nag_sparse_nsym_fac_sol (f11dcc)

1. Purpose

nag_sparse_nsym_fac_sol (f11dcc) 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, with incomplete LU preconditioning.

2. Specification

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

void nag_sparse_nsym_fac_sol(Nag_SparseNsym_Method method,  Integer n,
           Integer nnz, double a[], Integer la, Integer irow[], Integer icol[],
           Integer ipivp[], Integer ipivq[], Integer istr[], Integer idiag[],
           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 linear system of equations:

$$Ax = b,$$

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

nag_sparse_nsym_fac_sol uses the incomplete LU factorization determined by nag_sparse_nsym_fac (f11dac) as the preconditioning matrix. A call to nag_sparse_nsym_fac_sol must always be preceded by a call to nag_sparse_nsym_fac (f11dac). Alternative preconditioners for the same storage scheme are available by calling nag_sparse_nsym_sol (f11dec).

The matrix A, and the preconditioning matrix M, are represented in coordinate storage (CS) format (see Section 2.1.1 of the Chapter Introduction) in the arrays **a**, **irow** and **icol**, as returned from nag_sparse_nsym_fac (f11dac). The array **a** holds the non-zero entries in these matrices, 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**.

n

Input: the order of the matrix A. This **must** be the same value as was supplied in the preceding call to nag_sparse_nsym_fac (f11dac).

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

nnz

Input: the number of non-zero-elements in the matrix A. This **must** be the same value as was supplied in the preceding call to nag_sparse_nsym_fac (f11dac).

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

a[la]

Input: the values returned in the array **a** by a previous call to nag_sparse_nsym_fac (f11dac).

la

Input: the dimension of the arrays **a**, **irow** and **icol**. This must be the same value as returned by a previous call to nag_sparse_nsym_fac (f11dac).

Constraint: $\mathbf{la} \ge 2 \times \mathbf{nnz}$.

irow[la]

icol[la]

ipivp[n]

ipivq[n]

istr[n+1]

idiag[n]

Input: the values returned in the arrays **irow**, **icol**, **ipivp**, **ipivq**, **istr** and **idiag** by a previous call to nag_sparse_nsym_fac (f11dac).

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**, $0 < \mathbf{m} \le \min(\mathbf{n},50)$;

if **method** = **Nag_SparseNsym_BiCGSTAB**, $0 < \mathbf{m} \le \min(\mathbf{n},10)$.

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:

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

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

Constraint: $\mathbf{tol} < 1.0$.

maxitn

Input: the maximum number of iterations allowed.

Constraint: $\mathbf{maxitn} \ge 1$.

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.

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

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

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

NE_2_INT_ARG_LT
On entry, **la** = ⟨*value*⟩ while **nnz** = ⟨*value*⟩.
These parameters must satisfy **la** $\geq 2 \times$ **nnz**.

NE_REAL_ARG_GE
On entry, **tol** must not be greater than or equal to 1.0: **tol** = ⟨*value*⟩.

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_INVALID_CS
On entry, the CS representation of A is invalid. Check that the call to nag_sparse_nsym_fac_sol has been preceded by a valid call to nag_sparse_nsym_fac (f11dac), and that the arrays **a**, **irow** and **icol** have not been corrupted between the two calls.

NE_INVALID_CS_PRECOND
On entry, the CS representation of the preconditioning matrix M is invalid. Check that the call to nag_sparse_nsym_fac_sol has been preceded by a valid call to nag_sparse_nsym_fac (f11dac), and that the arrays **a**, **irow**, **icol**, **ipivp**, **ipivq** , **istr** and **idiag** have not been corrupted between the two calls.

NE_ALLOC_FAIL
Memory allocation failed.

6. Further Comments

The time taken by nag_sparse_nsym_fac_sol for each iteration is roughly proportional to the value of **nnzc** returned from the preceding call to nag_sparse_nsym_fac (f11dac).

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 coefficient matrix, $\bar{A} = M^{-1}A$.

Some illustrations of the application of nag_sparse_nsym_fac_sol to linear systems arising from the discretization of two-dimensional elliptic partial differential equations, and to random-valued randomly structured linear systems, can be found in Salvini and Shaw (1996).

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.

Salvini S A and Shaw G J (1996) An evaluation of new NAG Library solvers for large sparse unsymmetric linear systems *NAG Technical Report TR2/96*, NAG Ltd, Oxford.

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.

7. See Also

nag_sparse_nsym_fac (f11dac)

8. Example

This example program solves a sparse linear system of equations using the CGS method, with incomplete *LU* preconditioning.

