nag_sparse_sym_sol (f11jec)

1. Purpose
nag_sparse_sym_sol (f11jec) solves a real sparse symmetric system of linear equations, represented
in symmetric coordinate storage format, using a conjugate gradient or Lanczos method, without
preconditioning, with Jacobi or with SSOR preconditioning.

2. Specification
#include <nag.h>
#include <nagf11.h>

void nag_sparse_sym_sol(Nag_SparseSym_Method method,
    Nag_SparseSym_PrecType precon, Integer n, Integer nnz,
    double a[], Integer irow[], Integer icol[],
    double omega, double b[], double tol,
    Integer maxitn, double x[], double *rnorm,
    Integer *itn, Nag_Sparse_Comm *comm, NagError *fail)

3. Description
This routine solves a real sparse symmetric linear system of equations:

\[ Ax = b, \]

using a preconditioned conjugate gradient method (see Barrett et al. (1994)), or a preconditioned
Lanczos method based on the algorithm SYMMLQ (Paige and Saunders (1975)). The conjugate
gradient method is more efficient if \( A \) is positive-definite, but may fail to converge for indefinite
matrices. In this case the Lanczos method should be used instead. For further details see Barrett
et al. (1994).

The routine allows the following choices for the preconditioner:

- no preconditioning;
- Jacobi preconditioning (see Young (1971);
- symmetric successive-over-relaxation (SSOR) preconditioning (see Young (1971)).

For incomplete Cholesky (IC) preconditioning see nag_sparse_sym_chol_sol (f11jcc).

The matrix \( A \) is represented in symmetric coordinate storage (SCS) format (see Section 2.1.2 of
the Chapter Introduction) in the arrays \( a, irow \) and \( icol \). The array \( a \) holds the non-zero entries in
the lower triangular part of the matrix, while \( irow \) and \( icol \) hold the corresponding row and column
indices.

4. Parameters
method
Input: specifies the iterative method to be used. The possible choices are:

- if method = Nag_SparseSym_CG then the conjugate gradient method is used;
- if method = Nag_SparseSym_Lanczos then the Lanczos method (SYMMLQ) is used.

Constraint: method = Nag_SparseSym_CG or Nag_SparseSym_Lanczos.

precon
Input: specifies the type of preconditioning to be used. The possible choices are :

- if precon = Nag_SparseSym_NoPrec then no preconditioning is used;
- if precon = Nag_SparseSym_SSORPrec then symmetric successive-over-relaxation is
  used;
- if precon = Nag_SparseSym_JacPrec then Jacobi preconditioning is used.

Constraint: precon = Nag_SparseSym_NoPrec, Nag_SparseSym_SSORPrec or
Nag_SparseSym_JacPrec.
nag_sparse_sym_sol

Input: the order of the matrix A.
Constraint: \( n \geq 1 \).

**nnz**

Input: the number of non-zero elements in the lower triangular part of the matrix A.
Constraint: \( 1 \leq \text{nnz} \leq n \times (n+1)/2 \).

**a[nnz]**

Input: the non-zero elements of 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 routine nag_sparse_sym_sort (f11zbc) may be used to order the elements in this way.

**irow[nnz]**

**icol[nnz]**

Input: the row and column indices of the non-zero elements supplied in A.
Constraint: \( \text{irow[i]} \) and \( \text{icol[i]} \) must satisfy the following constraints (which may be imposed by a call to nag_sparse_sym_sort (f11zbc)):
- \( 1 \leq \text{irow}[i] \leq n \), and \( 1 \leq \text{icol}[i] \leq \text{irow}[i] \), for \( i = 0, 1, \ldots, \text{nnz} - 1 \).
- \( \text{irow}[i-1] < \text{irow}[i] \), or
- \( \text{irow}[i-1] = \text{irow}[i] \) and \( \text{icol}[i-1] < \text{icol}[i] \), for \( i = 1, 2, \ldots, \text{nnz} - 1 \).

**omega**

Input: if \( \text{precon} = \text{Nag_SparseSym_SSORPrec} \), \( \omega \) is the relaxation parameter \( \omega \) to be used in the SSOR method. Otherwise \( \omega \) need not be initialised.
Constraint: \( 0.0 \leq \omega \leq 2.0 \).

**b[n]**

Input: the right-hand side vector b.

