NAG Library Function Document

nag_sparse_nsym_jacobi (f11dkc)

1 Purpose

nag_sparse_nsym_jacobi (f11dkc) computes the approximate solution of a real, symmetric or nonsymmetric, sparse system of linear equations applying a number of Jacobi iterations. It is expected that nag_sparse_nsym_jacobi (f11dkc) will be used as a preconditioner for the iterative solution of real sparse systems of equations.

2 Specification

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

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

3 Description

nag_sparse_nsym_jacobi (f11dkc) computes the approximate solution of the real sparse system of linear equations $Ax = b$ using niter iterations of the Jacobi algorithm (see also Golub and Van Loan (1996) and Young (1971)):

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

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

nag_sparse_nsym_jacobi (f11dkc) can be used both for nonsymmetric and symmetric systems of equations. For symmetric 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_nsym_jacobi (f11dkc) will be used as a preconditioner for the iterative solution of real sparse systems of equations. This may be with either the symmetric or nonsymmetric suites of functions.

For symmetric systems the suite consists of:

- nag_sparse_sym_basic_setup (f11gdc),
- nag_sparse_sym_basic_solver (f11gec),
- nag_sparse_sym_basic_diagnostic (f11gfc).

For nonsymmetric systems the suite consists of:

- nag_sparse_nsym_basic_setup (f11bdc),
- nag_sparse_nsym_basic_solver (f11bec),
- nag_sparse_nsym_basic_diagnostic (f11bfc).

4 References


5 Arguments

1: store – Nag_SparseNsym_Store

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

   \[
   \text{store} = \begin{cases} 
   \text{Nag_SparseNsym_StoreCS} & \text{if stored in CS format.} \\
   \text{Nag_SparseNsym_StoreSCS} & \text{if stored in SCS format.} 
\end{cases}
\]

   \text{Constraint: store = Nag_SparseNsym_StoreCS or Nag_SparseNsym_StoreSCS.}

2: trans – Nag_TransType

   On entry: if \text{store} = \text{Nag_SparseNsym_StoreCS}, specifies whether the approximate solution of $Ax = b$ or of $A^T x = b$ is required.

   \[
   \text{trans} = \begin{cases} 
   \text{Nag_NoTrans} & \text{if } A x = b \text{ is calculated.} \\
   \text{Nag_Trans} & \text{if } A^T x = b \text{ is calculated.} 
\end{cases}
\]

   \text{Suggested value: if the matrix $A$ is symmetric and stored in CS format, it is recommended that \text{trans} = \text{Nag_NoTrans} for reasons of efficiency.}

   \text{Constraint: trans = Nag_NoTrans or Nag_Trans.}

3: init – Nag_InitializeA

   On entry: on first entry, \text{init} should be set to \text{Nag_Initialize}, unless the diagonal elements of $A$ are already stored in the array \text{diag}. If \text{diag} already contains the diagonal of $A$, it must be set to \text{Nag_InitializeA}.

   \[
   \text{init} = \begin{cases} 
   \text{Nag_InputA} & \text{if } diag \text{ must contain the diagonal of } A. \\
   \text{Nag_Initialize} & \text{if } diag \text{ will store the diagonal of } A \text{ on exit.} 
\end{cases}
\]

   \text{Suggested value: init = Nag_Initialize on first entry; init = Nag_InputA, subsequently, unless diag has been overwritten.}

   \text{Constraint: init = Nag_InputA or Nag_Initialize.}

4: niter – Integer

   On entry: the number of Jacobi iterations requested.

   \text{Constraint: niter \geq 1.}

5: n – Integer

   On entry: $n$, the order of the matrix $A$.

   \text{Constraint: n \geq 1.}

6: nnz – Integer

   On entry: if \text{store} = \text{Nag_SparseNsym_StoreCS}, the number of nonzero elements in the matrix $A$. If \text{store} = \text{Nag_SparseNsym_StoreSCS}, the number of nonzero elements in the lower triangle of the matrix $A$. 
Constraints:

if store = Nag_SparseNsym_StoreCS, $1 \leq \text{nnz} \leq n^2$;
if store = Nag_SparseNsym_StoreSCS, $1 \leq \text{nnz} \leq n \times (n + 1)/2$.

