NAG Library Function Document

nag_zppequ (f07gtc)

1 Purpose
nag_zppequ (f07gtc) computes a diagonal scaling matrix $S$ intended to equilibrate a complex $n$ by $n$ Hermitian positive definite matrix $A$, stored in packed format, and reduce its condition number.

2 Specification
#include <nag.h>
#include <nagf07.h>

void nag_zppequ (Nag_OrderType order, Nag_UploType uplo, Integer n,
const Complex ap[], double s[], double *scond, double *amax,
NagError *fail)

3 Description
nag_zppequ (f07gtc) computes a diagonal scaling matrix $S$ chosen so that

$s_j = 1/\sqrt{a_{jj}}.$

This means that the matrix $B$ given by

$B = SAS,$

has diagonal elements equal to unity. This in turn means that the condition number of $B$, $\kappa_2(B)$, is within a factor $n$ of the matrix of smallest possible condition number over all possible choices of diagonal scalings (see Corollary 7.6 of Higham (2002)).

4 References

5 Arguments

1: order – Nag_OrderType

Input

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: uplo – Nag_UploType

Input

On entry: indicates whether the upper or lower triangular part of $A$ is stored in the array ap, as follows:

uplo = Nag_Upper

The upper triangle of $A$ is stored.

uplo = Nag_Lower

The lower triangle of $A$ is stored.

Constraint: uplo = Nag_Upper or Nag_Lower.
3: \( n \) – Integer

\textit{Input}

\textit{On entry:} \( n \), the order of the matrix \( A \).

\textit{Constraint:} \( n \geq 0 \).

4: \( \text{ap}[\text{dim}] \) – const Complex

\textit{Input}

\textit{Note:} the dimension, \( \text{dim} \), of the array \( \text{ap} \) must be at least \( \text{max}(1, n \times (n + 1)/2) \).

\textit{On entry:} the \( n \) by \( n \) Hermitian matrix \( A \), packed by rows or columns.

The storage of elements \( A_{ij} \) depends on the \textit{order} and \textit{uplo} arguments as follows:

- if \( \text{order} = \text{Nag\_ColMajor} \) and \( \text{uplo} = \text{Nag\_Upper} \),
  \( A_{ij} \) is stored in \( \text{ap}[(j - 1) \times j/2 + i - 1] \), for \( i \leq j \);
- if \( \text{order} = \text{Nag\_ColMajor} \) and \( \text{uplo} = \text{Nag\_Lower} \),
  \( A_{ij} \) is stored in \( \text{ap}[(2n - j) \times (j - 1)/2 + i - 1] \), for \( i \geq j \);
- if \( \text{order} = \text{Nag\_RowMajor} \) and \( \text{uplo} = \text{Nag\_Upper} \),
  \( A_{ij} \) is stored in \( \text{ap}[(2n - i) \times (i - 1)/2 + j - 1] \), for \( i \leq j \);
- if \( \text{order} = \text{Nag\_RowMajor} \) and \( \text{uplo} = \text{Nag\_Lower} \),
  \( A_{ij} \) is stored in \( \text{ap}[(i - 1) \times i/2 + j - 1] \), for \( i \geq j \).

Only the elements of \( \text{ap} \) corresponding to the diagonal elements \( A \) are referenced.

5: \( s[n] \) – double

\textit{Output}

\textit{On exit:} if \( \text{fail} \).\text{code} = \text{NE\_NOERROR} \), \( s \) contains the diagonal elements of the scaling matrix \( S \).

6: \( \text{scond} \) – double *

\textit{Output}

\textit{On exit:} if \( \text{fail} \).\text{code} = \text{NE\_NOERROR} \), \( \text{scond} \) contains the ratio of the smallest value of \( s \) to the largest value of \( s \). If \( \text{scond} \geq 0.1 \) and \( \text{amax} \) is neither too large nor too small, it is not worth scaling by \( S \).

7: \( \text{amax} \) – double *

\textit{Output}

\textit{On exit:} \( \text{max} |A_{ij}| \). If \( \text{amax} \) is very close to overflow or underflow, the matrix \( A \) should be scaled.

8: \( \text{fail} \) – \text{Nag\_Error} *

\textit{Input/Output}

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

6 \ Error Indicators and Warnings

\textbf{NE\_ALLOC\_FAIL}

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

\textbf{NE\_BAD\_PARAM}

On entry, argument \( \langle \text{value} \rangle \) had an illegal value.

