

NAG Library Function Document

nag_sparse_sym_sol (f11jec)

1 Purpose

nag_sparse_sym_sol (f11jec) solves a real sparse symmetric system of linear equations, represented in symmetric coordinate storage format, using a conjugate gradient or Lanczos method, without preconditioning, with Jacobi or with SSOR preconditioning.

2 Specification

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

void nag_sparse_sym_sol (Nag_SparseSym_Method method,
    Nag_SparseSym_PrecType precon, Integer n, Integer nnz, const double a[],
    const Integer irow[], const Integer icol[], double omega,
    const double b[], double tol, Integer maxitn, double x[], double *rnorm,
    Integer *itn, Nag_Sparse_Comm *comm, NagError *fail)
```

3 Description

nag_sparse_sym_sol (f11jec) solves a real sparse symmetric linear system of equations:

$$Ax = b,$$

using a preconditioned conjugate gradient method (see Barrett *et al.* (1994)), or a preconditioned Lanczos method based on the algorithm SYMMLQ (Paige and Saunders (1975)). The conjugate gradient method is more efficient if A is positive definite, but may fail to converge for indefinite matrices. In this case the Lanczos method should be used instead. For further details see Barrett *et al.* (1994).

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 Cholesky (IC) preconditioning see nag_sparse_sym_chol_sol (f11jcc).

The matrix A is represented in symmetric coordinate storage (SCS) format (see the f11 Chapter Introduction) in the arrays **a**, **irow** and **icol**. The array **a** holds the nonzero entries in the lower triangular part of the matrix, while **irow** and **icol** hold the corresponding row and column indices.

4 References

Barrett R, Berry M, Chan T F, Demmel J, Donato J, Dongarra J, Eijkhout V, Pozo R, Romine C and Van der Vorst H (1994) *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods* SIAM, Philadelphia

Paige C C and Saunders M A (1975) Solution of sparse indefinite systems of linear equations *SIAM J. Numer. Anal.* **12** 617–629

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

5 Arguments

- 1: **method** – Nag_SparseSym_Method *Input*
On entry: specifies the iterative method to be used.
method = Nag_SparseSym_CG
The conjugate gradient method is used.
method = Nag_SparseSym_Lanczos
The Lanczos method (SYMMLQ) is used.
Constraint: **method** = Nag_SparseSym_CG or Nag_SparseSym_Lanczos.
- 2: **precon** – Nag_SparseSym_PrecType *Input*
On entry: specifies the type of preconditioning to be used.
precon = Nag_SparseSym_NoPrec
No preconditioning is used.
precon = Nag_SparseSym_SSORPrec
Symmetric successive-over-relaxation is used.
precon = Nag_SparseSym_JacPrec
Jacobi preconditioning is used.
Constraint: **precon** = Nag_SparseSym_NoPrec, Nag_SparseSym_SSORPrec or Nag_SparseSym_JacPrec.
- 3: **n** – Integer *Input*
On entry: the order of the matrix A .
Constraint: $n \geq 1$.
- 4: **nnz** – Integer *Input*
On entry: the number of nonzero elements in the lower triangular part of the matrix A .
Constraint: $1 \leq \text{nnz} \leq n \times (n + 1)/2$.
- 5: **a[nnz]** – const double *Input*
On entry: the nonzero elements of the lower triangular part 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_sym_sort (f11zbc) 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_sym_sort (f11zbc));
 $1 \leq \text{irow}[i] \leq n$ and $1 \leq \text{icol}[i] \leq \text{irow}[i]$, for $i = 0, 1, \dots, \text{nnz} - 1$;
irow $[i - 1] < \text{irow}[i]$ or **irow** $[i - 1] = \text{irow}[i]$ and **icol** $[i - 1] < \text{icol}[i]$, for $i = 1, 2, \dots, \text{nnz} - 1$.
- 8: **omega** – double *Input*
On entry: if **precon** = Nag_SparseSym_SSORPrec, **omega** is the relaxation argument ω to be used in the SSOR method. Otherwise **omega** need not be initialized.
Constraint: $0.0 \leq \text{omega} \leq 2.0$.

- 9: **b[n]** – const double *Input*
On entry: the right-hand side vector b .
- 10: **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:

$$\|r_k\|_\infty \leq \tau \times (\|b\|_\infty + \|A\|_\infty \|x_k\|_\infty).$$
If **tol** ≤ 0.0 , $\tau = \max(\sqrt{\epsilon}, \sqrt{n}, \epsilon)$ is used, where ϵ is the *machine precision*. Otherwise $\tau = \max(\mathbf{tol}, 10\epsilon, \sqrt{n}, \epsilon)$ is used.
Constraint: **tol** < 1.0 .
- 11: **maxitn** – Integer *Input*
On entry: the maximum number of iterations allowed.
Constraint: **maxitn** ≥ 1 .
- 12: **x[n]** – double *Input/Output*
On entry: an initial approximation of the solution vector x .
On exit: an improved approximation to the solution vector x .
- 13: **rnorm** – double * *Output*
On exit: the final value of the residual norm $\|r_k\|_\infty$, where k is the output value of **itn**.
- 14: **itn** – Integer * *Output*
On exit: the number of iterations carried out.
- 15: **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.
- 16: **fail** – NagError * *Input/Output*
The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

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.

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_COEFF_NOT_POS_DEF

The matrix of coefficients appears not to be positive definite (conjugate gradient method only).

NE_INT_2

On entry, **nnz** = $\langle value \rangle$, **n** = $\langle value \rangle$.
 Constraint: $1 \leq \mathbf{nnz} \leq \mathbf{n} \times (\mathbf{n} + 1)/2$.

NE_INT_ARG_LT

On entry, **maxitn** = $\langle value \rangle$.
 Constraint: **maxitn** ≥ 1 .
 On entry, **n** = $\langle value \rangle$.
 Constraint: **n** ≥ 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_NOT_REQ_ACC

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

NE_PRECOND_NOT_POS_DEF

The preconditioner appears not to be positive definite.

NE_REAL

On entry, **omega** = $\langle value \rangle$.
 Constraint: $0.0 \leq \mathbf{omega} \leq 2.0$.

NE_REAL_ARG_GE

On entry, **tol** must not be greater than or equal to 1.0: **tol** = $\langle value \rangle$.

NE_SYMM_MATRIX_DUP

A nonzero element has been supplied which does not lie in the lower triangular part of 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:

$$1 \leq \mathbf{irow}[i] \leq \mathbf{n} \text{ and } 1 \leq \mathbf{icol}[i] \leq \mathbf{irow}[i], \text{ for } i = 0, 1, \dots, \mathbf{nnz} - 1$$

$$\mathbf{irow}[i - 1] < \mathbf{irow}[i], \text{ or}$$

$$\mathbf{irow}[i - 1] = \mathbf{irow}[i] \text{ and } \mathbf{icol}[i - 1] < \mathbf{icol}[i], \text{ for } i = 1, 2, \dots, \mathbf{nnz} - 1.$$

Call `nag_sparse_sym_sort (f11zbc)` to reorder and sum or remove duplicates.

NE_ZERO_DIAGONAL_ELEM

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 $k = \mathbf{itn}$, 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 **rnrm**.

8 Parallelism and Performance

`nag_sparse_sym_sol (f11jec)` is not threaded in any implementation.

9 Further Comments

The time taken by `nag_sparse_sym_sol` (f11jec) for each iteration is roughly proportional to **nnz**. One iteration with the Lanczos method (SYMMLQ) requires a slightly larger number of operations than one iteration with the conjugate gradient method.

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 symmetric positive definite system of equations using the conjugate gradient method, with SSOR preconditioning.

10.1 Program Text