8.1. Program Text

```
/* nag_sparse_nsym_fac_solve(f11dcc) 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 dtol;
  double *a=0, *b=0;
  double *x=0;
  double rnorm;
  double tol;

  Integer *irow, *icol;
  Integer *istr=0, *idiag, *ipivp=0, *ipivq=0;
  Integer i, m, n, nnzc;
  Integer lfill, npivm;
  Integer maxitn;
  Integer itn;
  Integer nnz;
  Integer num;

  Nag_SparseNsym_Method method;
  Nag_SparseNsym_Piv pstrat;
```

```
Nag_SparseNsym_Fact milu;
Nag_Sparse_Comm comm;

char char_enum[20];

Vprintf("f11dcc Example Program Results\n");

/*  Skip heading in data file */
Vscanf("%*[^\n]");

Vscanf("%ld%*[^\n]",&n);
Vscanf("%ld%*[^\n]",&nnz);
Vscanf("%s%*[^\n]",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("%ld%lf%*[^\n]",&lfill,&dtol);
Vscanf("%s%*[^\n]", char_enum);
if (!strcmp(char_enum, "NoPiv"))
  pstrat = Nag_SparseNsym_NoPiv;
else if (!strcmp(char_enum, "UserPiv"))
  pstrat = Nag_SparseNsym_UserPiv;
else if (!strcmp(char_enum, "PartialPiv"))
  pstrat = Nag_SparseNsym_PartialPiv;
else if (!strcmp(char_enum, "CompletePiv"))
  pstrat = Nag_SparseNsym_CompletePiv;
else
  {
    Vprintf("Unrecognised string for psstrat enum representation.\n");
    exit (EXIT_FAILURE);
  }

Vscanf("%s%*[^\n]", char_enum);
if (!strcmp(char_enum, "ModFact"))
  milu = Nag_SparseNsym_ModFact;
else if (!strcmp(char_enum, "UnModFact"))
  milu = Nag_SparseNsym_UnModFact;
else
  {
    Vprintf("Unrecognised string for milu enum representation.\n");
    exit (EXIT_FAILURE);
  }
Vscanf("%ld%lf%ld%*[^\n]",&m,&tol,&maxitn);

/*  Read the matrix a */

num = 2*nnz;
istr = NAG_ALLOC(n+1, Integer);
idiag = NAG_ALLOC(n, Integer);
ipivp = NAG_ALLOC(n, Integer);
ipivq = NAG_ALLOC(n, Integer);
x = NAG_ALLOC(n,double);
b = NAG_ALLOC(n,double);
a = NAG_ALLOC(num,double);
irow = NAG_ALLOC(num,Integer);
icol = NAG_ALLOC(num,Integer);
if (!istr || !idiag || !ipivp || !ipivq ||!irow || !icol || !a || !x || !b)
  {
    Vprintf("Allocation failure\n");
    exit (EXIT_FAILURE);
  }
```

```
    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]");

    /*  Calculate incomplete LU factorization */

    f11dac(n, nnz, &a, &num, &irow, &icol, lfill, dtol, pstrat, milu,
           ipivp, ipivq, istr, idiag, &nnzc, &npivm, NAGERR_DEFAULT);

    /*  Solve Ax = b using F11DCC */

    f11dcc(method, n, nnz, a, num, irow, icol, ipivp, ipivq, istr,
           idiag, 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(istr);
    NAG_FREE(idiag);
    NAG_FREE(ipivp);
    NAG_FREE(ipivq);
    NAG_FREE(irow);
    NAG_FREE(icol);
    NAG_FREE(a);
    NAG_FREE(x);
    NAG_FREE(b);

    exit(EXIT_SUCCESS);
  }
```

8.2. Program Data

```
f11dcc Example Program Data
  8                     n
  24                    nnz
  CGS                   method
  0  0.0                lfill, dtol
  CompletePiv           pstrat
  UnModFact             milu
  4  1.0e-10   100      m, tol, maxitn
  2.   1    1
 -1.   1    4
  1.   1    8
  4.   2    1
 -3.   2    2
  2.   2    5
 -7.   3    3
  2.   3    6
  3.   4    1
 -4.   4    3
  5.   4    4
  5.   4    7
 -1.   5    2
  8.   5    5
 -3.   5    7
 -6.   6    1
  5.   6    3
  2.   6    6
 -5.   7    3
 -1.   7    5
  6.   7    7
 -1.   8    2
  2.   8    6
  3.   8    8         a[i-1], irow[i-1], icol[i-1], i=1,...,nnz
  6.   8.  -9.  46.
 17.  21.  22.  34.   b[i-1], i=1,...,n
  0.   0.   0.   0.
  0.   0.   0.   0.   x[i-1], i=1,...,n
```

8.3. Program Results

```
f11dcc Example Program Results
Converged in         4 iterations
Final residual norm =       8.527e-14
           x
     1.000000e+00
     2.000000e+00
     3.000000e+00
     4.000000e+00
     5.000000e+00
     6.000000e+00
     7.000000e+00
     8.000000e+00
```