NAG Library Function Document

nag_sparse_sym_precon_ssor_solve (f11jdc)

1 Purpose

nag_sparse_sym_precon_ssor_solve (f11jdc) solves a system of linear equations involving the preconditioning matrix corresponding to SSOR applied to a real sparse symmetric matrix, represented in symmetric coordinate storage format.

2 Specification

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

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

3 Description

nag_sparse_sym_precon_ssor_solve (f11jdc) solves a system of equations

$$Mx = y$$

involving the preconditioning matrix

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

corresponding to symmetric successive-over-relaxation (SSOR) (see Young (1971)) on a linear system $Ax = b$, where A is a sparse symmetric 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 parameter.

It is envisaged that a common use of nag_sparse_sym_precon_ssor_solve (f11jdc) will be to carry out the preconditioning step required in the application of nag_sparse_sym_basic_solver (f11gec) to sparse linear systems. For an illustration of this use of nag_sparse_sym_precon_ssor_solve (f11jdc) see the example program given in Section 10.1. nag_sparse_sym_precon_ssor_solve (f11jdc) is also used for this purpose by the Black Box function nag_sparse_sym_sol (f11jec).

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 A.

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

3: **a**[**nnz**] – const double *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_sym_sort (f11zbc) 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 these constraints (which may be imposed by a call to nag_sparse_sym_sort (f11zbc)):

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

7: **omega** – double *Input*

On entry: the relaxation parameter 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.

See also Section 9.2.

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

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

On entry: the right-hand side vector y.

10: **x**[**n**] – double *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.
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, **nnz** $=$ ⟨*value*⟩.
Constraint: **nnz** ≥ 1.

NE_INT_2

On entry, **nnz** $=$ ⟨*value*⟩ and **n** $=$ ⟨*value*⟩.
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.

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_SCS

On entry, $I =$ ⟨*value*⟩, **icol**$[I-1] =$ ⟨*value*⟩ and **irow**$[I-1] =$ ⟨*value*⟩.
Constraint: **icol**$[I-1] \geq 1$ and **icol**$[I-1] \leq$ **irow**$[I-1]$.

On entry, $i =$ ⟨*value*⟩, **irow**$[i-1] =$ ⟨*value*⟩ and **n** $=$ ⟨*value*⟩.
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: $i =$ ⟨*value*⟩.

On entry, the location (**irow**$[I-1]$, **icol**$[I-1]$) is a duplicate: $I =$ ⟨*value*⟩. Consider calling nag_sparse_sym_sort (f11zbc) to reorder and sum or remove duplicates.

NE_REAL

On entry, **omega** $=$ ⟨*value*⟩.
Constraint: $0.0 <$ **omega** < 2.0

NE_ZERO_DIAG_ELEM

The matrix A has no diagonal entry in row ⟨*value*⟩.

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_sym_precon_ssor_solve (f11jdc) is proportional to **nnz**.

9.2 Use of check

It is expected that a common use of nag_sparse_sym_precon_ssor_solve (f11jdc) will be to carry out the preconditioning step required in the application of nag_sparse_sym_basic_solver (f11gec) to sparse symmetric linear systems. In this situation nag_sparse_sym_precon_ssor_solve (f11jdc) 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_SparseSym_Check for the first of such calls, and to set **check** = Nag_SparseSym_NoCheck for all subsequent calls.

10 Example

This example solves a sparse symmetric linear system of equations

$$Ax = b,$$

using the conjugate-gradient (CG) method with SSOR preconditioning.

The CG algorithm itself is implemented by the reverse communication function nag_sparse_sym_basic_solver (f11gec), 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 **irevcm** $= 1$, a matrix-vector product $v = Au$ is required. This is implemented by a call to nag_sparse_sym_matvec (f11xec).

If **irevcm** $= 2$, a solution of the preconditioning equation $Mv = u$ is required. This is achieved by a call to nag_sparse_sym_precon_ssor_solve (f11jdc).

If **irevcm** $= 4$, nag_sparse_sym_basic_solver (f11gec) 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_sym_basic_solver (f11gec).

