nag_sparse_nsym_sol (f11dec)

1. Purpose

nag_sparse_nsym_sol (f11dec) solves a real sparse nonsymmetric system of linear equations, represented in coordinate storage format, using a restarted generalized minimal residual (RGMRES), conjugate gradient squared (CGS), or stabilized bi-conjugate gradient (Bi-CGSTAB) method, without preconditioning, with Jacobi, or with SSOR preconditioning.

2. Specification

```
#include <nag.h>
#include <nagf11.h>

void nag_sparse_nsym_sol(Nag_SparseNsym_Method method,
            Nag_SparseNsym_PrecType precon, Integer n, Integer nnz,
            double a[], Integer irow[], Integer icol[], double omega, double b[],
            Integer m, double tol, Integer maxitn, double x[], double *rnorm,
            Integer *itn, Nag_Sparse_Comm *comm, NagError *fail)
```

3. Description

This routine solves a real sparse nonsymmetric system of linear equations:

$$Ax = b,$$

using an RGMRES (see Saad and Schultz (1986)), CGS (see Sonneveld (1989)), or Bi-CGSTAB(ℓ) method (see van der Vorst (1989), Sleijpen and Fokkema (1993)).

The routine allows the following choices for the preconditioner:

no preconditioning;

Jacobi preconditioning (see Young (1971));

symmetric successive-over-relaxation (SSOR) preconditioning (see Young (1971)).

For incomplete LU (ILU) preconditioning see nag_sparse_nsym_fac_sol (f11dcc).

The matrix A is represented in coordinate storage (CS) format (see Section 2.1.1 of the Chapter Introduction) in the arrays **a**, **irow** and **icol**. The array **a** holds the non-zero entries in the matrix, while **irow** and **icol** hold the corresponding row and column indices.

4. Parameters

method

Input: specifies the iterative method to be used. The possible choices are:

if **method** = **Nag_SparseNsym_RGMRES** then the restarted generalised minimum residual method is used;

if **method** = **Nag_SparseNsym_CGS** then the conjugate gradient squared method is used;

if **method** = **Nag_SparseNsym_BiCGSTAB** then the bi-conjugate gradient stabilised (ℓ) method is used.

Constraint: **method** = **Nag_SparseNsym_RGMRES**, **Nag_SparseNsym_CGS** or **Nag_SparseNsym_BiCGSTAB**.

precon

Input: specifies the type of preconditioning to be used. The possible choices are:

if **precon** = **Nag_SparseNsym_NoPrec**, no preconditioning;

if **precon** = **Nag_SparseNsym_SSORPrec**, symmetric successive-over-relaxation;

if **precon** = **Nag_SparseNsym_JacPrec**, Jacobi.

Constraint: **precon** = **Nag_SparseNsym_NoPrec**, **Nag_SparseNsym_SSORPrec** or **Nag_SparseNsym_JacPrec**.

n

Input: the order of the matrix A.

Constraint: $\mathbf{n} \geq 1$.

nnz

Input: the number of non-zero elements in the matrix A.

Constraint: $1 \leq \mathbf{nnz} \leq \mathbf{n}^2$.

a[nnz]

Input: the non-zero elements of the matrix A, ordered by increasing row index, and by increasing column index within each row. Multiple entries for the same row and column indices are not permitted. The routine nag_sparse_nsym_sort (f11zac) may be used to order the elements in this way.

irow[nnz]

icol[nnz]

Input: the row and column indices of the non-zero elements supplied in **a**.

