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

```c
#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


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

5: \( \text{licomm} \) – Integer \( Input \)
   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: \( \text{comm}[\max(1, \text{licomm})] \) – double \( Communication \ Array \)
   On exit: contains data to be communicated to the other functions in the suite.

7: \( \text{lcomm} \) – Integer \( Input \)
   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 \( \text{nag_real_sparse_eigensystem_iter} \) (\( f12abc \)), which it returns in \( \text{icomm}[0] \) and \( \text{comm}[0] \) respectively.
   Constraint: \( \text{lcomm} \geq 3 \times n + 3 \times nev \times nev + 6 \times nev + 60 \) or \( \text{lcomm} = -1 \).

8: \( \text{fail} \) – NagError * \( Input/Output \)
   The NAGError * 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, \( n = \langle value \rangle \).
Constraint: \( n > 0 \).
On entry, \( nev = \langle value \rangle \).
Constraint: \( nev > 0 \).

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

NE_INT_3
On entry, \( lcomm = \langle value \rangle \), \( n = \langle value \rangle \) and \( ncv = \langle value \rangle \).
Constraint: \( lcomm \geq 3 \times n + 3 \times ncv \times ncv + 6 \times ncv + 60 \).
On entry, \( ncv = \langle value \rangle \), \( nev = \langle value \rangle \) and \( n = \langle value \rangle \).
Constraint: \( ncv > nev + 1 \) 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, 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 cnv 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
```c
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;
/* 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). */
    }
    else
    {
        printf("Error from nag_real_sparse_eigensystem_iter (f12abc).\n%s\n",
                fail.message);
        exit_status = 1;
        goto END;
    }
}
```
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
", estnrm);
}
go to 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);
    printf("\n\n The %4"NAG_IFMT" Ritz values", nconv);
    for (j = 0; j <= nconv-1; ++j)
    {          printf("%8"NAG_IFMT"%s( %12.4f , %12.4f )\n", j+1, ",
    eigvr[j], eigvi[j]);
    }
} else
{
    printf("Error from nag_real_sparse_eigensystem_iter (f12abc).\n
    exit_status = 1;
go to 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:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>251.8027</td>
<td>152.7109</td>
</tr>
<tr>
<td>2</td>
<td>251.8027</td>
<td>-152.7109</td>
</tr>
<tr>
<td>3</td>
<td>280.4166</td>
<td>152.7109</td>
</tr>
<tr>
<td>4</td>
<td>280.4166</td>
<td>-152.7109</td>
</tr>
<tr>
<td>5</td>
<td>325.5237</td>
<td>152.7109</td>
</tr>
<tr>
<td>6</td>
<td>325.5237</td>
<td>-152.7109</td>
</tr>
<tr>
<td>7</td>
<td>383.4696</td>
<td>152.7109</td>
</tr>
<tr>
<td>8</td>
<td>383.4696</td>
<td>-152.7109</td>
</tr>
<tr>
<td>9</td>
<td>449.5598</td>
<td>152.7109</td>
</tr>
<tr>
<td>10</td>
<td>449.5598</td>
<td>-152.7109</td>
</tr>
</tbody>
</table>