1 Purpose

nag_sparse_nherm_sol (f11dsc) solves a complex sparse non-Hermitian system of linear equations, represented in coordinate storage format, using a restarted generalized minimal residual (RGMRES), conjugate gradient squared (CGS), stabilized bi-conjugate gradient (Bi-CGSTAB), or transpose-free quasi-minimal residual (TFQMR) method, without preconditioning, with Jacobi, or with SSOR preconditioning.

2 Specification

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

void nag_sparse_nherm_sol (Nag_SparseNsym_Method method,
                         Nag_SparseNsym_PrecType precon, Integer n, Integer nnz,
                         const Complex a[], const Integer irow[], const Integer icol[],
                         double omega, const Complex b[], Integer m, double tol, Integer maxitn,
                         Complex x[], double *rnorm, Integer *itn, NagError *fail)
```

3 Description

nag_sparse_nherm_sol (f11dsc) solves a complex sparse non-Hermitian system of linear equations:

\[ Ax = b, \]

using an RGMRES (see Saad and Schultz (1986)), CGS (see Sonneveld (1989)), Bi-CGSTAB(\(\ell\)) (see Van der Vorst (1989) and Sleijpen and Fokkema (1993)), or TFQMR (see Freund and Nachtigal (1991) and Freund (1993)) method.

nag_sparse_nherm_sol (f11dsc) 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_nherm_fac_sol (f11dqc).

The matrix \(A\) is represented in coordinate storage (CS) format (see Section 2.1.1 in 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.

nag_sparse_nherm_sol (f11dsc) is a Black Box function which calls nag_sparse_nherm_basic_setup (f11brc), nag_sparse_nherm_basic_solver (f11bsc) and nag_sparse_nherm_basic_diagnostic (f11btc). If you wish to use an alternative storage scheme, preconditioner, or termination criterion, or require additional diagnostic information, you should call these underlying functions directly.

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: \textbf{method} – Nag_SparseNsym_Method \textit{Input}

\textit{On entry:} specifies the iterative method to be used.

\textit{method} = Nag_SparseNsym_RGMRES
Restarted generalized minimum residual method.

\textit{method} = Nag_SparseNsym_CGS
Conjugate gradient squared method.

\textit{method} = Nag_SparseNsym_BiCGSTAB
Bi-conjugate gradient stabilized (\ell) method.

\textit{method} = Nag_SparseNsym_TFQMR
Transpose-free quasi-minimal residual method.

\textit{Constraint:} \textit{method} = Nag_SparseNsym_RGMRES, Nag_SparseNsym_CGS, Nag_SparseNsym_BiCGSTAB or Nag_SparseNsym_TFQMR.

2: \textbf{precon} – Nag_SparseNsym_PrecType \textit{Input}

\textit{On entry:} specifies the type of preconditioning to be used.

\textit{precon} = Nag_SparseNsym_NoPrec
No preconditioning.

\textit{precon} = Nag_SparseNsym_JacPrec
Jacobi.

\textit{precon} = Nag_SparseNsym_SSORPrec
Symmetric successive-over-relaxation (SSOR).

\textit{Constraint:} \textit{precon} = Nag_SparseNsym_NoPrec, Nag_SparseNsym_JacPrec or Nag_SparseNsym_SSORPrec.

3: \textbf{n} – Integer \textit{Input}

\textit{On entry:} \textit{n}, the order of the matrix \textit{A}.

\textit{Constraint:} \textit{n} \geq 1.

4: \textbf{nnz} – Integer \textit{Input}

\textit{On entry:} the number of nonzero elements in the matrix \textit{A}.

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

5: \textbf{a[nnz]} – const Complex \textit{Input}

\textit{On entry:} the nonzero elements of the matrix \textit{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_nherm_sort (f11znc) may be used to order the elements in this way.
