NAG Library Function Document
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

```c
#include <nag.h>
#include <nagf11.h>
void nag_sparse_nsym_fac_sol (Nag_SparseNsym_Method method, Integer n,
   Integer nnz, const double a[], Integer la, const Integer irow[],
   const Integer icol[], const Integer ipivp[], const Integer ipivq[],
   const Integer istr[], const Integer idiag[], 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_fac_sol (f11dcc) 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(\(\ell\)) method (see Van der Vorst (1989), Sleijpen and Fokkema (1993)).

nag_sparse_nsym_fac_sol (f11dcc) uses the incomplete LU factorization determined by nag_sparse_nsym_fac (f11dac) as the preconditioning matrix. A call to nag_sparse_nsym_fac_sol (f11dcc) 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 the f11 Chapter Introduction) in the arrays \( a \), \( irow \) and \( icol \), as returned from nag_sparse_nsym_fac (f11dac). The array \( a \) holds the nonzero entries in these matrices, while \( irow \) and \( icol \) hold the corresponding row and column indices.

4 References
Sleijpen G L G and Fokkema D R (1993) BiCGSTAB(\(\ell\)) for linear equations involving matrices with complex spectrum *ETNA* 1 11–32
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
    
    Then the bi-conjugate gradient stabilised (ℓ) method is used.
    
    *Constraint*: **method** = Nag_SparseNsym_RGMRES, Nag_SparseNsym_CGS or Nag_SparseNsym_BiCGSTAB.

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_nsym_fac (f11dac).
    
    *Constraint*: n ≥ 1.

3:  **nnz** – Integer
    
    *Input*
    
    *On entry*: the number of nonzero-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 ≤ nnz ≤ n².

4:  **a[la]** – const double
    
    *Input*
    
    *On entry*: the values returned in the array a by a previous call to nag_sparse_nsym_fac (f11dac).

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_nsym_fac (f11dac).
    
    *Constraint*: la ≥ 2 × nnz.

6:  **irow[la]** – const Integer
    
    *Input*
    
    *On entry*: the values returned in the arrays irow, icol, ipivp, ipivq, istr and idia by a previous call to nag_sparse_nsym_fac (f11dac).

7:  **icol[la]** – const Integer
    
    *Input*

8:  **ipivp[n]** – const Integer
    
    *Input*

9:  **ipivq[n]** – const Integer
    
    *Input*

10: **istr[n + 1]** – const Integer
    
    *Input*

11: **idiag[n]** – const Integer
    
    *Input*

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

13: **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 < m ≤ min(n, 50);  
if method = Nag_SparseNsym_BiCGSTAB, 0 < m ≤ min(n, 10).

14: 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 \( \epsilon \) is the machine precision. Otherwise \( \tau = \max(tol, 10\epsilon, \sqrt{n}, \epsilon) \) is used.

Constraint: tol < 1.0.

15: maxitn – Integer  
Input  
On entry: the maximum number of iterations allowed.

Constraint: maxitn ≥ 1.

16: 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.

17: rnorm – double *  
Output  
On exit: the final value of the residual norm \( \|r_k\|_\infty \), where k is the output value of itn.

18: itn – Integer *  
Output  
On exit: the number of iterations carried out.

19: 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.

20: fail – NagError *  
Input/Output  
The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_2_INT_ARG_LT
On entry, la = (value) while nnz = (value). These arguments must satisfy la ≥ 2 × 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.

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_ALG_FAIL
Algorithmic breakdown. A solution is returned, although it is possible that it is completely inaccurate.
**NE_ALLOC_FAIL**

Dynamic memory allocation failed.

**NE_BAD_PARAM**

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

**NE_INT_2**

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

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

On entry, \( nnz = \langle \text{value} \rangle, n = \langle \text{value} \rangle \).
Constraint: \( 1 \leq nnz \leq n^2 \).

**NE_INT_ARG_LT**

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

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

**NE_INVALID_CS**

On entry, the CS representation of \( A \) is invalid. Check that the call to nag_sparse_nsym_fac_sol (f11dcc) 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 (f11dcc) has been preceded by a valid call to nag_sparse_nsym_fac (f11dac), and that the arrays \( a, irow, icol, ipivp, ipivq, istr \) and \( idia \) have not been corrupted between the two calls.

**NE_NOT_REQ_ACC**

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

**NE_REAL_ARG_GE**

On entry, `tol` must not be greater than or equal to 1.0: `tol` = \langle \text{value} \rangle.

7 **Accuracy**

On successful termination, the final residual \( r_k = b - Ax_k \), where \( k = \text{itn} \), 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**

Not applicable.
9 Further Comments

The time taken by nag_sparse_nsym_fac_sol (f11dcc) 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, \( A = M^{-1}A \).

Some illustrations of the application of nag_sparse_nsym_fac_sol (f11dcc) 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).

10 Example

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

10.1 Program Text

