NAG Library Function Document

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,
     const double a[], const Integer irow[], const Integer icol[],
     double omega, const double b[], Integer m, double tol, Integer maxitn,
     double x[], double *rnorm, Integer *itn, Nag_Sparse_Comm *comm,
     NagError *fail)
```

3 Description

nag_sparse_nsym_sol (f11dec) 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 function 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 the f11 Chapter Introduction) in the arrays **a**, **irow** and **icol**. The array **a** holds the nonzero entries in the matrix, while **irow** and **icol** hold the corresponding row and column indices.

4 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

5 Arguments

1: **method** – Nag_SparseNsym_Method *Input*

On entry: specifies the iterative method to be used.

method = Nag_SparseNsym_RGMRES
The restarted generalized minimum residual method is used.

method = Nag_SparseNsym_CGS
The conjugate gradient squared method is used.

method = Nag_SparseNsym_BiCGSTAB
The bi-conjugate gradient stabilised (ℓ) method is used.

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

2: **precon** – Nag_SparseNsym_PrecType *Input*

On entry: specifies the type of preconditioning to be used.

precon = Nag_SparseNsym_NoPrec
No preconditioning.

precon = Nag_SparseNsym_SSORPrec
Symmetric successive-over-relaxation.

precon = Nag_SparseNsym_JacPrec
Jacobi.

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

3: **n** – Integer *Input*

On entry: the order of the matrix A.

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

4: **nnz** – Integer *Input*

On entry: the number of nonzero elements in the matrix A.

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

5: **a**[**nnz**] – const double *Input*

On entry: the nonzero 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 function nag_sparse_nsym_sort (f11zac) may be used to order the elements in this way.

6: **irow**[**nnz**] – const Integer *Input*
7: **icol**[**nnz**] – const Integer *Input*

On entry: the row and column indices of the nonzero elements supplied in **a**.

Constraints:

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 $i = 1, 2, \ldots, \mathbf{nnz} - 1$.

8: **omega** – double *Input*

On entry: if **precon** = Nag_SparseNsym_SSORPrec, **omega** is the relaxation argument ω to be used in the SSOR method. Otherwise **omega** need not be initialized and is not referenced.

Constraint: $0.0 < \textbf{omega} < 2.0$.

9: **b**[**n**] – const double *Input*

On entry: the right-hand side vector b.

10: **m** – Integer *Input*

On entry: 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, $0 < \textbf{m} \leq \min(\textbf{n}, 50)$;
if **method** = Nag_SparseNsym_BiCGSTAB, $0 < \textbf{m} \leq \min(\textbf{n}, 10)$.

11: **tol** – double *Input*

On entry: 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:

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

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

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

12: **maxitn** – Integer *Input*

On entry: the maximum number of iterations allowed.

Constraint: $\textbf{maxitn} \geq 1$.

13: **x**[**n**] – double *Input/Output*

On entry: an initial approximation to the solution vector x.

On exit: an improved approximation to the solution vector x.

14: **rnorm** – double * *Output*

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

15: **itn** – Integer * *Output*

On exit: the number of iterations carried out.

16: **comm** – Nag_Sparse_Comm * *Input/Output*

On entry/exit: a pointer to a structure of type Nag_Sparse_Comm whose members are used by the iterative solver.

17: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

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_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument **method** had an illegal value.

On entry, argument **precon** had an illegal value.

NE_INT_2

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

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

On entry, $\mathbf{nnz} = \langle value \rangle$, $\mathbf{n} = \langle value \rangle$.
Constraint: $1 \leq \mathbf{nnz} \leq \mathbf{n}^2$.

NE_INT_ARG_LT

On entry, $\mathbf{maxitn} = \langle value \rangle$.
Constraint: $\mathbf{maxitn} \geq 1$.

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 1$.

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.

NE_NONSYMM_MATRIX_DUP

A nonzero 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_NOT_REQ_ACC

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

NE_REAL

On entry, $\mathbf{omega} = \langle value \rangle$.
Constraint: $0.0 < \mathbf{omega} < 2.0$ when $\mathbf{precon} =$ Nag_SparseNsym_SSORPrec.

NE_REAL_ARG_GE

On entry, **tol** must not be greater than or equal to 1: **tol** $= \langle value \rangle$.

NE_ZERO_DIAGONAL_ELEM

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

7 Accuracy

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

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

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

8 Parallelism and Performance

Not applicable.

9 Further Comments

The time taken by nag_sparse_nsym_sol (f11dec) 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 $\bar{A} = M^{-1}A$.

10 Example

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

10.1 Program Text

