NAG Library Function Document

nag_real_symm_sparse_eigensystem_sol (f12fcc)

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_symm_sparse_eigensystem_option (f12fdc) need not be called. If, however, you wish to reset some or all of the settings please refer to Section 11 in nag_real_symm_sparse_eigensystem_option (f12fdc) for a detailed description of the specification of the optional arguments.

1 Purpose

nag_real_symm_sparse_eigensystem_sol (f12fcc) is a post-processing function in a suite of functions which includes nag_real_symm_sparse_eigensystem_init (f12fac), nag_real_symm_sparse_eigensystem_iter (f12fbc), nag_real_symm_sparse_eigensystem_option (f12fdc) and nag_real_symm_sparse_eigensystem_monit (f12fec). nag_real_symm_sparse_eigensystem_sol (f12fcc) must be called following a final exit from nag_real_symm_sparse_eigensystem_iter (f12fbc).

2 Specification

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

void nag_real_symm_sparse_eigensystem_sol (Integer *nconv, double d[],
   double z[], double sigma, const double resid[], double v[],
   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 symmetric. The suite can also be used to find selected eigenvalues/eigenvectors of smaller scale dense, real and symmetric problems.

Following a call to nag_real_symm_sparse_eigensystem_iter (f12fbc), nag_real_symm_sparse_eigensystem_sol (f12fcc) returns the converged approximations to eigenvalues and (optionally) the corresponding approximate eigenvectors and/or an orthonormal basis for the associated approximate invariant subspace. The eigenvalues (and eigenvectors) are selected from those of a standard or generalized eigenvalue problem defined by real symmetric matrices. There is negligible additional cost to obtain eigenvectors; an orthonormal basis is always computed, but there is an additional storage cost if both are requested.

nag_real_symm_sparse_eigensystem_sol (f12fcc) is based on the function dseupd from the ARPACK package, which uses the Implicitly Restarted Lanczos 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 symmetric matrices is provided in Lehoucq and Scott (1996). This suite of functions offers the same functionality as the ARPACK software for real symmetric problems, but the interface design is quite different in order to make the option setting clearer and to simplify some of the interfaces.

nag_real_symm_sparse_eigensystem_sol (f12fcc), is a post-processing function that must be called following a successful final exit from nag_real_symm_sparse_eigensystem_iter (f12fbc). nag_real_symm_sparse_eigensystem_sol (f12fcc) uses data returned from nag_real_symm_sparse_eigensystem_iter (f12fbc) and options, set either by default or explicitly by calling nag_real_symm_sparse_eigensystem_option (f12fdc), to return the converged approximations to selected eigenvalues and (optionally):
the corresponding approximate eigenvectors;
an orthonormal basis for the associated approximate invariant subspace;
both.

4 References

5 Arguments
1: nconv – Integer *  
   Output  
   On exit: the number of converged eigenvalues as found by nag_real_symm_sparse_eigensystem_iter (f12fbc).
2: d[dim] – double  
   Output  
   Note: the dimension, dim, of the array d must be at least ncv (see nag_real_symm_sparse_eigensystem_init (f12fac)).
   On exit: the first nconv locations of the array d contain the converged approximate eigenvalues.
3: z[n × (nev + 1)] – double  
   Output  
   On exit: if the default option Vectors = RITZ (see nag_real_symm_sparse_eigensystem_option (f12fdc)) has been selected then z contains the final set of eigenvectors corresponding to the eigenvalues held in d. The real eigenvector associated with an eigenvalue is stored in the corresponding array section of z.
4: sigma – double  
   Input  
   On entry: if one of the Shifted Inverse (see nag_real_symm_sparse_eigensystem_option (f12fdc)) modes has been selected then sigma contains the real shift used; otherwise sigma is not referenced.
5: resid[dim] – const double  
   Input  
   Note: the dimension, dim, of the array resid must be at least n (see nag_real_symm_sparse_eigensystem_init (f12fac)).
   On entry: must not be modified following a call to nag_real_symm_sparse_eigensystem_iter (f12fbc) since it contains data required by nag_real_symm_sparse_eigensystem_sol (f12fcc).
6: v[n × ncv] – double  
   Input/Output  
   The ith element of the jth basis vector is stored in location v[n × (j – 1) + i – 1], for i = 1, 2, ..., n and j = 1, 2, ..., ncv.
   On entry: the ncv sections of v, of length n, contain the Lanczos basis vectors for OP as constructed by nag_real_symm_sparse_eigensystem_iter (f12fbc).
On exit: if the option Vectors = SCHUR has been set, or the option Vectors = RITZ has been set and a separate array $z$ has been passed (i.e., $z$ does not equal $v$), then the first $nconv$ sections of $v$, of length $n$, will contain approximate Schur vectors that span the desired invariant subspace.