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 \( \text{nag_sparse_nherm_sort (f11znc)} \)):
  
  \[
  1 \leq irow[i] \leq n \quad \text{and} \quad 1 \leq icol[i] \leq n, \quad \text{for} \quad i = 0, 1, \ldots, \text{nnz} - 1;
  
  \text{either} \quad irow[i - 1] < irow[i] \quad \text{or both} \quad irow[i - 1] = irow[i] \quad \text{and} \quad icol[i - 1] < icol[i], \quad \text{for} \quad i = 1, 2, \ldots, \text{nnz} - 1.
  \]

8: \( \omega \) – double

On entry: if \( \text{precon} = \text{Nag\_SparseNsym\_SSORPrec} \), \( \omega \) is the relaxation parameter \( \omega \) to be used in the SSOR method. Otherwise \( \omega \) need not be initialized and is not referenced.

Constraint: \( 0.0 < \omega < 2.0 \).

9: \( b[n] \) – const Complex

On entry: the right-hand side vector \( b \).

10: \( m \) – Integer

On entry: if \( \text{method} = \text{Nag\_SparseNsym\_RGMRES} \), \( m \) is the dimension of the restart subspace.

If \( \text{method} = \text{Nag\_SparseNsym\_BiCGSTAB} \), \( m \) is the order \( \ell \) of the polynomial Bi-CGSTAB method.

Otherwise, \( m \) is not referenced.

Constraints:

\[
\begin{align*}
\text{if} \quad \text{method} = \text{Nag\_SparseNsym\_RGMRES}, \quad 0 < m \leq \min(n, 50); \\
\text{if} \quad \text{method} = \text{Nag\_SparseNsym\_BiCGSTAB}, \quad 0 < m \leq \min(n, 10).
\end{align*}
\]

11: \( \text{tol} \) – double

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 \( \text{tol} \leq 0.0 \), \( \tau = \max(\sqrt{\epsilon}, 10\epsilon, \sqrt{\text{tol}}) \) is used, where \( \epsilon \) is the machine precision. Otherwise \( \tau = \max(\text{tol}, 10\epsilon, \sqrt{\text{tol}}) \) is used.

Constraint: \( \text{tol} < 1.0 \).

12: \( \text{maxitn} \) – Integer

On entry: the maximum number of iterations allowed.

Constraint: \( \text{maxitn} \geq 1 \).

13: \( x[n] \) – Complex

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

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

14: \( \text{rnorm} \) – double *

On exit: the final value of the residual norm \( \|r_k\|_\infty \), where \( k \) is the output value of \( \text{itn} \).
15: \texttt{itn} – Integer \* \\
\textit{On exit:} the number of iterations carried out.

16: \texttt{fail} – NagError \* \\
\textit{Input/Output} \\
The NAG error argument (see Section 3.6 in the Essential Introduction).

6 \ Error Indicators and Warnings

\textbf{NE\_ACCURACY} \\
The required accuracy could not be obtained. However, a reasonable accuracy may have been achieved.

\textbf{NE\_ALG\_FAIL} \\
Algorithmic breakdown. A solution is returned, although it is possible that it is completely inaccurate.

\textbf{NE\_ALLOC\_FAIL} \\
Dynamic memory allocation failed. \\
See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE\_BAD\_PARAM} \\
On entry, argument \texttt{<value>} had an illegal value.

\textbf{NE\_CONVERGENCE} \\
The solution has not converged after \texttt{<value>} iterations.

\textbf{NE\_ENUM\_INT\_2} \\
On entry, \texttt{m} = \texttt{<value>} and \texttt{n} = \texttt{<value>}.
Constraint: \(0 < m \leq \min(n, \texttt{<value>})\).
On entry, \texttt{method} = \texttt{<value>}, \texttt{n} = \texttt{<value>} and \texttt{m} = \texttt{<value>}.
Constraint: if \texttt{method} = \texttt{Nag\_SparseNsym\_BiCGSTAB}, \(0 < m \leq \min(n,10)\).
On entry, \texttt{method} = \texttt{<value>}, \texttt{n} = \texttt{<value>} and \texttt{m} = \texttt{<value>}. 
Constraint: if \texttt{method} = \texttt{Nag\_SparseNsym\_RGMRES}, \(0 < m \leq \min(n,50)\).

