NAG Library Function Document

nag_sparse_herm_precon_ssor_solve (f11jrc)

1 Purpose

nag_sparse_herm_precon_ssor_solve (f11jrc) solves a system of linear equations involving the preconditioning matrix corresponding to SSOR applied to a complex sparse Hermitian matrix, represented in symmetric coordinate storage format.

2 Specification

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

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

3 Description

nag_sparse_herm_precon_ssor_solve (f11jrc) solves a system of equations

$$Mx = y$$

involving the preconditioning matrix

$$M = \frac{1}{\omega(2-\omega)}(D+\omega L)D^{-1}(D+\omega L)^H$$

corresponding to symmetric successive-over-relaxation (SSOR) (see Young (1971)) on a linear system $Ax = b$, where A is a sparse complex Hermitian matrix stored in symmetric coordinate storage (SCS) format (see Section 2.1.2 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 and ω is a user-defined relaxation argument. Note that since A is Hermitian the matrix D is necessarily real.

4 References

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

5 Arguments

1: **n** – Integer *Input*

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

Constraint: $\mathbf{n} \geq 1$.

2: **nnz** – Integer *Input*

On entry: the number of nonzero elements in the lower triangular part of the matrix A.

Constraint: $1 \leq \mathbf{nnz} \leq \mathbf{n} \times (\mathbf{n}+1)/2$.

3: **a**[**nnz**] – const Complex *Input*

On entry: the nonzero elements in the lower triangular part of 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_herm_sort (f11zpc) may be used to order the elements in this way.

4: **irow**[**nnz**] – const Integer *Input*
5: **icol**[**nnz**] – const Integer *Input*

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_herm_sort (f11zpc)):

$1 \leq \mathbf{irow}[i] \leq \mathbf{n}$ and $1 \leq \mathbf{icol}[i] \leq \mathbf{irow}[i]$, for $i = 0, 1, \ldots, \mathbf{nnz} - 1$;
$\mathbf{irow}[i-1] < \mathbf{irow}[i]$ or $\mathbf{irow}[i-1] = \mathbf{irow}[i]$ and $\mathbf{icol}[i-1] < \mathbf{icol}[i]$, for $i = 1, 2, \ldots, \mathbf{nnz} - 1$.

6: **rdiag**[**n**] – const double *Input*

On entry: the elements of the diagonal matrix D^{-1}, where D is the diagonal part of A. Note that since A is Hermitian the elements of D^{-1} are necessarily real.

7: **omega** – double *Input*

On entry: the relaxation argument ω.

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

8: **check** – Nag_SparseSym_CheckData *Input*

On entry: specifies whether or not the input data should be checked.

check = Nag_SparseSym_Check
Checks are carried out on the values of **n**, **nnz**, **irow**, **icol** and **omega**.

check = Nag_SparseSym_NoCheck
None of these checks are carried out.

Constraint: **check** = Nag_SparseSym_Check or Nag_SparseSym_NoCheck.

9: **y**[**n**] – const Complex *Input*

On entry: the right-hand side vector y.

10: **x**[**n**] – Complex *Output*

On exit: the solution vector x.

11: **fail** – NagError * *Input/Output*

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

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{n} \geq 1$.

On entry, **nnz** $= \langle value \rangle$.
Constraint: **nnz** ≥ 1.

NE_INT_2

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

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.

NE_INVALID_SCS

On entry, $I = \langle value \rangle$, **icol**$[I - 1] = \langle value \rangle$, **irow**$[I - 1] = \langle value \rangle$.
Constraint: $1 \leq$ **icol**$[i - 1] \leq$ **irow**$[i - 1]$.

On entry, $I = \langle value \rangle$, **irow**$[I - 1] = \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: $1 \leq$ **irow**$[i - 1] \leq$ **n**.

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: $I = \langle value \rangle$. Consider calling nag_sparse_herm_sort (f11zpc) to reorder and sum or remove duplicates.

NE_REAL

On entry, **omega** $= \langle value \rangle$.
Constraint: $0.0 <$ **omega** < 2.0.

NE_ZERO_DIAG_ELEM

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

7 Accuracy

The computed solution x is the exact solution of a perturbed system of equations $(M + \delta M)x = y$, where

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

$c(n)$ is a modest linear function of n, and ϵ is the ***machine precision***.

8 Parallelism and Performance

Not applicable.

9 Further Comments

9.1 Timing

The time taken for a call to nag_sparse_herm_precon_ssor_solve (f11jrc) is proportional to **nnz**.

10 Example

This example program solves the preconditioning equation $Mx = y$ for a 9 by 9 sparse complex Hermitian matrix A, given in symmetric coordinate storage (SCS) format.