**tol**

Input: the required tolerance. Let \( x_k \) denote the approximate solution at iteration \( k \), and \( r_k \) the corresponding residual. The algorithm is considered to have converged at iteration \( k \) if:

\[
\|r_k\|_\infty \leq \tau \times (\|b\|_\infty + \|A\|_\infty \|x_k\|_\infty).
\]

If \( \text{tol} \leq 0.0 \), \( \tau = \max(\sqrt{\epsilon}, \sqrt{n} \epsilon) \) is used, where \( \epsilon \) is the *machine precision*. Otherwise \( \tau = \max(\text{tol}, 10 \epsilon, \sqrt{n} \epsilon) \) is used.
Constraint: \( \text{tol} < 1.0 \).

**maxitn**

Input: the maximum number of iterations allowed.
Constraint: \( \text{maxitn} \geq 1 \).

**x[n]**

Input: an initial approximation of the solution vector x.
Output: an improved approximation to the solution vector x.

**rnorm**

Input: the final value of the residual norm \( \|r_k\|_\infty \), where \( k \) is the output value of \( \text{itn} \).

**itn**

Output: the number of iterations carried out.

**comm**

Input/Output: a pointer to a structure of type Nag_Sparse_Comm whose members are used by the iterative solver.

**fail**

The NAG error parameter, see the Essential Introduction to the NAG C Library.
5. Error Indications and Warnings

**NE_BAD_PARAM**
On entry, parameter `method` had an illegal value.
On entry, parameter `precon` had an illegal value.

**NE_REAL_ARG_GE**
On entry, `tol` must not be greater than or equal to 1.0: `tol = ⟨value⟩`.

**NE_INT_ARG_LT**
On entry, `n` must not be less than 1: `n = ⟨value⟩`.
On entry, `maxitn` must not be less than 1: `maxitn = ⟨value⟩`.

**NE_REAL**
On entry, `omega` = `⟨value⟩`.
Constraint: `0.0 ≤ omega ≤ 2.0`.

**NE_INT_2**
On entry, `nnz` = `⟨value⟩`, `n` = `⟨value⟩`.
Constraint: `1 ≤ nnz ≤ n × (n+1)/2`.

**NE_SYMM_MATRIX_DUP**
A non-zero element has been supplied which does not lie in the lower triangular part of the matrix `A`, is out of order, or has duplicate row and column indices, i.e., one or more of the following constraints has been violated:

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

\[ irow[i - 1] < irow[i], \text{ or} \]

\[ irow[i - 1] = irow[i] \text{ and } icol[i - 1] < icol[i], \text{ for } i = 1, 2, \ldots, nnz - 1. \]

Call `nag_sparse_sym_sort (f11zbc)` to reorder and sum or remove duplicates.

**NE_COEFF_NOT_POS_DEF**
The matrix of coefficients appears not to be positive-definite (conjugate gradient method only).

**NE_ZERO_DIAGONAL_ELEM**
The matrix `A` has a zero diagonal element. Jacobi and SSOR preconditioners are not appropriate for this problem.

**NE_PRECOND_NOT_POS_DEF**
The preconditioner appears not to be positive-definite.

**NE_ACC_LIMIT**
The required accuracy could not be obtained. However, a reasonable accuracy has been obtained and further iterations cannot improve the result.

**NE_NOT_REQ_ACC**
The required accuracy has not been obtained in `maxitn` iterations.

**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 consult NAG for assistance.

**NE_ALLOC_FAIL**
Memory allocation failed.

6. Further Comments
The time taken by `nag_sparse_sym_sol (f11jec)` for each iteration is roughly proportional to `nnz`. One iteration with the Lanczos method (SYMMLQ) requires a slightly larger number of operations than one iteration with the conjugate gradient method.

The number of iterations required to achieve a prescribed accuracy cannot be easily determined a priori, as it can depend dramatically on the conditioning and spectrum of the preconditioned matrix of the coefficients `\tilde{A} = M^{-1}A`.

[NP3275/5/pdf] 3.f11jec.3
6.1. Accuracy

On successful termination, the final residual \( r_k = b - Ax_k \), where \( k = \text{itn} \), satisfies the termination criterion

\[
\|r_k\|_\infty \leq \tau \times (\|b\|_\infty + \|A\|_\infty \|x_k\|_\infty).
\]

The value of the final residual norm is returned in \text{rnorm}.

6.2. References


7. See Also

\text{nag_sparse_sym_chol_sol (f11jcc)}
\text{nag_sparse_sym_sort (f11zbc)}

8. Example

This example program solves a symmetric positive-definite system of equations using the conjugate gradient method, with SSOR preconditioning.

