

## NAG Library Function Document

### nag\_sparse\_nherm\_precon\_ssor\_solve (f11drc)

#### 1 Purpose

nag\_sparse\_nherm\_precon\_ssor\_solve (f11drc) solves a system of linear equations involving the preconditioning matrix corresponding to SSOR applied to a complex sparse non-Hermitian matrix, represented in coordinate storage format.

#### 2 Specification

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

void nag_sparse_nherm_precon_ssor_solve (Nag_TransType trans, Integer n,
    Integer nnz, const Complex a[], const Integer irow[],
    const Integer icol[], const Complex rdiag[], double omega,
    Nag_SparseNsym_CheckData check, const Complex y[], Complex x[],
    NagError *fail)
```

#### 3 Description

nag\_sparse\_nherm\_precon\_ssor\_solve (f11drc) solves a system of linear equations

$$Mx = y, \quad \text{or} \quad M^H x = y,$$

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) Young (1971) applied to a linear system  $Ax = b$ , where  $A$  is a complex sparse non-Hermitian 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\_nherm\_precon\_ssor\_solve (f11drc) will be to carry out the preconditioning step required in the application of nag\_sparse\_nherm\_basic\_solver (f11bsc) to sparse linear systems. For an illustration of this use of nag\_sparse\_nherm\_precon\_ssor\_solve (f11drc) see the example program given in Section 10. nag\_sparse\_nherm\_precon\_ssor\_solve (f11drc) is also used for this purpose by the Black Box function nag\_sparse\_nherm\_sol (f11dsc).

#### 4 References

Young D (1971) *Iterative Solution of Large Linear Systems* Academic Press, New York

#### 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  
 $M^H x = y$  is solved.

*Constraint:* **trans** = Nag\_NoTrans or Nag\_Trans.

- 2: **n** – Integer *Input*  
*On entry:*  $n$ , the order of the matrix  $A$ .  
*Constraint:*  $n \geq 1$ .
- 3: **nnz** – Integer *Input*  
*On entry:* the number of nonzero elements in the matrix  $A$ .  
*Constraint:*  $1 \leq \mathbf{nnz} \leq n^2$ .
- 4: **a[nnz]** – const Complex *Input*  
*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\_nherm\_sort (f11znc) may be used to order the elements in this way.
- 5: **irow[nnz]** – const Integer *Input*  
6: **icol[nnz]** – const Integer *Input*  
*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 nag\_sparse\_nherm\_sort (f11znc)):  
 $1 \leq \mathbf{irow}[i] \leq n$  and  $1 \leq \mathbf{icol}[i] \leq n$ , for  $i = 0, 1, \dots, \mathbf{nnz} - 1$ ;  
either  $\mathbf{irow}[i - 1] < \mathbf{irow}[i]$  or both  $\mathbf{irow}[i - 1] = \mathbf{irow}[i]$  and  $\mathbf{icol}[i - 1] < \mathbf{icol}[i]$ , for  $i = 1, 2, \dots, \mathbf{nnz} - 1$ .
- 7: **rdiag[n]** – const Complex *Input*  
*On entry:* the elements of the diagonal matrix  $D^{-1}$ , where  $D$  is the diagonal part of  $A$ .
- 8: **omega** – double *Input*  
*On entry:* the relaxation parameter  $\omega$ .  
*Constraint:*  $0.0 < \mathbf{omega} < 2.0$ .
- 9: **check** – Nag\_SparseNsym\_CheckData *Input*  
*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**, **nnz**, **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 Complex *Input*  
*On entry:* the right-hand side vector  $y$ .

- 11: **x[n]** – Complex *Output*  
*On exit:* the solution vector  $x$ .
- 12: **fail** – NagError \* *Input/Output*  
 The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

See Section 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

### NE\_BAD\_PARAM

On entry, argument  $\langle value \rangle$  had an illegal value.

### NE\_INT

On entry, **n** =  $\langle value \rangle$ .

Constraint: **n**  $\geq 1$ .

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**  $\leq \mathbf{n}^2$ .

### 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 How to Use the NAG Library and its Documentation for further information.

### NE\_INVALID\_CS

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**[ $i - 1$ ]  $\leq \mathbf{n}$ .

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**[ $i - 1$ ]  $\leq \mathbf{n}$ .

### NE\_NO\_LICENCE

Your licence key may have expired or may not have been installed correctly.

See Section 3.6.5 in How to Use the NAG Library and its Documentation 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:  $I = \langle value \rangle$ .

### NE\_REAL

On entry, **omega** =  $\langle value \rangle$ .

Constraint:  $0.0 < \mathbf{omega} < 2.0$

**NE\_ZERO\_DIAG\_ELEM**

The matrix  $A$  has no diagonal entry in row  $\langle value \rangle$ .

**7 Accuracy**

If **trans** = 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 **trans** = Nag\_Trans.

**8 Parallelism and Performance**

nag\_sparse\_nherm\_precon\_ssor\_solve (f11drc) is not threaded in any implementation.

**9 Further Comments****9.1 Timing**

The time taken for a call to nag\_sparse\_nherm\_precon\_ssor\_solve (f11drc) is proportional to **nnz**.

**9.2 Use of check**

It is expected that a common use of nag\_sparse\_nherm\_precon\_ssor\_solve (f11drc) will be to carry out the preconditioning step required in the application of nag\_sparse\_nherm\_basic\_solver (f11bsc) to sparse linear systems. In this situation nag\_sparse\_nherm\_precon\_ssor\_solve (f11drc) 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 **check** = Nag\_SparseNsym\_Check for the first of such calls, and **check** = Nag\_SparseNsym\_NoCheck for all subsequent calls.

**10 Example**

This example solves a complex 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\_nherm\_basic\_solver (f11bsc), which returns repeatedly to the calling program with various values of the argument **irevcn**. This argument indicates the action to be taken by the calling program.

If **irevcn** = 1, a matrix-vector product  $v = Au$  is required. This is implemented by a call to nag\_sparse\_nherm\_matvec (f11xnc).

If **irevcn** = -1, a conjugate transposed matrix-vector product  $v = A^H u$  is required in the estimation of the norm of  $A$ . This is implemented by a call to nag\_sparse\_nherm\_matvec (f11xnc).

If **irevcn** = 2, a solution of the preconditioning equation  $Mv = u$  is required. This is achieved by a call to nag\_sparse\_nherm\_precon\_ssor\_solve (f11drc).

If **irevcn** = 4, nag\_sparse\_nherm\_basic\_solver (f11bsc) 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\_nherm\_basic\_solver (f11bsc).

## 10.1 Program Text

