NAG Library Function Document

nag_real_sym_posdef_tridiag_lin_solve (f04bgc)

1 Purpose

nag_real_sym_posdef_tridiag_lin_solve (f04bgc) computes the solution to a real system of linear equations \( AX = B \), where \( A \) is an \( n \) by \( n \) symmetric positive definite tridiagonal matrix and \( X \) and \( B \) are \( n \) by \( r \) matrices. An estimate of the condition number of \( A \) and an error bound for the computed solution are also returned.

2 Specification

#include <nag.h>
#include <nagf04.h>

void nag_real_sym_posdef_tridiag_lin_solve (Nag_OrderType order, Integer n, 
Integer nrhs, double d[], double e[], double b[], Integer pdb, 
double *rcond, double *errbnd, NagError *fail)

3 Description

\( A \) is factorized as \( A = LDL^T \), where \( L \) is a unit lower bidiagonal matrix and \( D \) is diagonal, and the factored form of \( A \) is then used to solve the system of equations.

4 References

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, 
Philadelphia http://www.netlib.org/lapack/lug


5 Arguments

1:  
order – Nag_OrderType  

On entry: the order argument specifies the two-dimensional storage scheme being used, i.e., row-
major ordering or column-major ordering. C language defined storage is specified by 
order = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed 
explanation of the use of this argument.

Constraint: order = Nag_RowMajor or Nag_ColMajor.

2:  
n – Integer  

On entry: the number of linear equations \( n \), i.e., the order of the matrix \( A \).

Constraint: \( n \geq 0 \).

3:  
nrhs – Integer  

On entry: the number of right-hand sides \( r \), i.e., the number of columns of the matrix \( B \).

Constraint: nrhs \( \geq 0 \).

4:  
d[\text{dim}] – double  

Note: the dimension, dim, of the array \( d \) must be at least max(1, n).
On entry: must contain the \( n \) diagonal elements of the tridiagonal matrix \( A \).

On exit: if fail.code = NE_NOERROR or NE_RCOND, \( d \) is overwritten by the \( n \) diagonal elements of the diagonal matrix \( D \) from the \( \text{LDL}^T \) factorization of \( A \).

5: \( e[dim] \) – double 

\text{Input/Output}

\text{Note:} the dimension, \( dim \), of the array \( e \) must be at least \( \max(1, n - 1) \).

On entry: must contain the \( (n - 1) \) subdiagonal elements of the tridiagonal matrix \( A \).

On exit: if fail.code = NE_NOERROR or NE_RCOND, \( e \) is overwritten by the \( (n - 1) \) subdiagonal elements of the unit lower bidiagonal matrix \( L \) from the \( \text{LDL}^T \) factorization of \( A \). \( (e \) can also be regarded as the superdiagonal of the unit upper bidiagonal factor \( U \) from the \( U^T D U \) factorization of \( A \).)

6: \( b[dim] \) – double 

\text{Input/Output}

\text{Note:} the dimension, \( dim \), of the array \( b \) must be at least \( \max(1, \text{pdb \times nrhs}) \) when \( \text{order} = \text{Nag\_ColMajor} \);
\( \max(1, n \times \text{pdb}) \) when \( \text{order} = \text{Nag\_RowMajor} \).

The \((i, j)\)th element of the matrix \( B \) is stored in
- \( b[(j - 1) \times \text{pdb} + i - 1] \) when \( \text{order} = \text{Nag\_ColMajor} \);
- \( b[(i - 1) \times \text{pdb} + j - 1] \) when \( \text{order} = \text{Nag\_RowMajor} \).

On entry: the \( n \) by \( r \) matrix of right-hand sides \( B \).

On exit: if fail.code = NE_NOERROR or NE_RCOND, the \( n \) by \( r \) solution matrix \( X \).

7: \( \text{pdb} \) – Integer 

\text{Input}

On entry: the stride separating row or column elements (depending on the value of \( \text{order} \)) in the array \( b \).

\text{Constraints:}
- if \( \text{order} = \text{Nag\_ColMajor} \), \( \text{pdb} \geq \max(1, n) \);
- if \( \text{order} = \text{Nag\_RowMajor} \), \( \text{pdb} \geq \max(1, \text{nrhs}) \).

8: \( \text{rcond} \) – double * 

\text{Output}

On exit: if fail.code = NE_NOERROR or NE_RCOND, an estimate of the reciprocal of the condition number of the matrix \( A \), computed as \( \text{rcond} = 1/(\|A\|_1 \|A^{-1}\|_1) \).