```
/* nag_sparse_sym_sol (f11jec) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#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, *irow;
    Integer i, n, maxitn, itn, nnz;
    char nag_enum_arg[40];
    Nag_SparseSym_Method method;
    Nag_SparseSym_PrecType precon;
    Nag_Sparse_Comm comm;
    NagError fail;

    INIT_FAIL(fail);

    printf("nag_sparse_sym_sol (f11jec) Example Program Results\n");

    /* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif

    /* Read algorithmic parameters */
#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
}
```

```

    scanf("%" NAG_IFMT "%*[\n]", &nnz);
#endif

#ifdef _WIN32
    scanf_s("%39s", nag_enum_arg, (unsigned)_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_SparseSym_Method) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
    scanf_s("%39s%*[\n]", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
    scanf("%39s%*[\n]", nag_enum_arg);
#endif
precon = (Nag_SparseSym_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("%lf%" NAG_IFMT "%*[\n]", &tol, &maxitn);
#else
    scanf("%lf%" NAG_IFMT "%*[\n]", &tol, &maxitn);
#endif

/* Allocate memory */
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 */
/* nag_sparse_sym_sol (f11jec).
 * Solver with Jacobi, SSOR, or no preconditioning
 * (symmetric)
 */
nag_sparse_sym_sol(method, precon, n, nnz, a, irow, icol, omega, b, tol,
                  maxitn, x, &rnrm, &itn, &comm, &fail);

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

/* Output x */
for (i = 1; i <= n; ++i)
    printf(" %16.4e\\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_sym_sol (f11jec) Example Program Data

7				n
16				nnz
Nag_SparseSym_CG	Nag_SparseSym_SSORPrec			method, precon
1.1				omega
1.0E-6	100			tol, maxitn
4.	1	1		
1.	2	1		
5.	2	2		
2.	3	3		
2.	4	2		
3.	4	4		
-1.	5	1		
1.	5	4		
4.	5	5		
1.	6	2		
-2.	6	5		
3.	6	6		
2.	7	1		
-1.	7	2		
-2.	7	3		
5.	7	7		a[i-1], irow[i-1], icol[i-1], i=1,...,nnz
15.	18.	-8.	21.	
11.	10.	29.		b[i-1], i=1,...,n
0.	0.	0.	0.	
0.	0.	0.		x[i-1], i=1,...,n

10.3 Program Results

```
nag_sparse_sym_sol (f11jec) Example Program Results
Converged in          6 iterations
Final residual norm =      5.026e-06
1.0000e-00
2.0000e+00
3.0000e+00
4.0000e+00
5.0000e+00
6.0000e+00
7.0000e+00
```