7: $a[\text{nnz}]$ – const double  

Input  

On entry: if store = Nag_SparseNsym_StoreCS, the nonzero elements in the matrix $A$ (CS format).
If store = Nag_SparseNsym_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
columns indices are not permitted. The function nag_sparse_nsym_sort (f11zac) or
nag_sparse_sym_sort (f11zbc) may be used to reorder the elements in this way for CS and
SCS storage, respectively.

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

Input  

On entry: if store = Nag_SparseNsym_StoreCS, the row and column indices of the nonzero
elements supplied in $a$.  
If store = Nag_SparseNsym_StoreSCS, the row and column indices of the nonzero elements of the
lower triangle of the matrix $A$ supplied in $a$.

Constraints:

$1 \leq irow[i] \leq n$, for $i = 0, 1, \ldots, \text{nnz} - 1$;
if store = Nag_SparseNsym_StoreCS, $1 \leq icol[i] \leq n$, for $i = 0, 1, \ldots, \text{nnz} - 1$;
if store = Nag_SparseNsym_StoreSCS, $1 \leq icol[i] \leq irow[i]$, for $i = 0, 1, \ldots, \text{nnz} - 1$;
either $irow[i-1] < irow[i]$ or both $irow[i] = irow[i]$ and $icol[i-1] < icol[i]$, for $i = 1, 2, \ldots, \text{nnz} - 1$.

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

Input  

On entry: if store = Nag_SparseNsym_StoreCS, the row and column indices of the nonzero
elements supplied in $a$.
If store = Nag_SparseNsym_StoreSCS, the row and column indices of the nonzero elements of the
lower triangle of the matrix $A$ supplied in $a$.

Constraints:

10: check – Nag_SparseNsym_CheckData  

Input  

On entry: specifies whether or not the CS or SCS representation of the matrix $A$ should be
checked.

check = Nag_SparseNsym_Check  

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

check = Nag_SparseNsym_NoCheck  

None of these checks are carried out.

See also Section 9.2.
Constraint: check = Nag_SparseNsym_Check or Nag_SparseNsym_NoCheck.

11: $b[n]$ – const double  

Input  

On entry: the right-hand side vector $b$.

12: $x[n]$ – double  

Output  

On exit: the approximate solution vector $x_{\text{iter}}$.

13: diag[n] – double  

Input/Output  

On entry: if init = Nag_InputA, the diagonal elements of $A$.
On exit: if init = Nag_InputA, unchanged on exit.
If init = Nag_InitializeI, the diagonal elements of $A$.  

Mark 25  

f11dkc.3
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 = <value>`.  
Constraint: `n ≥ 1`.  
On entry, `niter = <value>`.  
Constraint: `niter ≥ 1`.  
On entry, `nnz = <value>`.  
Constraint: `nnz ≥ 1`.

**NE_INT_2**
On entry, `nnz = <value>` and `n = <value>`.  
Constraint: `nnz ≤ n × (n + 1)/2`.  
On entry, `nnz = <value>` and `n = <value>`.  
Constraint: `nnz ≤ 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 = <value>`, `icol[I - 1] = <value>` and `irow[I - 1] = <value>`.  
On entry, `I = <value>`, `icol[I - 1] = <value>` and `n = <value>`.  
On entry, `I = <value>`, `irow[I - 1] = <value>` and `n = <value>`.  

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

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

### 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

$nag\_sparse\_nsym\_jacobi$ (f11dkc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

$nag\_sparse\_nsym\_jacobi$ (f11dkc) 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 $nag\_sparse\_nsym\_jacobi$ (f11dkc) is proportional to $niter \times \text{nnz}$.