9: \( \text{errbnd} \) – double * 

\text{Output}

On exit: if fail.code = NE_NOERROR or NE_RCOND, an estimate of the forward error bound for a computed solution \( \hat{x} \), such that \( \|\hat{x} - x\|_1/\|x\|_1 \leq \text{errbnd} \), where \( \hat{x} \) is a column of the computed solution returned in the array \( b \) and \( x \) is the corresponding column of the exact solution \( X \). If \( \text{rcond} \) is less than \text{machine precision}, then \( \text{errbnd} \) is returned as unity.

10: \( \text{fail} \) – NagError * 

\text{Input/Output}

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

6 \text{ 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 \langle value \rangle had an illegal value.

NE_INT
On entry, n = \langle value \rangle.
Constraint: n \geq 0.

On entry, nrhs = \langle value \rangle.
Constraint: nrhs \geq 0.

On entry, pdb = \langle value \rangle.
Constraint: pdb > 0.

NE_INT_2
On entry, pdb = \langle value \rangle and n = \langle value \rangle.
Constraint: pdb \geq \max(1, n).

On entry, pdb = \langle value \rangle and nrhs = \langle value \rangle.
Constraint: pdb \geq \max(1, nrhs).

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_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_POS_DEF
The principal minor of order \langle value \rangle of the matrix A is not positive definite. The factorization has not been completed and the solution could not be computed.

NE_RCOND
A solution has been computed, but rcond is less than machine precision so that the matrix A is numerically singular.

Accuracy
The computed solution for a single right-hand side, \( \hat{x} \), satisfies an equation of the form

\[(A + E)\hat{x} = b,\]

where

\[\|E\|_1 = O(\epsilon)\|A\|_1\]

and \( \epsilon \) is the machine precision. An approximate error bound for the computed solution is given by

\[\frac{\|\hat{x} - x\|_1}{\|x\|_1} \leq \kappa(A)\|E\|_1,\]

where \( \kappa(A) = \|A^{-1}\|_1\|A\|_1 \), the condition number of A with respect to the solution of the linear equations. nag_real_sym_posdef_tridiag_lin_solve (f04bgc) uses the approximation \( \|E\|_1 = \epsilon\|A\|_1 \) to estimate errbnd. See Section 4.4 of Anderson et al. (1999) for further details.
8 Parallelism and Performance

nag_real_sym_posdef_tridiag_lin_solve (f04bgc) is not threaded by NAG in any implementation.
nag_real_sym_posdef_tridiag_lin_solve (f04bgc) makes calls to BLAS and/or LAPACK routines, which
may be threaded within the vendor library used by this implementation. Consult the documentation for
the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the
OpenMP environment used within this function. Please also consult the Users’ Note for your
implementation for any additional implementation-specific information.

9 Further Comments

The double allocatable memory required is n. In this case the factorization and the solution X have been
computed, but rcond and errbnd have not been computed.

The total number of floating-point operations required to solve the equations AX = B is proportional to
nr. The condition number estimation requires \(O(n)\) floating-point operations.

See Section 15.3 of Higham (2002) for further details on computing the condition number of tridiagonal
matrices.

The complex analogue of nag_real_sym_posdef_tridiag_lin_solve (f04bgc) is
nag_herm_posdef_tridiag_lin_solve (f04cgc).

10 Example

This example solves the equations

\[ AX = B, \]

where \(A\) is the symmetric positive definite tridiagonal matrix

\[
A = \begin{pmatrix}
4.0 & -2.0 & 0 & 0 & 0 \\
-2.0 & 10.0 & -6.0 & 0 & 0 \\
0 & -6.0 & 29.0 & 15.0 & 0 \\
0 & 0 & 15.0 & 25.0 & 8.0 \\
0 & 0 & 0 & 8.0 & 5.0
\end{pmatrix}
\quad \text{and} \quad
B = \begin{pmatrix}
6.0 & 10.0 \\
9.0 & 4.0 \\
2.0 & 9.0 \\
14.0 & 65.0 \\
7.0 & 23.0
\end{pmatrix}.
\]

An estimate of the condition number of \(A\) and an approximate error bound for the computed solutions
are also printed.

10.1 Program Text

/* nag_real_sym_posdef_tridiag_lin_solve (f04bgc) Example Program. *
* Copyright 2014 Numerical Algorithms Group. *
* Mark 8, 2004. */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf04.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    double errbnd, rcond;
    Integer exit_status, i, j, n, nrhs, pdb;

    /* Arrays */
    double *b, *d = 0, *e = 0;
/* Nag Types */
NagError fail;
Nag_OrderType order;

#ifdef NAG_COLUMN_MAJOR
#define B(I, J) b[(J-1)*pdb + I - 1]
order = Nag_ColMajor;
#else
#define B(I, J) b[(I-1)*pdb + J - 1]
order = Nag_RowMajor;
#endif
#define b_ref(a_1, a_2) b[(a_2)*8 + a_1 - 9]

exit_status = 0;
INIT_FAIL(fail);

printf("nag_real_sym_posdef_tridiag_lin_solve (f04bgc) Example "
"Program Results\n\n");