```

/* nag_sparse_nherm_precon_ssor_solve (f11drc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
#include <nagf11.h>
int main(void)
{
    /* Scalars */
    Integer exit_status = 0;
    double anorm, omega, sigmax, stplhs, stprhs, tol;
    Integer i, irevcm, iterm, itn, liwork, lwneed, lwork, m,
        maxitn, monit, n, nnz;
    /* Arrays */
    char nag_enum_arg[100];
    Complex *a = 0, *b = 0, *rdiag = 0, *work = 0, *x = 0;
    double *wgt = 0;
    Integer *icol = 0, *irow = 0, *iwork = 0;
    /* NAG types */
    Nag_SparseNsym_CheckData ckdr, ckxn;
    Nag_NormType norm;
    Nag_SparseNsym_PrecType precon;
    Nag_SparseNsym_Method method;
    Nag_TransType trans;
    Nag_SparseNsym_Weight weight;
    NagError fail, fail1;

    INIT_FAIL(fail);
    INIT_FAIL(fail1);

    printf("nag_sparse_nherm_precon_ssor_solve (f11drc) Example Program Results");
    printf("\n\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
    lwork = MAX(121 + n * (3 + m) + m * (m + 5), 120 + 7 * n);
    liwork = 2 * n + 1;
    if (!(a = NAG_ALLOC((nnz), Complex)) ||
        !(b = NAG_ALLOC((n), Complex)) ||
        !(rdiag = 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)) ||
        !(iwork = NAG_ALLOC((liwork), 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, (unsigned)_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, (unsigned)_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, (unsigned)_countof(nag_enum_arg));
#else
    scanf("%99s%[\n]", nag_enum_arg);
#endif
    norm = (Nag_NormType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _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%lf%*[\n]", &anorm, &sigmax);
#else
    scanf("%lf%lf%*[\n]", &anorm, &sigmax);
#endif
#ifdef _WIN32
    scanf_s("%lf%*[\n]", &omega);
#else
    scanf("%lf%*[\n]", &omega);
#endif

    /* Read the matrix a */
    for (i = 0; i < nnz; i++)
#ifdef _WIN32
        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 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
weight = Nag_SparseNsym_UnWeighted;
monit = 0;

