1 Purpose

nag_sparse_nsym_precon_ssor_solve (f11ddc) solves a system of linear equations involving the preconditioning matrix corresponding to SSOR applied to a real sparse nonsymmetric matrix, represented in coordinate storage format.

2 Specification

```c
#include <nag.h>
#include <nagf11.h>
void nag_sparse_nsym_precon_ssor_solve (Nag_TransType trans, Integer n,
   Integer nnz, const double a[], const Integer irow[],
   const Integer icol[], const double rdiag[], double omega,
   Nag_SparseNsym_CheckData check, const double y[], double x[],
   NagError *fail)
```

3 Description

nag_sparse_nsym_precon_ssor_solve (f11ddc) solves a system of linear equations

\[ Mx = y, \quad \text{or} \quad M^T x = y, \]

according to the value of the argument `trans`, where the matrix

\[ M = \frac{1}{\omega(2 - \omega)}(D + \omega L)D^{-1}(D + \omega U) \]

corresponds to symmetric successive-over-relaxation (SSOR) (see Young (1971)) applied to a linear system \( Ax = b \), where \( A \) is a real sparse nonsymmetric matrix stored in coordinate storage (CS) format (see Section 2.1.1 in the f11 Chapter Introduction).

In the definition of \( M \) given above \( D \) is the diagonal part of \( A \), \( L \) is the strictly lower triangular part of \( A \), \( U \) is the strictly upper triangular part of \( A \), and \( \omega \) is a user-defined relaxation parameter.

It is envisaged that a common use of nag_sparse_nsym_precon_ssor_solve (f11ddc) will be to carry out the preconditioning step required in the application of nag_sparse_nsym_basic_solver (f11bec) to sparse linear systems. For an illustration of this use of nag_sparse_nsym_precon_ssor_solve (f11ddc) see the example program given in Section 10. nag_sparse_nsym_precon_ssor_solve (f11ddc) is also used for this purpose by the Black Box function nag_sparse_nsym_sol (f11dec).

4 References


5 Arguments

1:  
   `trans` – Nag_TransType
   
   *Input*
   
   *On entry:* specifies whether or not the matrix \( M \) is transposed.

   `trans = Nag_NoTrans`

   \( Mx = y \) is solved.
trans = Nag_Trans

$M^T x = y$ is solved.

Constraint: trans = Nag_NoTrans or Nag_Trans.

2: 
$n$ – Integer

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

Constraint: $n \geq 1$.

3: 
nz – Integer

On entry: the number of nonzero elements in the matrix $A$.

Constraint: $1 \leq \text{nz} \leq n^2$.

4: 
a[nz] – const double

On entry: the nonzero elements in the matrix $A$, 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_nsym_sort (f11zac) may be used to order the elements in this way.

5: 
irow[nz] – const Integer

6: 
icol[nz] – const Integer

On entry: the row and column indices of the nonzero elements supplied in array a.

Constraints:

irow and icol must satisfy the following constraints (which may be imposed by a call to nag_sparse_nsym_sort (f11zac)):

$1 \leq \text{irow}[i] \leq n$ and $1 \leq \text{icol}[i] \leq n$, for $i = 0, 1, \ldots, \text{nz} - 1$;

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 $i = 1, 2, \ldots, \text{nz} - 1$.

7: 
rdiag[n] – const double

On entry: the elements of the diagonal matrix $D^{-1}$, where $D$ is the diagonal part of $A$.

8: 
omega – double

On entry: the relaxation parameter $\omega$.

Constraint: $0.0 < \omega < 2.0$.

9: 
check – Nag_SparseNsym_CheckData

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

check = Nag_SparseNsym_Check

Checks are carried on the values of $n$, nz, irow, icol and omega.

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.

10: 
y[n] – const double

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

f11ddc.2 Mark 25
11: \( x[n] \) – double

On exit: the solution vector \( x \).

12: fail – NagError

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

\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 <value> had an illegal value.

\textbf{NE_INT}

On entry, \( n = \langle \text{value} \rangle \).

Constraint: \( n \geq 1 \).

On entry, \( nnz = \langle \text{value} \rangle \).

Constraint: \( 1 \leq nnz \leq n^2 \).

\textbf{NE_INT_2}

On entry, \( nnz = \langle \text{value} \rangle \) and \( n = \langle \text{value} \rangle \).

Constraint: \( 1 \leq nnz \leq n^2 \).

\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.

\textbf{NE_INVALID_CS}

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

Constraint: \( icol[i-1] \geq 1 \) and \( icol[i-1] \leq n \).

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

Constraint: \( irow[i-1] \geq 1 \) and \( irow[i-1] \leq n \).

\textbf{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.

\textbf{NE_NOT_STRICTLY_INCREASING}

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

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

\textbf{NE_REAL}

On entry, \( \omega = \langle \text{value} \rangle \).

Constraint: \( 0.0 < \omega < 2.0 \).
The matrix $A$ has no diagonal entry in row $\langle value\rangle$.

