NAG Library Function Document

nag_sparse_nherm_jacobi (f11dxc)

1 Purpose

nag_sparse_nherm_jacobi (f11dxc) computes the approximate solution of a complex, Hermitian or non-Hermitian, sparse system of linear equations applying a number of Jacobi iterations. It is expected that nag_sparse_nherm_jacobi (f11dxc) will be used as a preconditioner for the iterative solution of complex sparse systems of equations.

2 Specification

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

void nag_sparse_nherm_jacobi (Nag_SparseNsym_Store store, 
                           Nag_TransType trans, Nag_InitializeA init, Integer niter, Integer n, 
                           Integer nnz, const Complex a[], const Integer irow[], 
                           const Integer icol[], Nag_SparseNsym_CheckData check, const Complex b[], 
                           Complex x[], Complex diag[], NagError *fail)
```

3 Description

nag_sparse_nherm_jacobi (f11dxc) computes the approximate solution of the complex sparse system of linear equations $Ax = b$ using niter iterations of the Jacobi algorithm (see also Golub and Van Loan (1996) and Young (1971)):

$$x_{k+1} = x_k + D^{-1}(b - Ax_k)$$  \hspace{1cm} (1)

where $k = 1, 2, \ldots, \text{niter}$ and $x_0 = 0$.

nag_sparse_nherm_jacobi (f11dxc) can be used both for non-Hermitian and Hermitian systems of equations. For Hermitian matrices, either all nonzero elements of the matrix $A$ can be supplied using coordinate storage (CS), or only the nonzero elements of the lower triangle of $A$, using symmetric coordinate storage (SCS) (see the f11 Chapter Introduction).

It is expected that nag_sparse_nherm_jacobi (f11dxc) will be used as a preconditioner for the iterative solution of complex sparse systems of equations. This may be with either the Hermitian or non-Hermitian suites of functions.

For Hermitian systems the suite consists of:

- nag_sparse_herm_basic_setup (f11grc),
- nag_sparse_herm_basic_solver (f11gsc),
- nag_sparse_herm_basic_diagnostic (f11gtc).

For non-Hermitian systems the suite consists of:

- nag_sparse_nherm_basic_setup (f11brc),
- nag_sparse_nherm_basic_solver (f11bsc),
- nag_sparse_nherm_basic_diagnostic (f11btc).

4 References


5 Arguments

1: \texttt{store} – \texttt{Nag\_SparseNsym\_Store} \hspace{2cm} \textit{Input}

\textit{On entry:} specifies whether the matrix \(A\) is stored using symmetric coordinate storage (SCS) (applicable only to a Hermitian matrix \(A\)) or coordinate storage (CS) (applicable to both Hermitian and non-Hermitian matrices).

\texttt{store} = \texttt{Nag\_SparseNsym\_StoreCS}

The complete matrix \(A\) is stored in CS format.

\texttt{store} = \texttt{Nag\_SparseNsym\_StoreSCS}

The lower triangle of the Hermitian matrix \(A\) is stored in SCS format.

\textit{Constraint:} \texttt{store} = \texttt{Nag\_SparseNsym\_StoreCS} or \texttt{Nag\_SparseNsym\_StoreSCS}.

2: \texttt{trans} – \texttt{Nag\_TransType} \hspace{2cm} \textit{Input}

\textit{On entry:} if \texttt{store} = \texttt{Nag\_SparseNsym\_StoreCS}, specifies whether the approximate solution of \(Ax = b\) or of \(A^Hx = b\) is required.

\texttt{trans} = \texttt{Nag\_NoTrans}

The approximate solution of \(Ax = b\) is calculated.

\texttt{trans} = \texttt{Nag\_Trans}

The approximate solution of \(A^Hx = b\) is calculated.

\textit{Suggested value:} if the matrix \(A\) is Hermitian and stored in CS format, it is recommended that \texttt{trans} = \texttt{Nag\_NoTrans} for reasons of efficiency.

\textit{Constraint:} \texttt{trans} = \texttt{Nag\_NoTrans} or \texttt{Nag\_Trans}.

3: \texttt{init} – \texttt{Nag\_InitializeA} \hspace{2cm} \textit{Input}

\textit{On entry:} on first entry, \texttt{init} should be set to \texttt{Nag\_Initialize1}, unless the diagonal elements of \(A\) are already stored in the array \texttt{diag}. If \texttt{diag} already contains the diagonal of \(A\), it must be set to \texttt{Nag\_InputA}.