7: $\text{comm}[\text{dim}]$ – double  

*Communication Array*

Note: the dimension, $\text{dim}$, of the array $\text{comm}$ must be at least $\max(1, \text{lcomm})$ (see nag_real_symm_sparse_eigensystem_init (f12fac)).

On initial entry: must remain unchanged from the prior call to nag_real_symm_sparse_eigensystem_init (f12fac).

On exit: contains data on the current state of the solution.

8: $\text{icomm}[\text{dim}]$ – Integer  

*Communication Array*

Note: the dimension, $\text{dim}$, of the array $\text{icomm}$ must be at least $\max(1, \text{licomm})$ (see nag_real_symm_sparse_eigensystem_init (f12fac)).

On initial entry: must remain unchanged from the prior call to nag_real_symm_sparse_eigensystem_init (f12fac).

On exit: contains data on the current state of the solution.

9: $\text{fail}$ – 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 $\langle\text{value}\rangle$ had an illegal value.

**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_INVALID_OPTION**

On entry, Vectors = SELECT, but this is not yet implemented.

**NE_MAX_ITER**

During calculation of a tridiagonal form, there was a failure to compute $\langle\text{value}\rangle$ eigenvalues in a total of $\langle\text{value}\rangle$ iterations.

**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_RITZ_COUNT

Got a different count of the number of converged Ritz values than the value passed to it through the argument icomm: number counted = (value), number expected = (value). This usually indicates that a communication array has been altered or has become corrupted between calls to nag_real_symm_sparse_eigensystem_iter (f12fbc) and nag_real_symm_sparse_eigensystem_sol (f12fcc).

NE_ZERO_EIGS_FOUND

The number of eigenvalues found to sufficient accuracy, as communicated through the argument icomm, is zero. You should experiment with different values of nev and ncv, or select a different computational mode or increase the maximum number of iterations prior to calling nag_real_symm_sparse_eigensystem_iter (f12fbc).

7 Accuracy

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

8 Parallelism and Performance

nag_real_symm_sparse_eigensystem_sol (f12fcc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_real_symm_sparse_eigensystem_sol (f12fcc) 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 $Ax = \lambda Bx$ in regular mode, where $A$ and $B$ are obtained from the standard central difference discretization of the one-dimensional Laplacian operator $\frac{d^2u}{dx^2}$ on $[0, 1]$, with zero Dirichlet boundary conditions.

10.1 Program Text

/* nag_real_symm_sparse_eigensystem_sol (f12fcc) 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 av(Integer, double *);
static void mv(Integer, double *, double *);
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 estnrm, h, r1, r2, sigma = 0.0;
  Integer exit_status, info, irevcm, j, lcomm, n, nconv, ncv;
  Integer nev, niter, nshift;
  /* Nag types */
  NagError fail;

  /* Arrays */
  double *dd = 0, *dl = 0, *du = 0, *du2 = 0, *comm = 0, *eigest = 0;
  double *eigv = 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_symm_sparse_eigensystem_sol (f12fcc) 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]> ", &n, &nev, &ncv);
  #else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[\n]> ", &n, &nev, &ncv);
  #endif
  /* Allocate memory */
  lcomm = 3*n + ncv*ncv + 8*ncv + 60;
  if (!dd = NAG_ALLOC(n, double)) ||
      !ldl = NAG_ALLOC(n, double)) ||
      !du = NAG_ALLOC(n, double)) ||
      !du2 = NAG_ALLOC(n, double)) ||
      !comm = NAG_ALLOC(lcomm, double)) ||
      !eigv = NAG_ALLOC(ncv, double)) ||
      !eigest = NAG_ALLOC(ncv, double)) ||
      !resid = NAG_ALLOC(n, double)) ||
      !v = NAG_ALLOC(n * ncv, double)) ||
      !icomm = NAG_ALLOC(lcomm, Integer)) ||
      !ipiv = NAG_ALLOC(n, Integer))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
    /* Initialise communication arrays for problem using
     * nag_real_symm_sparse_eigensystem_init (f12fac). */
    nag_real_symm_sparse_eigensystem_init(n, nev, ncv, icomm, licomm, comm, lcomm, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_real_symm_sparse_eigensystem_init "
             "(f12fac).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
    /* Select the problem type using
     * nag_real_symm_sparse_eigensystem_option (f12fdc). */
nag_real_symm_sparse_eigensystem_option("generalized", icomm, comm, &fail);
if (fail.code != NE_NOERROR)
{
    printf(" Error from nag_real_symm_sparse_eigensystem_option "
"(f12fdc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Setup M and factorise */
static h = 1.0 / (double)(n + 1);
r1 = 2.0 * h / 3.0;
r2 = h / 6.0;
for (j = 0; j <= n-1; ++j)
{
    dd[j] = r1;
    dl[j] = r2;
    du[j] = r2;
}
my_dgttrf(n, dl, dd, du, du2, ipiv, &info);
irevcm = 0;