\textbf{NE\_INT}

On entry, \( n = \langle \text{value} \rangle \).

\textit{Constraint:} \( n \geq 0 \).

\textbf{NE\_INTERNAL\_ERROR}

An internal error has occurred in this function. Check the function call and any array sizes. If the

\textit{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_MAT_NOT_POS_DEF**

The \langle value\rangle th diagonal element of A is not positive (and hence A cannot be positive definite).

**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.

7 Accuracy

The computed scale factors will be close to the exact scale factors.

8 Parallelism and Performance

Not applicable.

9 Further Comments

The real analogue of this function is nag_dppequ (f07gfc).

10 Example

This example equilibrates the Hermitian positive definite matrix A given by

\[
A = \begin{pmatrix}
3.23 & 1.51 - 1.92i & (1.90 + 0.84i) \times 10^5 & 0.42 + 2.50i \\
1.51 + 1.92i & 3.58 & (0.23 + 1.11i) \times 10^5 & -1.18 + 1.37i \\
(1.90 - 0.84i) \times 10^5 & (-0.23 - 1.11i) \times 10^5 & 4.09 \times 10^{10} & (2.33 - 0.14i) \times 10^5 \\
0.42 - 2.50i & -1.18 - 1.37i & (2.33 + 0.14i) \times 10^5 & 4.29
\end{pmatrix}.
\]

Details of the scaling factors and the scaled matrix are output.

10.1 Program Text

/* nag_zppequ (f07gtc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* * Mark 23, 2011. */
*/
#include <stdio.h>
#include <f07.h>
#include <f07gtc.h>

int main(void)
{
    /* Scalars */
    double amax, big, scond, small;
    Integer exit_status = 0, i, j, n;

    /* Arrays */
    Complex *ap = 0;
    double *s = 0;
    char nag_enum_arg[40];

    /* Nag Types */
NagError fail;
Nag_OrderType order;
Nag_UploType uplo;

#ifdef NAG_COLUMN_MAJOR
#define A_UPPER(I, J) ap[J*(J-1)/2 + I - 1]
#define A_LOWER(I, J) ap[(2*n-J)*(J-1)/2 + I - 1]
#else
#define A_LOWER(I, J) ap[I*(I-1)/2 + J - 1]
#define A_UPPER(I, J) ap[(2*n-I)*(I-1)/2 + J - 1]
#endif

order = Nag_ColMajor;

INIT_FAIL(fail);

printf("nag_zppequ (f07gtc) Example Program Results\n\n");

/* Skip heading in data file */
#ifdef _WIN32
scanf_s("%*[\n"]);
#else
scanf("%*[\n"]);
#endif

#define NAG_IFMT "%d"
#include " nag_enum_name_to_value (x04nac)."
if (n < 0)
{
    printf("Invalid n\n");
    exit_status = 1;
    goto END;
}

uplo = (Nag_UploType) nag_enum_name_to_value(nag_enum_arg);

#define NAG_ALLOC "malloc"
#include " nag_zlauup (f07gtc)."

/* Read the upper or lower triangular part of the matrix A from data file */

if (uplo == Nag_Upper)
    for (i = 1; i <= n; ++i)
        for (j = i; j <= n; ++j)
            scanf(" ( %lf , %lf )", &A_UPPER(i, j).re, &A_UPPER(i, j).im);
else
    scanf(" ( %lf , %lf )", &A_UPPER(i, j).re, &A_UPPER(i, j).im);
#endif
#endif