\texttt{init} = \texttt{Nag\_InputA}

\texttt{diag} must contain the diagonal of \(A\).

\texttt{init} = \texttt{Nag\_Initialize1}

\texttt{diag} will store the diagonal of \(A\) on exit.

\textit{Suggested value:} \texttt{init} = \texttt{Nag\_Initialize1} on first entry; \texttt{init} = \texttt{Nag\_InputA}, subsequently, unless \texttt{diag} has been overwritten.

\textit{Constraint:} \texttt{init} = \texttt{Nag\_InputA} or \texttt{Nag\_Initialize1}.

4: \texttt{niter} – Integer \hspace{2cm} \textit{Input}

\textit{On entry:} the number of Jacobi iterations requested.

\textit{Constraint:} \texttt{niter} \(\geq 1\).

5: \texttt{n} – Integer \hspace{2cm} \textit{Input}

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

\textit{Constraint:} \texttt{n} \(\geq 1\).

6: \texttt{nnz} – Integer \hspace{2cm} \textit{Input}

\textit{On entry:} if \texttt{store} = \texttt{Nag\_SparseNsym\_StoreCS}, the number of nonzero elements in the matrix \(A\). If \texttt{store} = \texttt{Nag\_SparseNsym\_StoreSCS}, the number of nonzero elements in the lower triangle of the matrix \(A\).
Constraints:

\[
\begin{align*}
\text{if } & \text{ store } = \text{Nag\_Sparse\_Nsym\_StoreCS}, 1 \leq \text{nnz} \leq n^2; \\
\text{if } & \text{ store } = \text{Nag\_Sparse\_Nsym\_StoreSCS}, 1 \leq \text{nnz} \leq n \times (n + 1)/2.
\end{align*}
\]

7: \( a[\text{nnz}] \) – const Complex

*Input*

*On entry:* if \( \text{store} = \text{Nag\_Sparse\_Nsym\_StoreCS} \), the nonzero elements in the matrix \( A \) (CS format).

If \( \text{store} = \text{Nag\_Sparse\_Nsym\_StoreSCS} \), the nonzero elements in the lower triangle of the matrix \( A \) (SCS format).

In both cases, the elements of either \( A \) or of its lower triangle must be 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 \( \text{nag\_sparse\_nherm\_sort (f11znc)} \) or \( \text{nag\_sparse\_herm\_sort (f11zpc)} \) may be used to reorder the elements in this way for CS and SCS storage, respectively.

8: \( \text{irow[\text{nnz}]} \) – const Integer

*Input*

*On entry:* if \( \text{store} = \text{Nag\_Sparse\_Nsym\_StoreCS} \), the row and column indices of the nonzero elements supplied in \( a \).

If \( \text{store} = \text{Nag\_Sparse\_Nsym\_StoreSCS} \), the row and column indices of the nonzero elements of the lower triangle of the matrix \( A \) supplied in \( a \).

Constraints:

\[
1 \leq \text{irow}[i] \leq n, \text{ for } i = 0, 1, \ldots, \text{nnz} - 1;
\]

\[
\text{if } \text{store} = \text{Nag\_Sparse\_Nsym\_StoreCS}, 1 \leq \text{icol}[i] \leq n, \text{ for } i = 0, 1, \ldots, \text{nnz} - 1;
\]

\[
\text{if } \text{store} = \text{Nag\_Sparse\_Nsym\_StoreSCS}, 1 \leq \text{icol}[i] \leq \text{irow}[i], \text{ for } i = 0, 1, \ldots, \text{nnz} - 1;
\]

either \( \text{irow}[i - 1] < \text{irow}[i] \) or both \( \text{irow}[i - 1] = \text{irow}[i] \) and \( \text{icol}[i - 1] < \text{icol}[i] \), for \( i = 1, 2, \ldots, \text{nnz} - 1 \).

9: \( \text{icol[\text{nnz}]} \) – const Integer

*Input*

*On entry:* if \( \text{store} = \text{Nag\_Sparse\_Nsym\_StoreCS} \), the row and column indices of the nonzero elements supplied in \( a \).

If \( \text{store} = \text{Nag\_Sparse\_Nsym\_StoreSCS} \), the row and column indices of the nonzero elements of the lower triangle of the matrix \( A \) supplied in \( a \).