```
/* nag_sparse_nsym_sol (f11dec) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 */

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

int main(void)
{
  double                  *a = 0, *b = 0, *x = 0;
  double                  omega;
  double                  rnorm;
  double                  tol;
  Integer                 exit_status = 0;
  Integer                 *icol = 0, *irow = 0;
  Integer                 i, m, n;
  Integer                 maxitn, itn;
  Integer                 nnz;
  char                    nag_enum_arg[40];
  Nag_SparseNsym_Method   method;
  Nag_SparseNsym_PrecType precon;
```

```
  Nag_Sparse_Comm         comm;
  NagError                fail;

  INIT_FAIL(fail);

  printf("nag_sparse_nsym_sol (f11dec) Example Program Results\n");
  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n]", &n);
#else
  scanf("%"NAG_IFMT"%*[^\n]", &n);
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n]", &nnz);
#else
  scanf("%"NAG_IFMT"%*[^\n]", &nnz);
#endif

#ifdef _WIN32
  scanf_s("%39s", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%39s", nag_enum_arg);
#endif
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  method = (Nag_SparseNsym_Method) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s("%39s%*[^\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%39s%*[^\n]", nag_enum_arg);
#endif
  precon = (Nag_SparseNsym_PrecType) nag_enum_name_to_value(nag_enum_arg);

#ifdef _WIN32
  scanf_s("%lf%*[^\n]", &omega);
#else
  scanf("%lf%*[^\n]", &omega);
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%lf%"NAG_IFMT"%*[^\n]", &m, &tol, &maxitn);
#else
  scanf("%"NAG_IFMT"%lf%"NAG_IFMT"%*[^\n]", &m, &tol, &maxitn);
#endif

  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)
    {
      printf("Allocation failure\n");
      exit_status = 1;
      goto END;
    }

  /* Read the matrix a */

  for (i = 1; i <= nnz; ++i)
#ifdef _WIN32
    scanf_s("%lf%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &a[i-1], &irow[i-1],
           &icol[i-1]);
#else
    scanf("%lf%"NAG_IFMT"%"NAG_IFMT"%*[^\n]", &a[i-1], &irow[i-1],
           &icol[i-1]);
```

```
#endif

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

  for (i = 1; i <= n; ++i)
#ifdef _WIN32
    scanf_s("%lf", &b[i-1]);
#else
    scanf("%lf", &b[i-1]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  for (i = 1; i <= n; ++i)
#ifdef _WIN32
    scanf_s("%lf", &x[i-1]);
#else
    scanf("%lf", &x[i-1]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Solve Ax = b using nag_sparse_nsym_sol (f11dec) */

  /* nag_sparse_nsym_sol (f11dec).
   * Solver with no Jacobi/SSOR preconditioning (nonsymmetric)
   */
  nag_sparse_nsym_sol(method, precon, n, nnz, a, irow, icol, omega, b, m, tol,
                      maxitn, x, &rnorm, &itn, &comm, &fail);

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

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

 END:
  NAG_FREE(irow);
  NAG_FREE(icol);
  NAG_FREE(a);
  NAG_FREE(x);
  NAG_FREE(b);

  return exit_status;
}
```

10.2 Program Data

```
nag_sparse_nsym_sol (f11dec) Example Program Data
  5                                                 n
  16                                                nnz
  Nag_SparseNsym_RGMRES Nag_SparseNsym_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
```

10.3 Program Results

```
nag_sparse_nsym_sol (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
```