/* Call to initialize solver */
/* nag_sparse_nherm_basic_setup (f11brc)
 * Complex sparse non-Hermitian linear systems, setup
 */
nag_sparse_nherm_basic_setup(method, precon, norm, weight, iterm, n, m, tol,
                             maxitn, anorm, sigmax, monit, &lwneed, 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;
}
/* 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 (nag_complex_equal(a[i], nag_complex(0.0, 0.0))) {
                printf("Matrix has a zero diagonal element\n");
                goto END;
            }
        }
        else {
            rdiag[irow[i] - 1] =
                nag_complex_divide(nag_complex(1.0, 0.0), 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;
ckxn = Nag_SparseNsym_Check;
ckdr = Nag_SparseNsym_Check;
while (irevcm != 4) {
    /* nag_sparse_nherm_basic_solver (f11bsc).
     * Complex sparse non-Hermitian linear systems, solver routine
     * preconditioned RGMRES, CGS, Bi-CGSTAB or TFQMR method
     */
    nag_sparse_nherm_basic_solver(&irevcm, x, b, wgt, work, lwork, &fail);
    switch (irevcm) {
    case 1:
        /* Compute matrix-vector product */
        trans = Nag_NoTrans;
        /* nag_sparse_nherm_matvec (f11xnc).
         * Complex sparse non-Hermitian matrix vector multiply
         */
        nag_sparse_nherm_matvec(trans, n, nnz, a, irow, icol, ckxn, x, b,
                                &fail1);
        ckxn = Nag_SparseNsym_NoCheck;
        break;
    case -1:

```

```

/* Compute conjugate transposed matrix-vector product */
trans = Nag_ConjTrans;
nag_sparse_nherm_matvec(trans, n, nnz, a, irow, icol, ckxn, x, b,
                        &fail1);
ckxn = Nag_SparseNsym_NoCheck;
break;
case 2:
/* SSOR preconditioning */
trans = Nag_NoTrans;
/* nag_sparse_nherm_precon_ssor_solve (f11drc).
 * Solution of linear system involving preconditioning matrix generated
 * by applying SSOR to complex sparse non-Hermitian matrix
 */
nag_sparse_nherm_precon_ssor_solve(trans, n, nnz, a, irow, icol, rdiag,
                                   omega, ckdr, x, b, &fail1);

ckdr = Nag_SparseNsym_NoCheck;
break;
case 4:
/* Termination */
break;
default:
goto END;
}
if (fail1.code != NE_NOERROR) {
printf("Error from matrix-vector or preconditioning stage.\n%s\n",
      fail1.message);
exit_status = 2;
goto END;
}
}
if (fail.code != NE_NOERROR) {
printf("Error from nag_sparse_nherm_basic_solver (f11bsc).\n%s\n",
      fail.message);
exit_status = 3;
goto END;
}
/* nag_sparse_nherm_basic_diagnostic (f11btc)
 * Complex sparse non-Hermitian linear systems, diagnostic
 */
nag_sparse_nherm_basic_diagnostic(&itn, &stplhs, &stprhs, &anorm, &sigmax,
                                  work, lwork, &fail1);
printf("Converged in %12" NAG_IFMT " iterations\n", itn);
printf("Matrix norm          = %11.3e\n", anorm);
printf("Final residual norm = %11.3e\n\n", stplhs);
/* Output x */
printf("%14s\n", "Solution");
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(rdiag);
NAG_FREE(work);
NAG_FREE(x);
NAG_FREE(wgt);
NAG_FREE(icol);
NAG_FREE(irow);
NAG_FREE(iwork);
return exit_status;
}

```

## 10.2 Program Data

nag\_sparse\_nherm\_precon\_ssor\_solve (f11drc) Example Program Data

```

5           2           : n, m
16          : nnz
Nag_SparseNsym_CGS : method
Nag_SparseNsym_Prec : precon
Nag_InfNorm       : norm

```



```

1                               : iterm
1.e-10      1000                : tol, maxitn
0.0         0.                  : anorm, sigmax
1.4         : omega
(  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.)
(  0.,  0.)       : x[i], i=0,...,n-1

```

### 10.3 Program Results

nag\_sparse\_nherm\_precon\_ssor\_solve (f11drc) Example Program Results

```

Converged in          5 iterations
Matrix norm          =  1.500e+01
Final residual norm =  2.132e-14

```

```

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)

```

---