10: \( \text{check} \) – Nag\_Sparse\_Nsym\_CheckData

*Input*

*On entry:* specifies whether or not the CS or SCS representation of the matrix \( A \) should be checked.

\( \text{check} = \text{Nag\_Sparse\_Nsym\_Check} \)

Checks are carried out on the values of \( n, \text{nnz}, \text{irow}, \text{icol} \); if \( \text{init} = \text{Nag\_InputA} \), \( \text{diag} \) is also checked.

\( \text{check} = \text{Nag\_Sparse\_Nsym\_NoCheck} \)

None of these checks are carried out.

See also Section 9.2.

Constraint: \( \text{check} = \text{Nag\_Sparse\_Nsym\_Check} \) or \( \text{Nag\_Sparse\_Nsym\_NoCheck} \).

11: \( b[\text{n}] \) – const Complex

*Input*

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

12: \( x[\text{n}] \) – Complex

*Output*

*On exit:* the approximate solution vector \( x_{\text{iter}} \).

13: \( \text{diag[\text{n}]} \) – Complex

*Input/Output*

*On entry:* if \( \text{init} = \text{Nag\_InputA} \), the diagonal elements of \( A \).

*On exit:* if \( \text{init} = \text{Nag\_InputA} \), unchanged on exit.

If \( \text{init} = \text{Nag\_InitializeI} \), the diagonal elements of \( A \).
6 Error Indicators and Warnings

NE_ALLOC_FAIL
Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM
On entry, argument \(value\) had an illegal value.

NE_INT
On entry, \(n = \langle value\rangle\).
Constraint: \(n \geq 1\).
On entry, \(niter = \langle value\rangle\).
Constraint: \(niter \geq 1\).
On entry, \(nnz = \langle value\rangle\).
Constraint: \(nnz \geq 1\).

NE_INT_2
On entry, \(nnz = \langle value\rangle\) and \(n = \langle value\rangle\).
Constraint: \(nnz \leq n \times (n + 1)/2\)
On entry, \(nnz = \langle value\rangle\) and \(n = \langle value\rangle\).
Constraint: \(nnz \leq n^2\)

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

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: \(i = \langle value\rangle\).
On entry, the location \((irow[I - 1], icol[I - 1])\) is a duplicate: \(I = \langle value\rangle\).
On entry, the diagonal element of the $I$th row is zero or missing: $I = \langle\text{value}\rangle$.

On entry, the element $\text{diag}[I - 1]$ is zero: $I = \langle\text{value}\rangle$.

7 Accuracy

In general, the Jacobi method cannot be used on its own to solve systems of linear equations. The rate of convergence is bound by its spectral properties (see, for example, Golub and Van Loan (1996)) and as a solver, the Jacobi method can only be applied to a limited set of matrices. One condition that guarantees convergence is strict diagonal dominance.

However, the Jacobi method can be used successfully as a preconditioner to a wider class of systems of equations. The Jacobi method has good vector/parallel properties, hence it can be applied very efficiently. Unfortunately, it is not possible to provide criteria which define the applicability of the Jacobi method as a preconditioner, and its usefulness must be judged for each case.

8 Parallelism and Performance

nag_sparse_nherm_jacobi (f11dxc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_sparse_nherm_jacobi (f11dxc) 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

9.1 Timing

The time taken for a call to nag_sparse_nherm_jacobi (f11dxc) is proportional to $\text{niter} \times \text{nnz}$.

9.2 Use of check

It is expected that a common use of nag_sparse_nherm_jacobi (f11dxc) will be as preconditioner for the iterative solution of complex, Hermitian or non-Hermitian, linear systems. In this situation, nag_sparse_nherm_jacobi (f11dxc) is likely to be called many times. In the interests of both reliability and efficiency, you are recommended to set $\text{check} = \text{Nag_SparseNsym_Check}$ for the first of such calls, and to set $\text{check} = \text{Nag_SparseNsym_NoCheck}$ for all subsequent calls.

10 Example

This example solves the complex sparse non-Hermitian system of equations $Ax = b$ iteratively using nag_sparse_nherm_jacobi (f11dxc) as a preconditioner.