Constraint: **irow** and **icol** must satisfy the following constraints (which may be imposed by a call to nag_sparse_nsym_sort (f11zac)):

$1 \leq \mathbf{irow}[i] \leq \mathbf{n}$ and $1 \leq \mathbf{icol}[i] \leq \mathbf{n}$, for $i = 0, 1, \ldots, \mathbf{nnz} - 1$.
$\mathbf{irow}[i-1] < \mathbf{irow}[i]$, or
$\mathbf{irow}[i-1] = \mathbf{irow}[i]$ and $\mathbf{icol}[i-1] < \mathbf{icol}[i]$, for $i = 1, 2, \ldots,$**nnz**-1.

omega

Input: if **precon** = **Nag_SparseNsym_SSORPrec**, **omega** is the relaxation parameter ω to be used in the SSOR method. Otherwise **omega** need not be initialised and is not referenced.

Constraint: $0.0 <$ **omega** < 2.0.

b[n]

Input: the right-hand side vector b.

m

Input: if **method** = **Nag_SparseNsym_RGMRES**, **m** is the dimension of the restart subspace; if **method** = **Nag_SparseNsym_BiCGSTAB**, **m** is the order ℓ of the polynomial Bi-CGSTAB method; otherwise **m** is not referenced.

Constraints:

If **method** = **Nag_SparseNsym_RGMRES**, $0 < \mathbf{m} \leq \min(\mathbf{n},50)$;
If **method** = **Nag_SparseNsym_BiCGSTAB**, $0 < \mathbf{m} \leq \min(\mathbf{n},10)$.

tol

Input: the required tolerance. Let x_k denote the approximate solution at iteration k, and r_k the corresponding residual. The algorithm is considered to have converged at iteration k if:

$$\|r_k\|_\infty \leq \tau \times (\|b\|_\infty + \|A\|_\infty \|x_k\|_\infty).$$

If **tol** ≤ 0.0, $\tau = \max(\sqrt{\epsilon}, \sqrt{\mathbf{n}}\,\epsilon)$ is used, where ϵ is the ***machine precision***. Otherwise $\tau = \max(\mathbf{tol}, 10\epsilon, \sqrt{\mathbf{n}}\,\epsilon)$ is used.

Constraint: **tol** < 1.0.

maxitn

Input: the maximum number of iterations allowed.

Constraint: **maxitn** ≥ 1.

x[n]

Input: an initial approximation to the solution vector x.

Output: an improved approximation to the solution vector x.

rnorm

Output: the final value of the residual norm $\|r_k\|_\infty$, where k is the output value of **itn**.

itn
Output: the number of iterations carried out.

comm
Input/Output: a pointer to a structure of type **Nag_Sparse_Comm** whose members are used by the iterative solver.

fail
The NAG error parameter, see the Essential Introduction to the NAG C Library.

5. Error Indications and Warnings

NE_BAD_PARAM
On entry, parameter **method** had an illegal value.
On entry, parameter **precon** had an illegal value.

NE_INT_ARG_LT
On entry, **n** must not be less than 1: **n** = ⟨*value*⟩.
On entry, **maxitn** must not be less than 1: **maxitn** = ⟨*value*⟩.

NE_INT_2
On entry, **nnz** = ⟨*value*⟩, **n** = ⟨*value*⟩.
Constraint: $1 \leq \mathbf{nnz} \leq \mathbf{n}^2$.

On entry, **m** = ⟨*value*⟩, min(**n**,50) = ⟨*value*⟩.
Constraint: $0 < \mathbf{m} \leq \min(\mathbf{n},50)$ when **method** = **Nag_SparseNsym_RGMRES**.

On entry, **m** = ⟨*value*⟩, min(**n**,10) = ⟨*value*⟩.
Constraint: $0 < \mathbf{m} \leq \min(\mathbf{n},10)$ when **method** = **Nag_SparseNsym_BiCGSTAB**.

NE_REAL
On entry, **omega** = ⟨*value*⟩.
Constraint: 0.0 < **omega** < 2.0 when **precon** = **Nag_SparseNsym_SSORPrec**.

NE_REAL_ARG_GE
On entry, **tol** must not be greater than or equal to 1: **tol** = ⟨*value*⟩.