#### 9.2 Use of check

It is expected that a common use of $nag\_sparse\_nsym\_jacobi$ (f11dkc) will be as preconditioner for the iterative solution of real, symmetric or nonsymmetric, linear systems. In this situation, $nag\_sparse\_nsym\_jacobi$ (f11dkc) is likely to be called many times. In the interests of both reliability and efficiency, you are recommended to set $\text{check} = \text{Nag\_Sparse\_Nsym\_Check}$ for the first of such calls, and to set $\text{check} = \text{Nag\_Sparse\_Nsym\_NoCheck}$ for all subsequent calls.

### 10 Example

This example solves the real sparse nonsymmetric system of equations $Ax = b$ iteratively using $nag\_sparse\_nsym\_jacobi$ (f11dkc) as a preconditioner.

#### 10.1 Program Text

/* $nag\_sparse\_nsym\_jacobi$ (f11dkc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 23, 2011. */
#include <nag.h>
#include <nag_stdlib.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 */
nag_enum_arg[100];
char *a = 0, *b = 0, *diag = 0, *wgt = 0, *work = 0,
*x = 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_nsym_jacobi (f11dkc) 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 = 200;
if (1)
(a = NAG_ALLOC((nnz), double)) ||
(b = NAG_ALLOC((n), double)) ||
(diag = NAG_ALLOC((n), double)) ||
(wgt = NAG_ALLOC((n), double)) ||
(work = NAG_ALLOC((lwork), double)) ||
(x = 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
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
method = (Nag_SparseNsym_Method) nag_enum_name_to_value(nag_enum_arg);
/* Read the parameters for the preconditioner */
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"%*[\n]", &iterm);
#else
    scanf("%"NAG_IFMT"%*[\n]", &iterm);
#endif
#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
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"%*[\n]", &monit);
#else
    scanf("%"NAG_IFMT"%*[\n]", &monit);
#endif
#ifdef _WIN32
    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 */
#ifdef _WIN32
    scanf_s("%lf"NAG_IFMT"%"NAG_IFMT"%*[\n]", &a[i], &irow[i], &icol[i]);
#else
    scanf("%lf"NAG_IFMT"%"NAG_IFMT"%*[\n]", &a[i], &irow[i], &icol[i]);
#endif
/* Read right-hand side vector b and initial approximate solution */
#ifdef _WIN32
    for (i = 0; i <= n - 1; i++) scanf_s("%lf", &b[i]);
#else
    for (i = 0; i <= n - 1; i++) scanf("%lf", &b[i]);
#endif
#ifdef _WIN32
    scanf_s("%*[\n]");
#else
    scanf("%*[\n]");
#endif
#ifdef _WIN32
    for (i = 0; i <= n - 1; i++) scanf_s("%lf", &x[i]);
#else
    for (i = 0; i <= n - 1; i++) scanf("%lf", &x[i]);
#endif
/* nag_sparse_nsym_basic_setup (f11bdc) */
* Real sparse nonsymmetric linear systems, setup routine */
*nag_sparse_nsym_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_nsym_basic_setup (f11bdc).\n%s\n", fail.message);
}
exit_status = 1;
go to END;
}
/* call solver repeatedly to solve the equations
 * Note that the arrays b and x are overwritten
 * On final exit, x will contain the solution and b the residual vector */
irevcm = 0;
lwreq = lwork;
init = Nag_InitializeI;
while (irevcm != 4) { /* nag_sparse_nsym_basic_solver (f11bec)
   * Real sparse nonsymmetric linear systems, solver routine
   * preconditioned RGMRES, CGS, Bi-CGSTAB or TFQMR method */
   nag_sparse_nsym_basic_solver(&irevcm, x, b, wgt, work, lwreq, &fail);
   switch (irevcm) {
   case -1: /* nag_sparse_nsym_matvec (f11xac)
             * Real sparse nonsymmetric matrix vector multiply */
             nag_sparse_nsym_matvec(Nag_Trans, n, nnz, a, irow, icol,
                                     Nag_SparseNsym_NoCheck, x, b, &fail1);
             break;
   case 1: nag_sparse_nsym_matvec(Nag_NoTrans, n, nnz, a, irow, icol,
                                         Nag_SparseNsym_NoCheck, x, b, &fail1);
             break;
   case 2: /* nag_sparse_nsym_jacobi (f11dkc)
            * Real sparse nonsymmetric linear systems, line Jacobi preconditioner */
            nag_sparse_nsym_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_nsym_basic_diagnostic (f11bfc)
            * Real sparse nonsymmetric linear systems, diagnostic for f11bec */
            nag_sparse_nsym_basic_diagnostic(&itn, &stplhs, &stprhs, &anorm, &sigmax,
                                             work, lwreq, &fail1);
            printf("%"NAG_IFMT" %f
", itn, stplhs);
   } if (fail1.code != NE_NOERROR) irevcm = 6;
} if (fail.code != NE_NOERROR) {
   printf("Error from nag_sparse_nsym_basic_solver (f11bec)\n%s\n", fail.message);
   exit_status = 2;
go to END;
}
/* Obtain information about the computation*/
   nag_sparse_nsym_basic_diagnostic(&itn, &stplhs, &stprhs, &anorm, &sigmax,
                                     work, lwreq, &fail);
/* Print the output data*/
   printf("\nFinal Results\n");
   printf("Number of iterations for convergence: %5"NAG_IFMT" \n", itn);
   printf("Residual norm: %14.4e\n", stplhs);
   printf("Right-hand side of termination criterion: %14.4e\n", stprhs);
   printf("1-norm of matrix A: %14.4e\n", anorm);
/* Output x*/
   printf("%16s%16s\n","Solution","Residuals");
   for (i = 0; i < n; i++) printf("%16.4f%16.4e\n", x[i], b[i]);
END:
NAG_FREE(a);
10.2 Program Data