10.1 Program Text

```
/* nag_sparse_sym_precon_ssor_solve (f11jdc) 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, sigerr, sigmax, sigtol, stplhs,
                             stprhs, tol;
  Integer                    i, irevcm, iterm, itn, its, j, listr, lcneed, lcomm,
                             maxitn, maxits, monit, n, nnz, nnz1;
  /* Arrays */
  char                       nag_enum_arg[100];
  double                     *a = 0, *b = 0, *rdiag = 0, *wgt = 0,
                             *commarray = 0, *x = 0;
  Integer                    *icol = 0, *irow = 0, *istr = 0;
```

```
  /* NAG types */
  Nag_NormType              norm;
  Nag_SparseNsym_Method     method;
  Nag_SparseNsym_PrecType   precon;
  Nag_SparseSym_Bisection   sigcmp;
  Nag_SparseSym_CheckData   ckjd, ckxe;
  Nag_SparseSym_Dups        dup;
  Nag_SparseSym_Weight      weight;
  Nag_SparseSym_Zeros       zero;
  NagError                  fail, fail1;

  INIT_FAIL(fail);

  printf("nag_sparse_sym_precon_ssor_solve (f11jdc) 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"%*[^\n]", &n);
#else
  scanf("%"NAG_IFMT"%*[^\n]", &n);
#endif
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n]", &nnz);
#else
  scanf("%"NAG_IFMT"%*[^\n]", &nnz);
#endif

  /* Allocate memory */
  listr = n + 1;
  lcomm = 6 * n + 120;
  if (
      !(a = NAG_ALLOC(nnz, double)) ||
      !(b = NAG_ALLOC(n, double)) ||
      !(rdiag = NAG_ALLOC(n, double)) ||
      !(wgt = NAG_ALLOC(n, double)) ||
      !(commarray = NAG_ALLOC(lcomm, double)) ||
      !(x = NAG_ALLOC(n, double)) ||
      !(icol = NAG_ALLOC(nnz, Integer)) ||
      !(irow = NAG_ALLOC(nnz, Integer)) ||
      !(istr = NAG_ALLOC(listr, Integer))
      )
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

#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
  sigcmp = (Nag_SparseSym_Bisection) 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
  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%"NAG_IFMT"%*[^\n]", &sigtol, &maxits);
#else
  scanf("%lf%"NAG_IFMT"%*[^\n]", &sigtol, &maxits);
#endif
#ifdef _WIN32
  scanf_s("%lf%*[^\n]", &omega);
#else
  scanf("%lf%*[^\n]", &omega);
#endif

  /* Read the matrix a */
  for (i = 0; i <= nnz-1; 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

  /* Sort matrix a removing zero or duplicate elements using
   * nag_sparse_sym_sort (f11zbc).
   */
  nnz1 = nnz;
  dup = Nag_SparseSym_RemoveDups;
  zero = Nag_SparseSym_RemoveZeros;
  nag_sparse_sym_sort (n, &nnz1, a, irow, icol, dup, zero, istr, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_sparse_sym_sort (f11zbc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
  if (nnz != nnz1)
    {
      printf("Warning, input Matrix has zero or duplicate elements\n");
      printf("         nnz has been reduced from %"NAG_IFMT" to %"NAG_IFMT"\n",
             nnz,nnz1);
      nnz = nnz1;
    }

  /* Check for zero diagonal matrix elements and calculate reciprocals.*/
  for (i = 0; i < n; i++)
    {
      /* j points to last element in row i */
      j = istr[i+1]-2;
      if (irow[j] == icol[j])
        rdiag[irow[j]-1] = 1.0/a[j];
```

```
      else
        {
          printf("Matrix has a missing element for diagonal %"NAG_IFMT"\n",i);
          goto END;
        }
    }

  /* Read right-hand side vector b and initial approximate solution x*/
  for (i = 0; i <= n-1; i++)
#ifdef _WIN32
    scanf_s("%lf", &b[i]);
#else
    scanf("%lf", &b[i]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
  for (i = 0; i <= n-1; i++)
#ifdef _WIN32
    scanf_s("%lf", &x[i]);
#else
    scanf("%lf", &x[i]);
#endif

  /* Initialize the basic symmteric solver (f11gec) using
   * nag_sparse_sym_basic_setup (f11gdc)
   */
  weight = Nag_SparseSym_UnWeighted;
  monit = 0;
  nag_sparse_sym_basic_setup(method, precon, sigcmp, norm, weight, iterm, n,
                             tol, maxitn, anorm, sigmax, sigtol, maxits,
                             monit, &lcneed, commarray, lcomm, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_sparse_sym_setup (f11gdc).\n%s\n", fail.message);
      exit_status = 2;
      goto END;
    }

  /* call solver repeatedly to solve the equations */
  irevcm = 0;
  ckxe = Nag_SparseSym_Check;
  ckjd = Nag_SparseSym_Check;
  while (1)
    {
      /* nag_sparse_sym_basic_solver (f11gec).
       * Real sparse symmetric linear systems, preconditioned conjugate gradient
       * or Lanczos method.
       */
      nag_sparse_sym_basic_solver(&irevcm, x, b, wgt, commarray, lcomm, &fail);
      if (irevcm != 4)
        {
          INIT_FAIL(fail1);
          switch (irevcm)
            {
            case 1:
              /* Compute sparse symmetric matrix vector product using
               * nag_sparse_sym_matvec (f11xec).
               */
              nag_sparse_sym_matvec(n, nnz, a, irow, icol, ckxe, x, b, &fail1);
              ckxe = Nag_SparseSym_NoCheck;
              break;
            case 2:
              /* SSOR preconditioning
               * nag_sparse_sym_precon_ssor_solve (f11jdc).
               * Solution of linear system involving preconditioning matrix
               * generated by applying SSOR to real sparse symmetric matrix
               */
              nag_sparse_sym_precon_ssor_solve(n, nnz, a, irow, icol, rdiag,
```

```
                                                    omega, ckjd, x, b, &fail1);
              ckjd = Nag_SparseSym_NoCheck;
            }
          if (fail1.code != NE_NOERROR)
            irevcm = 6;
        }
      else if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_sparse_sym_basic_solver (f11gec).\n%s\n",
                 fail.message);
          exit_status = 3;
          goto END;
        } else goto END_LOOP;
    }
 END_LOOP:
  /* Obtain and print diagnostic statistics using
   * nag_sparse_sym_basic_diagnostic (f11gfc).
   */
  nag_sparse_sym_basic_diagnostic(&itn, &stplhs, &stprhs, &anorm, &sigmax,
                                  &its, &sigerr, commarray, lcomm, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_sparse_sym_basic_diagnostic (f11gfc).\n%s\n",
             fail.message);
      exit_status = 4;
      goto END;
    }
  printf("Converged in   %10"NAG_IFMT" iterations \n", itn);
  printf("Final residual norm = %11.3e\n\n", stplhs);
  /* Output solution */
  printf("%16s\n","Solution");
  for (i = 0; i <= n - 1; i++)
    printf("%16.4e\n", x[i]);

 END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(rdiag);
  NAG_FREE(wgt);
  NAG_FREE(commarray);
  NAG_FREE(x);
  NAG_FREE(icol);
  NAG_FREE(irow);
  NAG_FREE(istr);
  return exit_status;
}
```