NE_ZERO_DIAGONAL_ELEM
On entry, the matrix **a** has a zero diagonal element. Jacobi and SSOR preconditioners are not appropriate for this problem.

NE_ACC_LIMIT
The required accuracy could not be obtained. However, a reasonable accuracy has been obtained and further iterations cannot improve the result.

You should check the output value of **rnorm** for acceptability. This error code usually implies that your problem has been fully and satisfactorily solved to within or close to the accuracy available on your system. Further iterations are unlikely to improve on this situation.

NE_NOT_REQ_ACC
The required accuracy has not been obtained in **maxitn** iterations.

NE_NONSYMM_MATRIX_DUP
A non-zero matrix element has been supplied which does not lie within the matrix A, is out of order or has duplicate row and column indices, i.e., one or more of the following constraints has been violated:

$1 \leq \mathbf{irow}[i] \leq \mathbf{n}$ and $1 \leq \mathbf{icol}[i] \leq \mathbf{n}$, for $i = 0, 1, \ldots, \mathbf{nnz}-1$.
$\mathbf{irow}[i-1] < \mathbf{irow}[i]$, or
$\mathbf{irow}[i-1] = \mathbf{irow}[i]$ and $\mathbf{icol}[i-1] < \mathbf{icol}[i]$, for $i = 1, 2, \ldots, \mathbf{nnz}-1$.

Call nag_sparse_nsym_sort (f11zac) to reorder and sum or remove duplicates.

NE_ALLOC_FAIL
Memory allocation failed.

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 consult NAG for assistance.

6. Further Comments

The time taken by nag_sparse_nsym_sol for each iteration is roughly proportional to **nnz**.

The number of iterations required to achieve a prescribed accuracy cannot be easily determined a priori, as it can depend dramatically on the conditioning and spectrum of the preconditioned matrix of the coefficients $\bar{A} = M^{-1}A$.

6.1. Accuracy

On successful termination, the final residual $r_k = b - Ax_k$, where $k =$ **itn**, satisfies the termination criterion

$$\|r_k\|_\infty \leq \tau \times (\|b\|_\infty + \|A\|_\infty \|x_k\|_\infty).$$

The value of the final residual norm is returned in **rnorm**.

6.2. References

Saad Y and Schultz M (1986) GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems *SIAM J. Sci. Statist. Comput.* **7** 856–869.

Sleijpen G L G and Fokkema D R (1993) BiCGSTAB(ℓ) for linear equations involving matrices with complex spectrum *ETNA* **1** 11–32.

Sonneveld P (1989) CGS, a fast Lanczos-type solver for nonsymmetric linear systems *SIAM J. Sci. Statist. Comput.* **10** 36–52.

van der Vorst H (1989) Bi-CGSTAB, A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems *SIAM J. Sci. Statist. Comput.* **13** 631–644.

Young D (1971) *Iterative Solution of Large Linear Systems* Academic Press, New York.

7. See Also

nag_sparse_nsym_fac_sol (f11dcc)
nag_sparse_nsym_sort (f11zac)

8. Example

This example program solves a sparse nonsymmetric system of equations using the RGMRES method, with SSOR preconditioning.