/* Skip heading in data file */
#ifdef _WIN32
scanf_s("%*[\n ] ");
#else
scanf("%*[\n ] ");
#endif
#ifdef _WIN32
scanf_s("%"NAG_IFMT"%"NAG_IFMT"%*[\n ] ", &n, &nrhs);
#else
scanf("%"NAG_IFMT"%"NAG_IFMT"%*[\n ] ", &n, &nrhs);
#endif
if (n > 0 && nrhs > 0)
{
    /* Allocate memory */
    if (!(b = NAG_ALLOC(n*nrhs, double)) ||
    !(d = NAG_ALLOC(n, double)) ||
    !(e = NAG_ALLOC(n-1, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
#ifdef NAG_COLUMN_MAJOR
    pdb = n;
#else
    pdb = nrhs;
#endif
    else
    {
        printf("%s\n", "n and/or nrhs too small");
        exit_status = 1;
        return exit_status;
    }
#endif

/* Read A from data file */
for (i = 1; i <= n; ++i)
{
    #ifdef _WIN32
    scanf_s("%lf", &d[i-1]);
    #else
    scanf("%lf", &d[i-1]);
    #endif
    #ifdef _WIN32
    scanf_s("%*[\n ] ");
    #else
    scanf("%*[\n ] ");
    #endif
    for (i = 1; i <= n-1; ++i)
```c
#ifdef _WIN32
    scanf_s("%lf", &e[i-1]);
#else
    scanf("%lf", &e[i-1]);
#endif
}
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
/* Read B from data file */
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= nrhs; ++j)
    {
#ifdef _WIN32
        scanf_s("%lf", &B(i, j));
#else
        scanf("%lf", &B(i, j));
#endif
    }
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
/* Solve the equations AX = B for X */
/* nag_real_sym_posdef_tridiag_lin_solve (f04bgc). */
/* Computes the solution and error-bound to a real symmetric
  positive-definite tridiagonal system of linear equations */
  nag_real_sym_posdef_tridiag_lin_solve(order, n, nrhs, d, e, b, pdb, &rcond,
  &errbnd, &fail);
if (fail.code == NE_NOERROR)
{
    /* Print solution, estimate of condition number and approximate */
    /* error bound */
    /* nag_gen_real_mat_print (x04cac). */
    /* Print real general matrix (easy-to-use) */
    fflush(stdout);
    nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
    nrhs, b, pdb, "Solution", 0, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_gen_real_mat_print (x04cac).\n", fail.message);
        exit_status = 1;
        goto END;
    }
    printf("%s\n%6s%10.1e\n", "Estimate of condition number", "", 1.0/rcond);
    printf("%s\n%6s%10.1e\n", "Estimate of error bound for computed solutions",
    "", errbnd);
}
else if (fail.code == NE_RCOND)
{
    /* Matrix A is numerically singular. Print estimate of */
    /* reciprocal of condition number and solution */
    printf("%s\n", "Estimate of reciprocal of condition number", "", rcond);
    printf("%s\n", "Estimate of reciprocal of condition number", "", rcond);
```

The code snippet appears to be part of a C program. It involves reading data from files, solving a system of linear equations, and printing the solution along with error bounds and condition numbers. The comments indicate that the program uses specific functions from the NAG Library (e.g., `f04bgc`) to solve tridiagonal systems.
/* nag_gen_real_mat_print (x04cac), see above. */
fflush(stdout);
nag_gen_real_mat_print(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n,
nrhs, b, pdb, "Solution", 0, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_gen_real_mat_print (x04cac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
else if (fail.code == NE_POS_DEF)
{
    printf("%s%3"NAG_IFMT"%s\n", "The leading minor of order ", fail.errnum, " is not positive definite");
}
else
{
    printf(
"Error from nag_real_sym_posdef_tridiag_lin_solve (f04bgc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
END:
NAG_FREE(b);
NAG_FREE(d);
NAG_FREE(e);
return exit_status;

10.2 Program Data

nag_real_sym_posdef_tridiag_lin_solve (f04bgc) Example Program Data

5 2 :Values of N and NRHS
4.0 10.0 29.0 25.0 5.0 :End of diagonal D
-2.0 -6.0 15.0 8.0 :End of sub-diagonal E

6.0 10.0
9.0 4.0
2.0 9.0
14.0 65.0
7.0 23.0 :End of matrix B

10.3 Program Results

nag_real_sym_posdef_tridiag_lin_solve (f04bgc) Example Program Results

Solution
1
2.5000 2.0000
2 2.0000 -1.0000
3 1.0000 -3.0000
4 -1.0000 6.0000
5 3.0000 -5.0000

Estimate of condition number
1.1e+02

Estimate of error bound for computed solutions
1.2e-14