10.1 Program Text

/* nag_sparse_nherm_jacobi (f11dxc) 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 anorm, sigmax, stplhs, stprhs, tol;
    Integer i, irevcm, iterm, itn, lwork, lwreq, m, maxitn,
              monit, n, niter, nnz;
/* Arrays */
    char nag_enum_arg[100];
    Complex *a = 0, *b = 0, *diag = 0, *work = 0, *x = 0;
    double *wgt = 0;
    Integer *icol = 0, *irow = 0;
/* NAG types */
    Nag_InitializeA init;
    Nag_SparseNsym_Method method;
    Nag_SparseNsym_PrecType precon;
    Nag_NormType norm;
    Nag_SparseNsym_Weight weight;
    NagError fail, fail1;
INIT_FAIL(fail);
INIT_FAIL(fail1);
printf("nag_sparse_nherm_jacobi (f11dxc) 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
lwork = 300;
if (! (a = NAG_ALLOC(nnz, Complex)) ||
    !(b = NAG_ALLOC(n, Complex)) ||
    !(diag = NAG_ALLOC(n, Complex)) ||
    !(work = NAG_ALLOC(lwork, Complex)) ||
    !(x = NAG_ALLOC(n, Complex)) ||
    !(wgt = NAG_ALLOC(n, double)) ||
    !(icol = NAG_ALLOC(nnz, Integer)) ||
    !(irow = NAG_ALLOC(nnz, Integer)) ) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
/* Read or initialize the parameters for the iterative solver*/
#ifdef _WIN32
    scanf_s("%99s%*[\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%99s%*[\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("%99s%*[\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%99s%*[\n]", nag_enum_arg);
#endif
    /* nag_enum_name_to_value (x04nac).
     * Converts NAG enum member name to value
     */
    precon = (Nag_SparseNsym_PrecType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
    scanf_s("%99s%*[\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%99s%*[\n]", nag_enum_arg);
#endif
    /* nag_enum_name_to_value (x04nac).
     * Converts NAG enum member name to value
     */
    norm = (Nag_NormType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
    scanf_s("%99s%*[\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%99s%*[\n]", nag_enum_arg);
#endif
    /* nag_enum_name_to_value (x04nac).
     * Converts NAG enum member name to value
     */
    weight = (Nag_SparseNsym_Weight) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
    scanf_s("%99s%*[\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%99s%*[\n]", nag_enum_arg);
#endif
    fail = fail1 = NagNoError;
  // ...
```c
scanf_s("%99s\[\n\", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%99s\[\n\", nag_enum_arg);
#endif
norm = (Nag_NormType) nag_enum_name_to_value(nag_enum_arg);
#endif
    scanf("%99s\[\n\", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%99s\[\n\", nag_enum_arg);
#endif
weight = (Nag_SparseNsym_Weight) nag_enum_name_to_value(nag_enum_arg);
#endif
    scanf_s("%"NAG_IFMT "%lf"NAG_IFMT "%[\n\", &iterm);
#else
    scanf("%"NAG_IFMT "%lf"NAG_IFMT "%[\n\", &iterm);
#endif
    scanf_s("%"NAG_IFMT "%lf"NAG_IFMT "%[\n\", &m, &tol, &maxitn);
#else
    scanf("%"NAG_IFMT "%lf"NAG_IFMT "%[\n\", &m, &tol, &maxitn);
#endif
    scanf("%"NAG_IFMT "%[\n\", &monit);
#else
    scanf("%"NAG_IFMT "%[\n\", &monit);
#endif
/* Read the parameters for the preconditioner*/
#endif
    scanf_s("%"NAG_IFMT "%[\n\", &niter);
#else
    scanf("%"NAG_IFMT "%[\n\", &niter);
#endif
anorm = 0.0;
sigmax = 0.0;
/* Read the non-zero elements of the matrix A*/
for (i = 0; i < nnz; i++)
#endif
    scanf_s(" ( %lf , %lf ) %"NAG_IFMT "%NAG_IFMT "%[\n\", &a[i].re, &a[i].im, &irow[i], &icol[i]);
#else
    scanf(" ( %lf , %lf ) %"NAG_IFMT "%NAG_IFMT "%[\n\", &a[i].re, &a[i].im, &irow[i], &icol[i]);
#endif
/* Read right-hand side vector b and initial approximate solution*/
#endif
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
#endif
    scanf_s("%*[\n\") ;
#else
    scanf("%*[\n\") ;
#endif
/* Call to initialize the solver */
#endif
nag_sparse_nherm_basic_setup(method, precon, norm, weight, iterm, n, m, tol,
```
maxitn, anorm, sigmax, monit, &lwreq, work, lwork, &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_sparse_nherm_basic_setup (f11brc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Call solver repeatedly to solve the equations.