\textbf{NE\_INT} \\
On entry, \texttt{maxitn} = \texttt{<value>}.
Constraint: \texttt{maxitn} \(\geq 1\)
On entry, \texttt{n} = \texttt{<value>}.
Constraint: \texttt{n} \(\geq 1\).
On entry, \texttt{nnz} = \texttt{<value>}.
Constraint: \texttt{nnz} \(\geq 1\).

\textbf{NE\_INT\_2} \\
On entry, \texttt{nnz} = \texttt{<value>} and \texttt{n} = \texttt{<value>}. 
Constraint: \(1 \leq \texttt{nnz} \leq n^2\).

\textbf{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.
An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

**NE_INVALID_CS**

On entry, \( i = \langle \text{value} \rangle \), \( \text{icol}[i-1] = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: \( \text{icol}[i-1] \geq 1 \) and \( \text{icol}[i-1] \leq n \).

On entry, \( i = \langle \text{value} \rangle \), \( \text{irow}[i-1] = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).
Constraint: \( \text{irow}[i-1] \geq 1 \) and \( \text{irow}[i-1] \leq n \).

**NE_NO_LICENCE**

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

**NE_NOT_STRICTLY_INCREASING**

On entry, \( a[i-1] \) is out of order: \( i = \langle \text{value} \rangle \).

On entry, the location \( (\text{irow}[I-1], \text{icol}[I-1]) \) is a duplicate: \( I = \langle \text{value} \rangle \).

**NE_REAL**

On entry, \( \omega = \langle \text{value} \rangle \).
Constraint: \( 0 < \omega < 2.0 \)

On entry, \( \text{tol} = \langle \text{value} \rangle \).
Constraint: \( \text{tol} < 1.0 \).

**NE_ZERO_DIAG_ELEM**

The matrix \( A \) has a zero diagonal entry in row \( \langle \text{value} \rangle \).

The matrix \( A \) has no diagonal entry in row \( \langle \text{value} \rangle \).

### 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 \( \text{rnorm} \).

### 8 Parallelism and Performance

\( \text{nag_sparse_nherm_sol (f11dsc)} \) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\( \text{nag_sparse_nherm_sol (f11dsc)} \) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users’ Note for your implementation for any additional implementation-specific information.

### 9 Further Comments

The time taken by \( \text{nag_sparse_nherm_sol (f11dsc)} \) for each iteration is roughly proportional to \( \text{nnz} \).
The number of iterations required to achieve a prescribed accuracy cannot easily be determined \textit{a priori}, as it can depend dramatically on the conditioning and spectrum of the preconditioned coefficient matrix $A = M^{-1}A$, for some preconditioning matrix $M$.

## 10 Example

This example solves a complex sparse non-Hermitian system of equations using the CGS method, with no preconditioning.

### 10.1 Program Text

/* nag_sparse_nherm_sol (f11dsc) Example Program.
 * Copyright 2014 Numerical Algorithms Group.
 * Mark 23, 2011.