nag_sparse_nsym_jacobi (f11dkc) Example Program Data

8 : n
24 : nnz
Nag_SparseNsym_BiCGSTAB : method
Nag_SparseNsym_Prec : precon
Nag_OneNorm : norm
Nag_SparseNsym_UnWeighted : weight
1 : iterm
2 1.0e-6 20 : m, tol, maxitn
1 : monit
4 : niter
4.0 1 1
-1.0 1 4
1.0 1 8
4.0 2 1
-5.0 2 2
2.0 2 5
-7.0 3 3
2.0 3 6
2.0 4 1
-1.0 4 3
6.0 4 4
2.0 4 7
-1.0 5 2
8.0 5 5
-2.0 5 7
-2.0 6 1
5.0 6 3
8.0 6 6
-2.0 7 3
-1.0 7 5
7.0 7 7
-1.0 8 2
2.0 8 6
6.0 8 8 : a[i], irow[i], icol[i], i=0,...,nnz-1
6.0 8.0 -9.0 46.0
17.0 21.0 22.0 34.0 : b[i], i=0,...,n-1
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 : x[i], i=0,...,n-1

10.3 Program Results

nag_sparse_nsym_jacobi (f11dkc) Example Program Results

Final Results
Number of iterations for convergence: 2
Residual norm: 1.1177e-04
Right-hand side of termination criterion: 5.4082e-04
1-norm of matrix A: 1.5000e+01

<table>
<thead>
<tr>
<th>Solution</th>
<th>Residuals</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.7035</td>
<td>3.2377e-07</td>
</tr>
<tr>
<td>1.0805</td>
<td>-1.7625e-05</td>
</tr>
<tr>
<td>1.8305</td>
<td>2.7964e-05</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>-------</td>
</tr>
<tr>
<td>6.0251</td>
<td>-2.5914e-05</td>
</tr>
<tr>
<td>3.2942</td>
<td>7.8156e-06</td>
</tr>
<tr>
<td>1.9068</td>
<td>9.2064e-06</td>
</tr>
<tr>
<td>4.1365</td>
<td>-3.0848e-06</td>
</tr>
<tr>
<td>5.2111</td>
<td>1.9834e-05</td>
</tr>
</tbody>
</table>