     * Note: the arrays b and x are overwritten; on final exit, x will
     * contain the solution and b the residual vector.
     */
    irevcm = 0;
    init = Nag_InitializeI;
    while (irevcm != 4) {
        /* nag_sparse_nherm_basic_solver (f11bsc)
         * Complex sparse non-Hermitian linear systems, preconditioned RGMRES, CGS,
         * Bi-CGSTAB or TPQMR method
         */
        nag_sparse_nherm_basic_solver(&irevcm, x, b, wgt, work, lwreq, &fail);
        switch (irevcm) {
            case -1:
                /* nag_sparse_nherm_matvec (f11xnc)
                 * Complex sparse non-Hermitian matrix vector multiply
                 */
                nag_sparse_nherm_matvec(Nag_ConjTrans, n, nnz, a, irow, icol,
                                   Nag_SparseNsym_NoCheck, x, b, &fail1);
                break;
            case 1:
                nag_sparse_nherm_matvec(Nag_NoTrans, n, nnz, a, irow, icol,
                                         Nag_SparseNsym_NoCheck, x, b, &fail1);
                break;
            case 2:
                /* nag_sparse_nherm_jacobi (f11dxc).
                 * Complex sparse nonsymmetric linear systems, line Jacobi preconditioner
                 */
                nag_sparse_nherm_jacobi(Nag_SparseNsym_StoreCS, Nag_NoTrans, init,
                                          niter, n, nnz, a, irow, icol,
                                          Nag_SparseNsym_Check, x, b, diag, &fail1);
                init = Nag_InputA;
                break;
            case 3:
                /* nag_sparse_nherm_basic_diagnostic (f11btc)
                 * Complex sparse non-Hermitian linear systems, diagnostic
                 */
                nag_sparse_nherm_basic_diagnostic(&itn, &stplhs, &stprhs, &anorm,
                                                   &sigmax, work, lwreq, &fail1);
                if (fail1.code == NE_NOERROR) {
                    printf("Monitoring at iteration no.%4"NAG_IFMT" residual %14.4e\n",
                            itn, stplhs);
                }
                if (fail1.code != NE_NOERROR) irevcm = 6;
            }
        }
        if (fail.code != NE_NOERROR) {
            printf("Error from nag_sparse_nherm_basic_solver (f11bsc)\n%s\n", fail.message);
            exit_status = 2;
            goto END;
        }
        /* Obtain information about the computation using
         * nag_sparse_nherm_basic_diagnostic (f11btc).
         * Complex sparse Hermitian linear systems, diagnostic.
         */
        nag_sparse_nherm_basic_diagnostic(&itn, &stplhs, &stprhs, &anorm, &sigmax,
                                           work, lwreq, &fail1);
        if (fail.code != NE_NOERROR) {
            printf("Error from nag_sparse_nherm_basic_diagnostic (f11btc)\n%s\n", fail.message);
            exit_status = 3;
        }
    }
10.2 Program Data

nag_sparse_nherm_jacobi (f11dxc) Example Program Data

<table>
<thead>
<tr>
<th>8</th>
<th>: n</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>: nnz</td>
</tr>
<tr>
<td>Nag_SparseNsym_TFQMR</td>
<td>: method</td>
</tr>
<tr>
<td>Nag_SparseNsym_Prec</td>
<td>: precon</td>
</tr>
<tr>