 */
#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
#include <nagf11.h>
int main(void)
{
    /* Scalars */
    Integer exit_status = 0;
    double omega, rnorm, tol;
    Integer i, itn, m, maxitn, n, nnz;
    /* Arrays */
    Complex *a = 0, *b = 0, *x = 0;
    Integer *icol = 0, *irow = 0;
    char nag_enum_arg[40];
    /* NAG types */
    Nag_SparseNsym_Method method;
    Nag_SparseNsym_PrecType precon;
    NagError fail;

    INIT_FAIL(fail);

    printf("nag_sparse_nherm_sol (f11dsc) Example Program Results\n\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%*[\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("%39s%*[\n]", nag_enum_arg, _countof(nag_enum_arg));
    #else
        scanf("%39s%*[\n]", nag_enum_arg);
    #endif
}
#endif
precon = (Nag_SparseNsym_PrecType) nag_enum_name_to_value(nag_enum_arg);
#endif _WIN32
scanf_s("%lf%*[\n"]", &omega);
#else
scanf("%lf%*[\n"]", &omega);
#endif
#endif _WIN32
scanf_s("%"NAG_IFMT"%lf%"NAG_IFMT"%*[\n"]", &m, &tol, &maxitn);
#else
scanf("%"NAG_IFMT"%lf%"NAG_IFMT"%*[\n"]", &m, &tol, &maxitn);
#endif
if (! (a = NAG_ALLOC((nnz), Complex)) || ! (b = NAG_ALLOC((n), Complex)) || ! (x = NAG_ALLOC((n), Complex)) || ! (icol = NAG_ALLOC((nnz), Integer)) || ! (irow = NAG_ALLOC((nnz), Integer)) ) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
/* Read the matrix A*/
for (i = 0; i < nnz; i++)
#ifdef _WIN32
    scanf_s(" ( %lf , %lf ) %"NAG_IFMT"%*[\n"]
        &a[i].re, &a[i].im, &irow[i], &icol[i]);
#else
    scanf(" ( %lf , %lf ) %"NAG_IFMT"%*[\n"]
        &a[i].re, &a[i].im, &irow[i], &icol[i]);
#endif
/* Read rhs vector b and initial approximate solution x*/
#ifdef _WIN32
for (i = 0; i < n; i++) scanf_s(" ( %lf , %lf ) ", &b[i].re, &b[i].im);
#else
for (i = 0; i < n; i++) scanf(" ( %lf , %lf ) ", &b[i].re, &b[i].im);
#endif
#ifdef _WIN32
scanf_s("%*[\n"]");
#else
scanf("%*[\n"]");
#endif
#ifdef _WIN32
for (i = 0; i < n; i++) scanf_s(" ( %lf , %lf ) ", &x[i].re, &x[i].im);
#else
for (i = 0; i < n; i++) scanf(" ( %lf , %lf ) ", &x[i].re, &x[i].im);
#endif
/* solve ax = b */
/* nag_sparse_nherm_sol (f11dsc).
 * Solution of complex sparse non-Hermitian linear system, RGMRES, CGS,
 * Bi-CGSTAB or TFQMR method, Jacobi or SSOR preconditioner Black Box.
 */
if (fail.code != NE_NOERROR) {
    printf("Error from nag_sparse_nherm_sol (f11dsc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
printf("Converged in%13"NAG_IFMT" iterations\n", itn);
printf("Final residual norm = %11.3e\n", rnorm);
#ifdef _WIN32
printf("%14s\n","Solution");
#endif
for (i = 0; i < n; i++) printf("(%13.4e, %13.4e)\n", x[i].re, x[i].im);
END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(x);
NAG_FREE(icol);
NAG_FREE(irow);
return exit_status;
}

10.2 Program Data
nag_sparse_nherm_sol (f11dsc) Example Program Data

5 : n
16 : nnz
Nag_SparseNsym_CGS : method
Nag_SparseNsym_NoPrec : precon
1.05 : omega
1 1.e-10 1000 : m, tol, maxitn
( 2., 3.) 1 1
( 1., -1.) 1 2
( -1., 0.) 1 4
( 0., 2.) 2 2
( -2., 1.) 2 3
( 1., 0.) 2 5
( 0., -1.) 3 1
( 5., 4.) 3 3
( 3., -1.) 3 4
( 1., 0.) 3 5
( -2., 2.) 4 1
( -3., 1.) 4 4
( 0., 3.) 4 5
( 4., -2.) 5 2
( -2., 0.) 5 3
( -6., 1.) 5 5 : a[i], irow[i], icol[i], i=0,...,nnz-1
( -3., 3.)
(-11., 5.)
( 23., 48.)
(-41., 2.)
(-28., -31.) : b[i], i=0,...,n-1
( 0., 0.)
( 0., 0.)
( 0., 0.)
( 0., 0.) : x[i], i=0,...,n-1

10.3 Program Results
nag_sparse_nherm_sol (f11dsc) Example Program Results

Converged in 5 iterations
Final residual norm = 1.052e-10

Solution
( 1.0000e+00, 2.0000e+00)
( 2.0000e+00, 3.0000e+00)
( 3.0000e+00, 4.0000e+00)
( 4.0000e+00, 5.0000e+00)
( 5.0000e+00, 6.0000e+00)