

NAG Library Function Document

nag_sparse_sym_chol_sol (f11jcc)

1 Purpose

nag_sparse_sym_chol_sol (f11jcc) solves a real sparse symmetric system of linear equations, represented in symmetric coordinate storage format, using a conjugate gradient or Lanczos method, with incomplete Cholesky preconditioning.

2 Specification

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

void nag_sparse_sym_chol_sol (Nag_SparseSym_Method method, Integer n,
    Integer nnz, const double a[], Integer la, const Integer irow[],
    const Integer icol[], const Integer ipiv[], const Integer istr[],
    const double b[], double tol, Integer maxitn, double x[], double *rnorm,
    Integer *itn, Nag_Sparse_Comm *comm, NagError *fail)
```

3 Description

nag_sparse_sym_chol_sol (f11jcc) solves a real sparse symmetric linear system of equations:

$$Ax = b,$$

using a preconditioned conjugate gradient method (Meijerink and Van der Vorst (1977)), 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).

nag_sparse_sym_chol_sol (f11jcc) uses the incomplete Cholesky factorization determined by nag_sparse_sym_chol_fac (f11jac) as the preconditioning matrix. A call to nag_sparse_sym_chol_sol (f11jcc) must always be preceded by a call to nag_sparse_sym_chol_fac (f11jac). Alternative preconditioners for the same storage scheme are available by calling nag_sparse_sym_sol (f11jec).

The matrix A , and the preconditioning matrix M , are represented in symmetric coordinate storage (SCS) format (see the f11 Chapter Introduction) in the arrays **a**, **irow** and **icol**, as returned from nag_sparse_sym_chol_fac (f11jac). The array **a** holds the nonzero entries in the lower triangular parts of these matrices, 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

Meijerink J and Van der Vorst H (1977) An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix *Math. Comput.* **31** 148–162

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

Salvini S A and Shaw G J (1995) An evaluation of new NAG Library solvers for large sparse symmetric linear systems *NAG Technical Report TR1/95*

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: **n** – Integer *Input*
On entry: the order of the matrix A . This **must** be the same value as was supplied in the preceding call to nag_sparse_sym_chol_fac (f11jac).
Constraint: $n \geq 1$.
- 3: **nnz** – Integer *Input*
On entry: the number of nonzero elements in the lower triangular part of the matrix A . This **must** be the same value as was supplied in the preceding call to nag_sparse_sym_chol_fac (f11jac).
Constraint: $1 \leq \text{nnz} \leq n \times (n + 1)/2$.
- 4: **a[la]** – const double *Input*
On entry: the values returned in array **a** by a previous call to nag_sparse_sym_chol_fac (f11jac).
- 5: **la** – Integer *Input*
On entry: the second dimension of the arrays **a**, **irow** and **icol**. This **must** be the same value as returned by a previous call to nag_sparse_sym_chol_fac (f11jac).
Constraint: $la \geq 2 \times \text{nnz}$.
- 6: **irow[la]** – const Integer *Input*
- 7: **icol[la]** – const Integer *Input*
- 8: **ipiv[n]** – const Integer *Input*
- 9: **istr[n + 1]** – const Integer *Input*
On entry: the values returned in the arrays **irow**, **icol**, **ipiv** and **istr** by a previous call to nag_sparse_sym_chol_fac (f11jac).
- 10: **b[n]** – const double *Input*
On entry: the right-hand side vector b .
- 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:
$$\|r_k\|_\infty \leq \tau \times (\|b\|_\infty + \|A\|_\infty \|x_k\|_\infty).$$
If **tol** ≤ 0.0 , $\tau = \max(\sqrt{\epsilon}, \sqrt{n}\epsilon)$ is used, where ϵ is the *machine precision*. Otherwise $\tau = \max(\text{tol}, 10\epsilon, \sqrt{n}\epsilon)$ is used.
Constraint: **tol** < 1.0 .

- 12: **maxitn** – Integer *Input*
On entry: the maximum number of iterations allowed.
Constraint: **maxitn** ≥ 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 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_2_INT_ARG_LT

On entry, **la** = $\langle value \rangle$ while **nnz** = $\langle value \rangle$. These arguments must satisfy **la** $\geq 2 \times$ **nnz**.

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.

NE_COEFF_NOT_POS_DEF

The matrix of coefficients appears not to be positive definite.

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_INVALID_SCS

The SCS representation of the matrix A is invalid. Check that the call to `nag_sparse_sym_chol_sol` (f11jcc) has been preceded by a valid call to `nag_sparse_sym_chol_fac` (f11jac), and that the arrays **a**, **irow** and **icol** have not been corrupted between the two calls.

NE_INVALID_SCS_PRECOND

The SCS representation of the preconditioning matrix M is invalid. Check that the call to `nag_sparse_sym_chol_sol` (f11jcc) has been preceded by a valid call to `nag_sparse_sym_chol_fac` (f11jac), and that the arrays **a**, **irow**, **icol**, **ipiv** and **istr** have not been corrupted between the two calls.

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_ARG_GE

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

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 **rnorm**.

8 Parallelism and Performance

`nag_sparse_sym_chol_sol` (f11jcc) is not threaded in any implementation.

9 Further Comments

The time taken by `nag_sparse_sym_chol_sol` (f11jcc) for each iteration is roughly proportional to the value of **nnzc** returned from the preceding call to `nag_sparse_sym_chol_fac` (f11jac). 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 $\bar{A} = M^{-1}A$.

Some illustrations of the application of `nag_sparse_sym_chol_sol` (f11jcc) to linear systems arising from the discretization of two-dimensional elliptic partial differential equations, and to random-valued randomly structured symmetric positive definite linear systems, can be found in Salvini and Shaw (1995).

10 Example

This example program solves a symmetric positive definite system of equations using the conjugate gradient method, with incomplete Cholesky preconditioning.

10.1 Program Text