<td>Nag_OneNorm</td>
<td>: norm</td>
</tr>
<tr>
<td>Nag_SparseNsym_UnWeighted</td>
<td>: weight</td>
</tr>
<tr>
<td>1</td>
<td>: iterm</td>
</tr>
<tr>
<td>2</td>
<td>: monit</td>
</tr>
<tr>
<td>1.0E-6</td>
<td>: tol, maxitn</td>
</tr>
<tr>
<td>20</td>
<td>: niter</td>
</tr>
<tr>
<td>( 2., 1.)</td>
<td>1 1</td>
</tr>
<tr>
<td>( -1., 1.)</td>
<td>1 4</td>
</tr>
<tr>
<td>( 1., -3.)</td>
<td>1 8</td>
</tr>
<tr>
<td>( 4., 7.)</td>
<td>2 1</td>
</tr>
<tr>
<td>( -3., 0.)</td>
<td>2 2</td>
</tr>
<tr>
<td>( 2., 4.)</td>
<td>2 5</td>
</tr>
<tr>
<td>( -7., -5.)</td>
<td>3 3</td>
</tr>
<tr>
<td>( 2., 1.)</td>
<td>3 6</td>
</tr>
<tr>
<td>( 3., 2.)</td>
<td>4 1</td>
</tr>
<tr>
<td>( -4., 2.)</td>
<td>4 3</td>
</tr>
<tr>
<td>( 0., 1.)</td>
<td>4 4</td>
</tr>
<tr>
<td>( 5., -3.)</td>
<td>4 7</td>
</tr>
<tr>
<td>( -1., 2.)</td>
<td>5 2</td>
</tr>
<tr>
<td>( 8., 6.)</td>
<td>5 5</td>
</tr>
<tr>
<td>( -3., -4.)</td>
<td>5 7</td>
</tr>
<tr>
<td>( -6., -2.)</td>
<td>6 1</td>
</tr>
<tr>
<td>( 5., -2.)</td>
<td>6 3</td>
</tr>
<tr>
<td>( 2., 0.)</td>
<td>6 6</td>
</tr>
<tr>
<td>( 0., -5.)</td>
<td>7 3</td>
</tr>
<tr>
<td>( -1., 5.)</td>
<td>7 5</td>
</tr>
<tr>
<td>( 6., 2.)</td>
<td>7 7</td>
</tr>
<tr>
<td>( -1., 4.)</td>
<td>8 2</td>
</tr>
<tr>
<td>( 2., 0.)</td>
<td>8 6</td>
</tr>
<tr>
<td>( 3., 3.)</td>
<td>8 8</td>
</tr>
<tr>
<td>7.</td>
<td>: a[i], irow[i], icol[i], i=0,...,nnz-1</td>
</tr>
<tr>
<td>11.</td>
<td></td>
</tr>
<tr>
<td>( 1., 24.)</td>
<td></td>
</tr>
<tr>
<td>( -13., -18.)</td>
<td></td>
</tr>
<tr>
<td>( -10., 3.)</td>
<td></td>
</tr>
<tr>
<td>( 23., 14.)</td>
<td></td>
</tr>
<tr>
<td>( 17., -7.)</td>
<td></td>
</tr>
<tr>
<td>( 15., -3.)</td>
<td></td>
</tr>
<tr>
<td>( -3., 20.)</td>
<td></td>
</tr>
</tbody>
</table>

f11 – Large Scale Linear Systems

f11dxc.9
10.3 Program Results

nag_sparse_nherm_jacobi (f11dxc) Example Program Results

Monitoring at iteration no. 1 residual 1.5062e+02
Monitoring at iteration no. 2 residual 1.5704e+02
Monitoring at iteration no. 3 residual 1.4803e+02
Monitoring at iteration no. 4 residual 8.5215e+01
Monitoring at iteration no. 5 residual 4.2951e+01
Monitoring at iteration no. 6 residual 2.5055e+01
Monitoring at iteration no. 7 residual 1.9090e-01

Final Results
Number of iterations for convergence: 8
Residual norm: 9.5485e-08
Right-hand side of termination criterion: 8.9100e-04
1-norm of matrix A: 2.7000e+01

<table>
<thead>
<tr>
<th>Solution vector</th>
<th>Residual vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1.0000e+00, 1.0000e+00)</td>
<td>( 4.7145e-09, -3.9432e-09)</td>
</tr>
<tr>
<td>( 2.0000e+00, -1.0000e+00)</td>
<td>( 8.7691e-09, -9.8075e-09)</td>
</tr>
<tr>
<td>( 3.0000e+00, 1.0000e+00)</td>
<td>( -2.8739e-09, 4.1582e-09)</td>
</tr>
<tr>
<td>( 4.0000e+00, -1.0000e+00)</td>
<td>( 3.5787e-09, -1.1121e-08)</td>
</tr>
<tr>
<td>( 3.0000e+00, -1.0000e+00)</td>
<td>( -6.9210e-09, -8.6888e-09)</td>
</tr>
<tr>
<td>( 2.0000e+00, 1.0000e+00)</td>
<td>( -2.2468e-09, 8.4933e-09)</td>
</tr>
<tr>
<td>( 1.0000e+00, -1.0000e+00)</td>
<td>( 5.2201e-10, -3.3375e-09)</td>
</tr>
<tr>
<td>( 2.4713e-09, 3.0000e+00)</td>
<td>( -5.5778e-09, -1.0732e-08)</td>
</tr>
</tbody>
</table>