10.2 Program Data

```
nag_sparse_sym_precon_ssor_solve (f11jdc) Example Program Data
  7                      : n
 16                      : nnz
 Nag_SparseSym_CG        : method
 Nag_SparseSym_Prec      : precon
 Nag_SparseSym_NoBisect  : sigcmp
 Nag_InfNorm             : norm
  1                      : iterm
 1.0e-6 100              : tol, maxitn
 0.0   0.0               : anorm, sigmax
 0.0  10                 : sigtol, maxits
 1.0                     : omega
  4.   1    1
  1.   2    1
  5.   2    2
  2.   3    3
  2.   4    2
  3.   4    4
 -1.   5    1
  1.   5    4
  4.   5    5
```

```
  1.   6    2
 -2.   6    5
  3.   6    6
  2.   7    1
 -1.   7    2
 -2.   7    3
  5.   7    7          : a[i], irow[i], icol[i], i=0,...,nnz-1
 15.  18.  -8.   21.
 11.  10   29.         : b[i], i=0,...,n-1
  0.   0.   0.    0.
  0.   0.   0.         : x[i], i=0,...,n-1
```

10.3 Program Results

```
nag_sparse_sym_precon_ssor_solve (f11jdc) Example Program Results

Converged in            6 iterations
Final residual norm =   7.105e-15

        Solution
      1.0000e+00
      2.0000e+00
      3.0000e+00
      4.0000e+00
      5.0000e+00
      6.0000e+00
      7.0000e+00
```