had an illegal value.

**NE\_INT**

On entry,  $\mathbf{n} = \langle value \rangle$ .

Constraint:  $\mathbf{n} > 0$ .

On entry,  $\mathbf{nev} = \langle value \rangle$ .

Constraint:  $\mathbf{nev} > 0$ .

**NE\_INT\_2**

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

**NE\_INT\_3**

On entry,  $\mathbf{licomm} = \langle value \rangle$ ,  $\mathbf{n} = \langle value \rangle$  and  $\mathbf{nev} = \langle value \rangle$ .

Constraint:  $\mathbf{licomm} \geq 3 \times \mathbf{n} + 3 \times \mathbf{nev} \times \mathbf{nev} + 6 \times \mathbf{nev} + 60$ .

On entry,  $\mathbf{nev} = \langle value \rangle$ ,  $\mathbf{nev} = \langle value \rangle$  and  $\mathbf{n} = \langle value \rangle$ .

Constraint:  $\mathbf{nev} > \mathbf{nev} + 1$  and  $\mathbf{nev} \leq \mathbf{n}$ .

**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, where  $\rho = 100$ .

## 10.1 Program Text

```

/* nag_real_sparse_eigensystem_init (f12aac) 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 <nagf12.h>
#include <nagf16.h>

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

int main(void)
{
    /* Constants */
    Integer imon = 0;
    /* Scalars */
    double sigmai = 0, sigmar = 0, estnrm;
    Integer exit_status, irevcm, j, lcomm, licomm, n, nconv, ncv, nev;
    Integer niter, nshift, nx;
    /* Arrays */
    double *comm = 0, *eigvr = 0, *eigvi = 0, *eigest = 0;
    double *resid = 0, *v = 0;
    Integer *icomm = 0;
    /* Pointers */
    double *mx = 0, *x = 0, *y = 0;
    /* Nag types */
    NagError fail;

    exit_status = 0;
    INIT_FAIL(fail);

    printf("nag_real_sparse_eigensystem_init (f12aac) Example Program "
           "Results\n");
    /* Skip heading in data file. */
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

    /* Read values for nx, nev and ncv from data file. */
#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

    /* Allocate memory */
    n = nx * nx;
    if (!(eigvr = NAG_ALLOC(ncv, double)) ||
        !(eigvi = NAG_ALLOC(ncv, double)) ||
        !(eigest = NAG_ALLOC(ncv, double)) ||
        !(resid = NAG_ALLOC(n, double)) ||
        !(v = NAG_ALLOC(n * ncv, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

    /* Initialise communication arrays for problem using
    nag_real_sparse_eigensystem_init (f12aac).
    The first call sets lcomm = licomm = -1 to perform a workspace

```

```

    query. */
lcomm = licomm = -1;
if (!(comm = NAG_ALLOC(1, double)) ||
    !(icomm = NAG_ALLOC(1, Integer)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
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;
}
lcomm = (Integer)comm[0];
licomm = icomm[0];
NAG_FREE(comm);
NAG_FREE(icomm);
if (!(comm = NAG_ALLOC(lcomm, double)) ||
    !(icomm = NAG_ALLOC(licomm, Integer)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
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 spectrum using
   nag_real_sparse_eigensystem_option (f12adc). */
nag_real_sparse_eigensystem_option("SMALLEST MAG", icomm,
                                   comm, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_real_sparse_eigensystem_option (f12adc).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}
irevcm = 0;
REVCOMLOOP:
/* 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 matrix vector multiplication y <--- Op*x */
        av(nx, 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,
                    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
       (f12aac) to compute eigenvalues/vectors. */
    nag_real_sparse_eigensystem_sol(&nconv, eigvr, eigvi, v, sigmar,
                                    sigmai, resid, v, comm, icomm,
                                    &fail);

    printf("\n\n The %4"NAG_IFMT" Ritz values", nconv);
    printf(" of smallest magnitude 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%s\n",
           fail.message);
    exit_status = 1;
    goto END;
}
END:
NAG_FREE(comm);
NAG_FREE(eigvr);
NAG_FREE(eigvi);
NAG_FREE(eigest);
NAG_FREE(resid);
NAG_FREE(v);
NAG_FREE(icomm);
return exit_status;
}

static void av(Integer nx, double *v, double *w)
{
    /* Constants*/
    const double beta = 1.0;
    /* Scalars */
    double      nx2;
    Integer      j, lo;
    /* Nag types */
    NagError     fail;

    /* Function Body */
    INIT_FAIL(fail);
    nx2 = -((double)((nx + 1) * (nx + 1)));
    tv(nx, v, w);
    nag_daxpby(nx, nx2, &v[nx], 1, beta, w, 1, &fail);
    for (j = 2; j <= nx - 1; ++j)
    {
        lo = (j - 1) * nx;
        tv(nx, &v[lo], &w[lo]);
        nag_daxpby(nx, nx2, &v[lo-nx], 1, beta, &w[lo], 1, &fail);
        nag_daxpby(nx, nx2, &v[lo+nx], 1, beta, &w[lo], 1, &fail);
    }
    lo = (nx - 1) * nx;
    tv(nx, &v[lo], &w[lo]);
    nag_daxpby(nx, nx2, &v[lo-nx], 1, beta, &w[lo], 1, &fail);
    return;
} /* av */

static void tv(Integer nx, double *x, double *y)
{

```

```

/* Compute the matrix vector multiplication y<---T*x where T is a nx */
/* by nx tridiagonal matrix with constant diagonals (dd, dl and du). */
/* Scalars */
double dd, dl, du, nx1, nx2;
Integer j;
/* Function Body */
nx1 = (double)(nx + 1);
nx2 = nx1 * nx1;
dd = nx2 * 4.;
dl = -nx2 - nx1 * 50.;
du = -nx2 + nx1 * 50.;
y[0] = dd * x[0] + du * x[1];
for (j = 1; j <= nx - 2; ++j)
{
    y[j] = dl * x[j-1] + dd * x[j] + du * x[j+1];
}
y[nx-1] = dl * x[nx-2] + dd * x[nx-1];
return;
} /* tv */

```

## 10.2 Program Data

nag\_real\_sparse\_eigensystem\_init (f12aac) Example Program Data  
 10 10 30 : Values for nx, nev and ncv

## 10.3 Program Results

nag\_real\_sparse\_eigensystem\_init (f12aac) Example Program Results

The 10 Ritz values of smallest magnitude are:

1	(	251.8027	,	152.7109	)
2	(	251.8027	,	-152.7109	)
3	(	280.4166	,	152.7109	)
4	(	280.4166	,	-152.7109	)
5	(	325.5237	,	152.7109	)
6	(	325.5237	,	-152.7109	)
7	(	383.4696	,	152.7109	)
8	(	383.4696	,	-152.7109	)
9	(	449.5598	,	152.7109	)
10	(	449.5598	,	-152.7109	)

---