```c
/* nag_sparse_nsym_fac_solve (f11dcc) 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 dtol;
    double *a = 0, *b = 0;
    double *x = 0;
    double rnorm;
    double tol;
    Integer exit_status = 0;
    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;
    char nag_enum_arg[40];
    Nag_SparseNsym_Method method;
    Nag_SparseNsym_Piv pstrat;
    Nag_SparseNsym_Fact milu;
    Nag_Sparse_Comm comm;
    NagError fail;

    INIT_FAIL(fail);

    printf("nag_sparse_nsym_fac_sol (f11dcc) Example Program Results\n");

    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("\%*[\`\n"]);
    #else
        scanf("\%*[\`\n"]);
    #endif
```
```c
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"%[^\n]", &n);
#else
    scanf("%"NAG_IFMT"%[^\n]", &n);
#endif
#endif _WIN32
scanf_s("%"NAG_IFMT"%[^\n]", &nnz);
#else
scanf("%"NAG_IFMT"%[^\n]", &nnz);
#endif
#ifdef _WIN32
    scanf_s("%39s%[^\n]", nag_enum_arg, _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_SparseNsym_Method) nag_enum_name_to_value(nag_enum_arg);
#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, _countof(nag_enum_arg));
#else
    scanf("%39s%[^\n]", nag_enum_arg);
#endif
pstrat = (Nag_SparseNsym_Piv) 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
milu = (Nag_SparseNsym_Fact) nag_enum_name_to_value(nag_enum_arg);
#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
/* 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)
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
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)
#if defined _WIN32
    scanf_s("%lf", &b[i-1]);
#else
    scanf("%lf", &b[i-1]);
#endif
#if defined _WIN32
    scanf("%*[\n]");
#else
    scanf("%*[\n]");
#endif

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

/* Calculate incomplete LU factorization */

/* nag_sparse_nsym_fac (f11dac).
 * Incomplete LU factorization (nonsymmetric)
 */
nag_sparse_nsym_fac(n, nnz, &a, &num, &irow, &icol, lfill, dtol, pstrat,
                   milu, ipivp, ipivq, istr, idiag, &nnzc, &npivm, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_sparse_nsym_fac (f11dac).
%s
", fail.message);
    exit_status = 1;
    goto END;
}

/* nag_sparse_nsym_fac_sol (f11dcc).
 * Solve Ax = b using nag_sparse_nsym_fac_sol (f11dcc) */
nag_sparse_nsym_fac_sol(method, n, nnz, a, num, irow, icol, ipivp, ipivq,
                        istr, idiag, b, m, tol, maxitn, x, &rnorm, &itn,
                        &comm, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_sparse_nsym_fac_sol (f11dcc).
%s
", 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 */

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

END:
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);

return exit_status;
}

10.2 Program Data

nag_sparse_nsym_fac_sol (f11dcc) Example Program Data

8
24 nnz
Nag_SparseNsym_CGS method
0 0.0 lfill, dtol
Nag_SparseNsym_CompletePiv pstrat
Nag_SparseNsym_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
17. 21. 22. 34. 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_nsym_fac_sol (f11dcc) Example Program Results
Converged in 4 iterations
Final residual norm = 2.132e-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