8.1. Program Text

/* nag_sparse_sym_sol (f11jec) Example Program. */
/* Copyright 1999 Numerical Algorithms Group. */
/* Mark 5, 1998. */
/* */
#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nag_string.h>
#include <nagf11.h>

/* f11jec Example Program Text */

main()
{
    double *a=0, *b=0, *x=0;
    double omega;
    double rnorm;
    double tol;

    Integer *icol, *irow;
    Integer i, n, maxitn, itn, nnz;

    Nag_SparseSym_Method method;
    Nag_SparseSym_PrecType precon;
    Nag_Sparse_Comm comm;

    char char_enum[20];
    Vprintf("f11jec Example Program Results\n");
    /* Skip heading in data file */
    Vscanf(" %*\[^
");
/* Read algorithmic parameters */
Vscanf("%ld%*[\-\n]", &n);
Vscanf("%ld%*[\-\n]", &nnz);
Vscanf("%s", char_enum);
if (!strcmp(char_enum, "CG"))
    method = Nag_SparseSym_CG;
else if (!strcmp(char_enum, "Lanczos"))
    method = Nag_SparseSym_Lanczos;
else
{
    Vprintf("Unrecognised string for method enum representation.\n");
    exit (EXIT_FAILURE);
}
Vscanf("%s%*[\-\n]", char_enum);
if (!strcmp(char_enum, "Prec"))
    precon = Nag_SparseSym_Prec;
else if (!strcmp(char_enum, "NoPrec"))
    precon = Nag_SparseSym_NoPrec;
else if (!strcmp(char_enum, "SSORPrec"))
    precon = Nag_SparseSym_SSORPrec;
else if (!strcmp(char_enum, "JacPrec"))
    precon = Nag_SparseSym_JacPrec;
else
{
    Vprintf("Unrecognised string for precon enum representation.\n");
    exit (EXIT_FAILURE);
}
Vscanf("%lf%*[\-\n]", &omega);
Vscanf("%lf%ld%*[\-\n]", &tol, &maxitn);
x = NAG_ALLOC(n, double);
b = NAG_ALLOC(n, double);
a = NAG_ALLOC(nnz, double);
irow = NAG_ALLOC(nnz, Integer);
icol = NAG_ALLOC(nnz, Integer);
if (!irow || !icol || !a || !x || !b)
{
    Vprintf("Allocation failure\n");
    exit (EXIT_FAILURE);
}
/* Read the matrix a */
for (i = 1; i <= nnz; ++i)
    Vscanf("%lf%ld%ld%*[\-\n]", &a[i-1], &irow[i-1], &icol[i-1]);
/* Read right-hand side vector b and initial approximate solution x */
for (i = 1; i <= n; ++i)
    Vscanf("%lf", &b[i-1]);
Vscanf("%*[\-\n]");
for (i = 1; i <= n; ++i)
    Vscanf("%lf", &x[i-1]);
Vscanf("%*[\-\n]");
/* Solve Ax = b */
f11jec(method, precon, n, nnz, a, irow, icol, omega, b, tol, maxitn, x, &rnorm, &itn, &comm, NAGERR_DEFAULT);
Vprintf("%s%10ld%s\n", "Converged in", itn, " iterations");
Vprintf("%s%16.3e\n", "Final residual norm =", rnorm);
/* Output x */
for (i = 1; i <= n; ++i)
    Vprintf("%16.4e\n", x[i-1]);
NAG_FREE(irow);
NAG_FREE(icol);
NAG_FREE(a);
NAG_FREE(x);
NAG_FREE(b);
exit(EXIT_SUCCESS);
}

8.2. Program Data

f11jec Example Program Data

<table>
<thead>
<tr>
<th></th>
<th>n</th>
<th>nnz</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

CG SSORPrec method, precon

<table>
<thead>
<tr>
<th></th>
<th>1.0E-6</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>tol, maxitn</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4, 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2, 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2, 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3, 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1, 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1, 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4, 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5, 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6, 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2, 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1, 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1, 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2, 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1, 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2, 9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[i-1], irow[i-1], icol[i-1], i=1,...,nnz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15, 18, 21,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11, 10, 29,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0, 0, 0,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0, 0, 0,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b[i-1], i=1,...,n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x[i-1], i=1,...,n</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8.3. Program Results

f11jec Example Program Results

Converged in 6 iterations

Final residual norm = 5.026e-06

<table>
<thead>
<tr>
<th></th>
<th>1.0000e+00</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0000e+00</td>
<td></td>
</tr>
<tr>
<td>3.0000e+00</td>
<td></td>
</tr>
<tr>
<td>4.0000e+00</td>
<td></td>
</tr>
<tr>
<td>5.0000e+00</td>
<td></td>
</tr>
<tr>
<td>6.0000e+00</td>
<td></td>
</tr>
<tr>
<td>7.0000e+00</td>
<td></td>
</tr>
</tbody>
</table>