REVCOMLOOP:
/* Repeated calls to reverse communication routine
    nag_real_symm_sparse_eigensystem_iter (f12fbc). */
    nag_real_symm_sparse_eigensystem_iter(&irevcm, resid, v, &x, &y, &mx,
    &nshift, comm, icomm, &fail);
if (irevcm != 5)
{
    if (irevcm == -1 || irevcm == 1)
    {
        /* Perform the following operations in order:
            x <--- A*x;
            y <--- inv[M]*x. */
        av(n, x);
        my_dgttrs(n, dl, dd, du, du2, ipiv, x, y);
    }
    else if (irevcm == 2)
    {
        /* Perform y <--- M*x. */
        mv(n, x, y);
    }
    else if (irevcm == 4 && imon == 1)
    {
        /* If imon=1, get monitoring information using
            nag_real_symm_sparse_eigensystem_monit (f12fec). */
            nag_real_symm_sparse_eigensystem_monit(&niter, &nconv, eigv, eigest,
            icomm, comm);
        /* Compute 2-norm of Ritz estimates using
            nag_dge_norm (f16rac).*/
            nag_dge_norm(Nag_ColMajor, Nag_FrobeniusNorm, nev, 1, eigest,
            &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_symm_sparse_eigensystem_sol (f12fcc)
        to compute eigenvalues/vectors. */
    nag_real_symm_sparse_eigensystem_sol(&nconv, eigv, sigma, resid, v,
    comm, icomm, &fail);
    printf("\n The %4"NAG_IFMT" generalized Ritz values", nconv);
    printf(" of largest magnitude are:\n\n%9.1f
", eigv[j]);
    }
}
else
{
    printf(" Error from nag_real_symm_sparse_eigensystem_iter "
    "(f12fbc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
END:
NAG_FREE(dd);
NAG_FREE(dl);
NAG_FREE(du);
NAG_FREE(du2);
NAG_FREE(comm);
NAG_FREE(eigv);
NAG_FREE(eigest);
NAG_FREE(resid);
NAG_FREE(v);
NAG_FREE(icomm);
NAG_FREE(ipiv);

return exit_status;
}

static void mv(Integer n, double *v, double *y)
{
    /* Scalars */
    double h;
    Integer j;

    /* Function Body */
    h = 1.0 / ((double)(n + 1) * 6.0);
    y[0] = h * (v[0] * 4.0 + v[1]);
    for (j = 1; j <= n - 2; ++j)
    {
        y[j] = h * (v[j-1] + v[j] * 4.0 + v[j+1]);
    }
    y[n-1] = h * (v[n-2] + v[n-1] * 4.0);
} /*mv*/

static void av(Integer n, double *v)
{
    /* Scalars */
    double h, vj, vjm1;
    Integer j;

    /* Function Body */
    h = (double)(n + 1);
    vjm1 = v[0];
    v[0] = h * (vjm1 * 2.0 - v[1]);
    for (j = 1; j <= n - 2; ++j)
    {
        vj = v[j];
        v[j] = h*(-vjm1 + vj * 2.0 - v[j+1]);
        vjm1 = vj;
    }
    v[n-1] = h * (-vjm1 + v[n-1] * 2.0);
} /* av */

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[])

f12fcc.8  Mark 25
/* 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];
}
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_symm_sparse_eigensystem_sol (f12fcc) Example Program Data
100 4 10 : Values for n, nev and ncv

10.3 Program Results
nag_real_symm_sparse_eigensystem_sol (f12fcc) Example Program Results

The 4 generalized Ritz values of largest magnitude are:

1 121003.5
2 121616.6
3 122057.5
4 122323.2