### 7 Accuracy

If $\text{trans} = \text{Nag_NoTrans}$ the computed solution $x$ is the exact solution of a perturbed system of equations $(M + \delta M)x = y$, where

$$|\delta M| \leq c(n)\epsilon |D + \omega L||D^{-1}||D + \omega U|,$$

$c(n)$ is a modest linear function of $n$, and $\epsilon$ is the machine precision. An equivalent result holds when $\text{trans} = \text{Nag_Trans}$.

### 8 Parallelism and Performance

Not applicable.

### 9 Further Comments

#### 9.1 Timing

The time taken for a call to nag_sparse_nsym_precon_ssor_solve (f11ddc) is proportional to $\text{nnz}$.

#### 9.2 Use of check

It is expected that a common use of nag_sparse_nsym_precon_ssor_solve (f11ddc) will be to carry out the preconditioning step required in the application of nag_sparse_nsym_basic_solver (f11bec) to sparse linear systems. In this situation nag_sparse_nsym_precon_ssor_solve (f11ddc) is likely to be called many times with the same matrix $M$. 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 for all subsequent calls set $\text{check} = \text{Nag_SparseNsym_NoCheck}$.

### 10 Example

This example solves a sparse linear system of equations:

$$Ax = b,$$

using RGMRES with SSOR preconditioning.

The RGMRES algorithm itself is implemented by the reverse communication function nag_sparse_nsym_basic_solver (f11bec), which returns repeatedly to the calling program with various values of the argument irevcm. This argument indicates the action to be taken by the calling program.

- If $\text{irevcm} = 1$, a matrix-vector product $v = Au$ is required. This is implemented by a call to nag_sparse_nsym_matvec (f11xac).
- If $\text{irevcm} = -1$, a transposed matrix-vector product $v = A^Tu$ is required in the estimation of the norm of $A$. This is implemented by a call to nag_sparse_nsym_matvec (f11xac).
- If $\text{irevcm} = 2$, a solution of the preconditioning equation $Mv = u$ is required. This is achieved by a call to nag_sparse_nsym_precon_ssor_solve (f11ddc).
- If $\text{irevcm} = 4$, nag_sparse_nsym_basic_solver (f11bec) has completed its tasks. Either the iteration has terminated, or an error condition has arisen.

For further details see the function document for nag_sparse_nsym_basic_solver (f11bec).
10.1 Program Text

/* nag_sparse_nsym_precon_ssor_solve (f11ddc) 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, omega, sigmax, stplhs, stprhs, tol;
    Integer i, irevcm, iterm, itn, la, liwork,
            lwneed, lwork, m, maxitn, monit, n, nnz;
    /* Arrays */
    char nag_enum_arg[100];
    double *a = 0, *b = 0, *rdiag = 0, *wgt = 0,
            *work = 0, *x = 0;
    Integer *icol = 0, *irow = 0, *iwork = 0;
    /* NAG types */
    Nag_SparseNsym_CheckData ckdd, ckxa;
    Nag_NormType norm;
    Nag_SparseNsym_PrecType precon;
    Nag_TransType trans;
    Nag_SparseNsym_Weight weight;
    Nag_SparseNsym_Method method;
    NagError fail, fail1;
    INIT_FAIL(fail);
    INIT_FAIL(fail1);

    printf("nag_sparse_nsym_precon_ssor_solve (f11ddc) Example Program Results\n");
    printf("\n");
    /* Skip heading in data file*/
    #ifdef _WIN32
        scanf_s("%*[\n]");
    #else
        scanf("%*[\n]");
    #endif
    /* Read algorithmic parameters*/
    #ifdef _WIN32
        scanf_s("%"NAG_IFMT"%"NAG_IFMT"%*[\n]", &n, &m);
    #else
        scanf("%"NAG_IFMT"%"NAG_IFMT"%*[\n]", &n, &m);
    #endif
    #ifdef _WIN32
        scanf_s("%"NAG_IFMT"%*[\n]", &nnz);
    #else
        scanf("%"NAG_IFMT"%*[\n]", &nnz);
    #endif
    la = 3 * nnz;
    lwork = MAX(n * (m + 3) + m * (m + 5) + 101, 7 * n + 100);
    liwork = 2 * n + 1;
    if (!
        (!a = NAG_ALLOC((la), double)) ||
        (!b = NAG_ALLOC((n), double)) ||
        (!rdiag = NAG_ALLOC((n), double)) ||
        (!wgt = NAG_ALLOC((n), double)) ||
        (!work = NAG_ALLOC((lwork), double)) ||
        (!x = NAG_ALLOC((n), double)) ||
        (!icol = NAG_ALLOC((la), Integer)) ||
        (!irow = NAG_ALLOC((la), Integer)) ||
        (!iwork = NAG_ALLOC((liwork), Integer))
    )
    {
        printf("Allocation failure\n");
        exit_status = -1;
    }

Mark 25
goto END;
}

#elif _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);

#endif _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);

#endif _WIN32
    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 _WIN32
    scanf_s("%NAG_IFMT%*[\n] ", &iterm);
#else
    scanf("%NAG_IFMT%*[\n] ", &iterm);
#endif