```
/* nag_sparse_sym_chol_sol (f11jcc) 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 dtol;
    double *a = 0, *b = 0;
    double *x = 0;
    double rnorm, dscale;
    double tol;
    Integer exit_status = 0;
    Integer *icol = 0;
    Integer *ipiv = 0, nnzc, *irow = 0, *istr = 0;
    Integer i;
    Integer n;
    Integer lfill, npivm;
    Integer maxitn;
    Integer itn;
    Integer nnz;
    Integer num;
    char nag_enum_arg[40];
    Nag_SparseSym_Method method;
    Nag_SparseSym_Piv pstrat;
    Nag_SparseSym_Fact mic;
    Nag_Sparse_Comm comm;
    NagError fail;

    INIT_FAIL(fail);

    printf("nag_sparse_sym_chol_sol (f11jcc) 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
}
```

```

#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%lf%*[\n]", &lfill, &dtol);
#else
    scanf("%" NAG_IFMT "%lf%*[\n]", &lfill, &dtol);
#endif
#ifdef _WIN32
    scanf_s("%39s%*[\n]", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
    scanf("%39s%*[\n]", 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%lf%*[\n]", nag_enum_arg, (unsigned)_countof(nag_enum_arg),
        &dscale);
#else
    scanf("%39s%lf%*[\n]", nag_enum_arg, &dscale);
#endif
    mic = (Nag_SparseSym_Fact) 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
    pstrat = (Nag_SparseSym_Piv) nag_enum_name_to_value(nag_enum_arg);

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

    /* Read the matrix a */

    /* Allocate memory */
    num = 2 * nnz;
    irow = NAG_ALLOC(num, Integer);
    icol = NAG_ALLOC(num, Integer);
    a = NAG_ALLOC(num, double);
    b = NAG_ALLOC(n, double);
    x = NAG_ALLOC(n, double);
    istr = NAG_ALLOC(n + 1, Integer);
    ipiv = NAG_ALLOC(num, Integer);

    if (!irow || !icol || !a || !x || !istr || !ipiv) {
        printf("Allocation failure\n");
        return EXIT_FAILURE;
    }

    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

    /* Calculate incomplete Cholesky factorization */

    /* nag_sparse_sym_chol_fac (f11jac).
     * Incomplete Cholesky factorization (symmetric)
     */
    nag_sparse_sym_chol_fac(n, nnz, &a, &num, &irow, &icol, lfill, dtol, mic,
                           dscale, pstrat, ipiv, istr, &nnzc, &npivm, &comm,
                           &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_sparse_sym_chol_fac (f11jac).\\n%s\\n",
              fail.message);
        exit_status = 1;
        goto END;
    }

    /* Solve Ax = b */

    /* nag_sparse_sym_chol_sol (f11jcc).
     * Solver with incomplete Cholesky preconditioning
     * (symmetric)
     */
    nag_sparse_sym_chol_sol(method, n, nnz, a, num, irow, icol, ipiv, istr, b,
                           tol, maxitn, x, &rnorm, &itn, &comm, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_sparse_sym_chol_sol (f11jcc).\\n%s\\n",
              fail.message);
        exit_status = 1;
        goto END;
    }

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

    /* 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(b);
    NAG_FREE(x);
    NAG_FREE(ipiv);
    NAG_FREE(istr);

    return exit_status;
}

```

10.2 Program Data

```
nag_sparse_sym_chol_sol (f11jcc) Example Program Data
  7                                n
 16                               nnz
 1 0.0                           lfill, dtol
Nag_SparseSym_CG                 method
Nag_SparseSym_UnModFact 0.0      mic  dscale
Nag_SparseSym_MarkPiv            pstrat
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_chol_sol (f11jcc) Example Program Results
Converged in      1 iterations
Final residual norm =      0.000e+00
1.0000e+00
2.0000e+00
3.0000e+00
4.0000e+00
5.0000e+00
6.0000e+00
7.0000e+00
```