10.1 Program Text

```
/* nag_sparse_herm_precon_ssor_solve (f11jrc) Example Program.
 *
 * Copyright 2011, 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;
  Integer                     i, n, nnz;
  /* Arrays */
  char                        nag_enum_arg[100];
  Complex                     *a = 0, *x = 0, *y = 0;
  double                      *rdiag = 0;
  Integer                     *icol = 0, *irow = 0;
  /* NAG types */
  Nag_SparseSym_CheckData     check;
  NagError                    fail;

  INIT_FAIL(fail);

  printf("nag_sparse_herm_precon_ssor_solve (f11jrc) Example Program Results");
  printf("\n");
  /* Skip heading in data file*/
  scanf("%*[^\n]");
  /* Read algorithmic parameters*/
  scanf("%ld%*[^\n] ", &n);
  scanf("%ld%*[^\n] ", &nnz);

  /* Allocate memory */
  if (
      !(a = NAG_ALLOC(nnz, Complex)) ||
      !(x = NAG_ALLOC(n, Complex)) ||
      !(y = NAG_ALLOC(n, Complex)) ||
      !(rdiag = NAG_ALLOC(n, double)) ||
      !(icol = NAG_ALLOC(nnz, Integer)) ||
      !(irow = NAG_ALLOC(nnz, Integer))
      )
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  scanf("%99s%*[^\n] ", nag_enum_arg);
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  check = (Nag_SparseSym_CheckData) nag_enum_name_to_value(nag_enum_arg);
  scanf("%lf%*[^\n]", &omega);
  /* Read the matrix a */
  for (i = 0; i < nnz; i++)
    scanf(" ( %lf , %lf ) %ld%ld%*[^\n]",
          &a[i].re, &a[i].im, &irow[i], &icol[i]);
  /* Read rhs vector y */
  for (i = 0; i < n; i++)
    scanf(" ( %lf , %lf ) ", &y[i].re, &y[i].im);
  scanf("%*[^\n]");
  /* Fill in the diagonal part */
  for (i = 0; i < nnz; i++)
    if (irow[i] == icol[i])
      rdiag[irow[i]-1] = 1.0/(double)(a[i].re);
```

```
  /* nag_sparse_herm_precon_ssor_solve (f11jrc).
   * Solution of linear system involving preconditioning matrix
   * generated by applying SSOR to complex sparse Hermitian matrix
   */
  nag_sparse_herm_precon_ssor_solve(n, nnz, a, irow, icol, rdiag,
                                    omega, check, y, x, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_sparse_herm_precon_ssor_solve (f11jrc)\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

  /* Output x*/
  printf("     Converged Solution\n");
  for (i = 0; i < n; i++)
    printf(" (%13.4e, %13.4e) \n", x[i].re, x[i].im);

 END:
  NAG_FREE(a);
  NAG_FREE(x);
  NAG_FREE(y);
  NAG_FREE(rdiag);
  NAG_FREE(icol);
  NAG_FREE(irow);
  return exit_status;
}
```

10.2 Program Data

```
nag_sparse_herm_precon_ssor_solve (f11jrc) Example Program Data
  9                     : n
  23                    : nnz
 Nag_SparseSym_Check    : check
  1.1                   : omega
 ( 6., 0.)   1    1
 (-1., 1.)   2    1
 ( 6., 0.)   2    2
 ( 0., 1.)   3    2
 ( 5., 0.)   3    3
 ( 5., 0.)   4    4
 ( 2.,-2.)   5    1
 ( 4., 0.)   5    5
 ( 1., 1.)   6    3
 ( 2., 0.)   6    4
 ( 6., 0.)   6    6
 (-4., 3.)   7    2
 ( 0., 1.)   7    5
 (-1., 0.)   7    6
 ( 6., 0.)   7    7
 (-1.,-1.)   8    4
 ( 0.,-1.)   8    6
 ( 9., 0.)   8    8
 ( 1., 3.)   9    1
 ( 1., 2.)   9    5
 (-1., 0.)   9    6
 ( 1., 4.)   9    8
 ( 9., 0.)   9    9     : a[i], irow[i], icol[i], i=0,...,nnz-1
 (  8.,  54.)
 (-10., -92.)
 ( 25.,  27.)
 ( 26., -28.)
 ( 54.,  12.)
 ( 26., -22.)
 ( 47.,  65.)
 ( 71., -57.)
 ( 60.,  70.)           : y[i], i=0,...,n-1
```

10.3 Program Results

```
nag_sparse_herm_precon_ssor_solve (f11jrc) Example Program Results
     Converged Solution
 (    1.0977e+00,    5.9139e+00)
 (    2.2304e-01,   -1.4085e+01)
 (    2.2315e+00,    7.0868e+00)
 (    4.8164e+00,   -6.1807e+00)
 (    6.7632e+00,    1.5690e+00)
 (    3.3531e+00,   -4.7849e+00)
 (    6.6991e-01,   -1.4646e+00)
 (    8.8315e+00,   -3.6326e+00)
 (    4.7685e+00,    1.2130e-01)
```