8.1. Program Text

```
/* nag_sparse_nsym_sol(f11dec) Example Program.
 *
 * Copyright 1998 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nag_string.h>
#include <nagf11.h>

main()
{
  double  *a=0, *b=0, *x=0;
  double omega;
  double rnorm;
  double tol;

  Integer *icol=0, *irow=0;
```

```
Integer i, m, n;
Integer maxitn, itn;
Integer nnz;

Nag_SparseNsym_Method method;
Nag_SparseNsym_PrecType precon;
Nag_Sparse_Comm comm;
char char_enum[20];

Vprintf("f11dec Example Program Results\n");
/*  Skip heading in data file */
Vscanf("%*[^\n]");
Vscanf("%ld%*[^\n]",&n);
Vscanf("%ld%*[^\n]",&nnz);

Vscanf("%s",char_enum);
if (!strcmp(char_enum, "RGMRES"))
  method = Nag_SparseNsym_RGMRES;
else if (!strcmp(char_enum, "CGS"))
  method = Nag_SparseNsym_CGS;
else if (!strcmp(char_enum, "BiCGSTAB"))
  method = Nag_SparseNsym_BiCGSTAB;
else
  {
    Vprintf("Unrecognised string for method enum representation.\n");
    exit(EXIT_FAILURE);
  }

Vscanf("%s%*[^\n]",char_enum);
if (!strcmp(char_enum, "NoPrec"))
  precon = Nag_SparseNsym_NoPrec;
else if (!strcmp(char_enum, "SSORPrec"))
  precon = Nag_SparseNsym_SSORPrec;
else if (!strcmp(char_enum, "JacPrec"))
  precon = Nag_SparseNsym_JacPrec;
else
  {
    Vprintf("Unrecognised string for precon enum representation.\n");
    exit(EXIT_FAILURE);
  }

Vscanf("%lf%*[^\n]",&omega);
Vscanf("%ld%lf%ld%*[^\n]",&m,&tol,&maxitn);

x = NAG_ALLOC(n,double);
b = NAG_ALLOC(n,double);
a = NAG_ALLOC(nnz,double);
irow = NAG_ALLOC(nnz,Integer);
icol = NAG_ALLOC(nnz,Integer);
if (!irow || !icol || !a || !x || !b)
  {
    Vprintf("Allocation failure\n");
    exit(EXIT_FAILURE);
  }



/*  Read the matrix a */

for (i = 1; i <= nnz; ++i)
  Vscanf("%lf%ld%ld%*[^\n]",&a[i-1], &irow[i-1], &icol[i-1] );

/*  Read right-hand side vector b and initial approximate solution x */

for (i = 1; i <= n; ++i)
  Vscanf("%lf",&b[i-1]);
Vscanf("%*[^\n]");

for (i = 1; i <= n; ++i)
  Vscanf("%lf",&x[i-1]);
```

```
  Vscanf("%*[^\n]");

  /*  Solve Ax = b using f11dec */

  f11dec(method, precon, n, nnz, a, irow, icol, omega, b, m, tol,
         maxitn, x, &rnorm, &itn, &comm, NAGERR_DEFAULT);

  Vprintf("%s%10ld%s\n","Converged in",itn," iterations");
  Vprintf("%s%16.3e\n","Final residual norm =",rnorm);

  /*  Output x */
  Vprintf("          x\n");
  for (i = 1; i <= n; ++i)
    Vprintf(" %16.6e\n",x[i-1]);

  NAG_FREE(irow);
  NAG_FREE(icol);
  NAG_FREE(a);
  NAG_FREE(x);
  NAG_FREE(b);
  exit(EXIT_SUCCESS);
}
```

8.2. Program Data

```
f11dec Example Program Data
  5                     n
  16                    nnz
  RGMRES  SSORPrec      method, precon
  1.05                  omega
  1 1.e-10 1000         m, tol, maxitn
  2.   1    1
  1.   1    2
 -1.   1    4
 -3.   2    2
 -2.   2    3
  1.   2    5
  1.   3    1
  5.   3    3
  3.   3    4
  1.   3    5
 -2.   4    1
 -3.   4    4
 -1.   4    5
  4.   5    2
 -2.   5    3
 -6.   5    5         a[i-1], irow[i-1], icol[i-1], i=1,...,nnz
  0.  -7.  33.
-19. -28.             b[i-1], i=1,...,n
  0.   0.   0.
  0.   0.             x[i-1], i=1,...,n
```

8.3. Program Results

```
f11dec Example Program Results
Converged in        13 iterations
Final residual norm =       5.087e-09
          x
     1.000000e+00
     2.000000e+00
     3.000000e+00
     4.000000e+00
     5.000000e+00
```