else if (uplo == Nag_Lower)
    for (i = 1; i <= n; ++i)
        for (j = 1; j <= i; ++j)
            scanf(" ( %lf , %lf )", &A_LOWER(i, j).re, &A_LOWER(i, j).im);
else
    scanf(" ( %lf , %lf )", &A_LOWER(i, j).re, &A_LOWER(i, j).im);
#endif
```c
#ifdef _WIN32
    scanf_s("%*[^
");
#else
    scanf("%*[^
");
#endif

/* Print the matrix A using nag_pack_complex_mat_print_comp (x04ddc). */
fflush(stdout);

nag_pack_complex_mat_print_comp(order, uplo, Nag_NonUnitDiag, n, ap,
    Nag_BracketForm, "%11.2e", "Matrix A",
    Nag_IntegerLabels, 0, Nag_IntegerLabels, 0,
    80, 0, 0, &fail);

if (fail.code != NE_NOERROR)
{
    printf("Error from nag_pack_complex_mat_print_comp (x04ddc).
    fail.message);
    exit_status = 1;
    goto END;
}
printf("\n");

/* Compute diagonal scaling factors using nag_zppequ (f07gtc). */

nag_zppequ(order, uplo, n, ap, s, &scond, &amax, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zppequ (f07gtc).
    fail.message");
    exit_status = 1;
    goto END;
}

/* Print scond, amax and the scale factors */

printf("scond = %10.1e, amax = %10.1e\n", scond, amax);

for (i = 0; i < n; ++i) printf("%11.1e%s", s[i], i%6 == 5?"\n":" ");
printf("\n");

/* Compute values close to underflow and overflow using
   * nag_real_safe_small_number (x02amc), nag_machine_precision (x02ajc) and
   * nag_real_base (x02bhc)
   */
small = nag_real_safe_small_number / (nag_machine_precision * nag_real_base);
b = 1.0 / small;
if (scond < 0.1 || amax < small || amax > b)
{
    /* Scale A */
    if (uplo == Nag_Upper)
        for (j = 1; j <= n; ++j)
            for (i = 1; i <= j; ++i)
                { 
                    A_UPPER(i, j).re *= s[i-1] * s[j-1];
                    A_UPPER(i, j).im *= s[i-1] * s[j-1];
                }
    else
        for (j = 1; j <= n; ++j)
            for (i = j; i <= n; ++i)
                { 
                    A_LOWER(i, j).re *= s[i-1] * s[j-1];
                    A_LOWER(i, j).im *= s[i-1] * s[j-1];
                }

    /* Print the scaled matrix using
    * nag_pack_complex_mat_print_comp (x04ddc).
    */
    fflush(stdout);
    nag_pack_complex_mat_print_comp(order, uplo, Nag_NonUnitDiag, n, ap,
        Nag_BracketForm, 0, "Scaled matrix",
        Nag_IntegerLabels, 0, Nag_IntegerLabels, 0,
        80, 0, 0, &fail);
    if (fail.code != NE_NOERROR)
    {
```
printf("Error from nag_pack_complx_mat_print_comp (x04ddc).\n%s\n", fail.message);
exit_status = 1;
go to END;
}
}
END:
NAG_FREE(ap);
NAG_FREE(s);
return exit_status;
}
#define A_UPPER
#define A_LOWER

10.2 Program Data

nag_zppequ (f07gtc) Example Program Data
4 : n
Nag_Upper : uplo
( 3.23, 0.00) ( 1.51,-1.92) ( 1.90e+05, 0.84e+05) ( 0.42 , 2.50 )
( 3.58, 0.00) (-0.23e+05, 1.11e+05) (-1.18 , 1.37 )
( 4.09e+10, 0.00 ) ( 2.33e+05,-0.14e+05)
( 4.29 , 0.00 ) : A

10.3 Program Results

nag_zppequ (f07gtc) Example Program Results

Matrix A

1 ( 3.23e+00, 0.00e+00) ( 1.51e+00, -1.92e+00)
2 ( 3.58e+00, 0.00e+00)
3
4

1 ( 1.90e+05, 8.40e+04) ( 4.20e-01, 2.50e+00)
2 ( -2.30e+04, 1.11e+05) (-1.18e+00, 1.37e+00)
3 ( 4.09e+10, 0.00e+00) ( 2.33e+05, -1.40e+04)
4 ( 4.29e+00, 0.00e+00)

scond = 8.9e-06, amax = 4.1e+10

Diagonal scaling factors

5.6e-01 5.3e-01 4.9e-06 4.8e-01

Scaled matrix

1 ( 1.0000, 0.0000) ( 0.4441, -0.5646) ( 0.5227, 0.2311)
2 ( 1.0000, 0.0000) (-0.0601, 0.2901)
3 ( 1.0000, 0.0000)
4

1 ( 0.1128, 0.6716)
2 ( -0.3011, 0.3496)
3 ( 0.5562, -0.0334)
4 ( 1.0000, 0.0000)