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

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

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

    /* Read the non-zero elements of the matrix a*/
    for (i = 0; i < nnz; i++)
    #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 x*/
    #ifdef _WIN32
        for (i = 0; i < n; i++) scanf_s("%lf", &b[i]);
    #else
        for (i = 0; i < n; i++) scanf("%lf", &b[i]);
    #endif

    #ifdef _WIN32
        scanf_s("%*[\n] ");
    #else
        scanf("%*[\n] ");
    #endif

    #ifdef _WIN32
        for (i = 0; i < n; i++) scanf_s("%lf", &x[i]);
    #else
        for (i = 0; i < n; i++) scanf("%lf", &x[i]);
#endif

weight = Nag_SparseNsym_UnWeighted;
monit = 0;
/* Call to initialize the solver
 * 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, &lwneed, work, lwork, &fail);
/* Calculate reciprocal diagonal matrix elements if necessary*/
if (precon == Nag_SparseNsym_Prec) {
    for (i = 0; i < n; i++) iwork[i] = 0;
    for (i = 0; i < nnz; i++) {
        if (irow[i] == icol[i]) {
            iwork[irow[i]-1]++;
            if (a[i] == 0.0) {
                printf("Matrix has a zero diagonal element \n");
                goto END;
            }
            rdiag[(irow[i]-1)] = 1.0/a[i];
        }
    }
    for (i = 0; i < n; i++) {
        if (iwork[i] == 0) {
            printf("Matrix has a missing diagonal element \n");
            goto END;
        }
        if (iwork[i] >= 2) {
            printf("Matrix has a multiple diagonal element \n");
            goto END;
        }
    }
}
/* call solver repeatedly to solve the equations*/
irevcm = 0;
ckxa = Nag_SparseNsym_Check;
ckdd = Nag_SparseNsym_Check;
while (irevcm!=4) {
    switch (irevcm) {
    case 1:
        /* Compute matrix-vector product using
         * nag_sparse_nsym_matvec (f11xac)
         * Real sparse nonsymmetric matrix vector multiply */
        trans = Nag_NoTrans;
        nag_sparse_nsym_matvec(trans, n, nnz, a, irow, icol, ckxa, x, b,
            &fail1);
        ckxa = Nag_SparseNsym_NoCheck;
        break;
    case -1:
        /* Compute transposed matrix-vector product */
        trans = Nag_Trans;
        nag_sparse_nsym_matvec(trans, n, nnz, a, irow, icol, ckxa, x, b,
            &fail1);
        ckxa = Nag_SparseNsym_NoCheck;
        break;
    case 2:
        /* SSOR preconditioning using
         * nag_sparse_nsym_precon_ssor_solve (f11ddc)
         * Solution of linear system involving preconditioning matrix generated
         * by applying SSOR to real sparse nonsymmetric matrix */
        trans = Nag_NoTrans;
nag_sparse_nsym_precon_ssor_solve(trans, n, nnz, a, irow, icol, rdiag,
       omega, ckdd, x, b, &fail1);
ckdd = Nag_SparseNsym_NoCheck;
break;
}
if (fail1.code != NE_NOERROR) ievcm = 6;
}
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_sparse_nsym_basic_solver (f11bec)\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
/* nag_sparse_nsym_basic_diagnostic (f11bfc)
* Real sparse nonsymmetric linear systems, diagnostic
*/
nag_sparse_nsym_basic_diagnostic(&itn, &stplhs, &stprhs, &anorm, &sigmax,
       work, lwork, &fail);
printf(" Converged in %11"NAG_IFMT" iterations\n", itn);
printf(" Matrix norm = %9.3e\n", anorm);
printf(" Final residual norm = %9.3e\n", stplhs);
/* Output x*/
printf(" Solution of linear system\n");
for (i = 0; i < n; i++) printf("%16.4e\n", x[i]);
END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(rdiag);
NAG_FREE(work);
NAG_FREE(x);
NAG_FREE(icol);
NAG_FREE(irow);
NAG_FREE(iwork);
return exit_status;
}

10.2 Program Data

nag_sparse_nsym_precon_ssor_solve (f11ddc) Example Program Data

5    2 : n, m
16   : nnz
Nag_SparseNsym_RGMRES : method
Nag_SparseNsym_Prec : precon
Nag_InfNorm : norm
1    : iterm
1.e-10 1000 : tol, maxitn
0.0  0.0 : anorm, sigmax
1.1   : omega
2.  1  1
1.  1  2
-1.  1  4
-3.  2  2
-2.  2  3
1.  2  5
1.  3  1
5.  3  3
3.  3  4
1.  3  5
-2.  4  1
-3.  4  4
-1.  4  5
4.  5  2
-2.  5  3
-6.  5  5 : a[i], irow[i], icol[i], i=0,...,nnz-1
0.  -7. 33. -19. -28. : b[i], i=0,...,n-1
0.  0.  0.  0. : x[i], i=0,...,n-1

f11ddc.8  Mark 25
10.3 Program Results

nag_sparse_nsym_precon_ssor_solve (f11ddc) Example Program Results

Converged in 12 iterations
Matrix norm = 1.200e+01
Final residual norm = 3.841e-09

Solution of linear system
1.0000e+00
2.0000e+00
3.0000e+00
4.0000e+00
5.0000e+00