

# NAG Library Function Document

## nag\_zpoequ (f07ftc)

### 1 Purpose

nag\_zpoequ (f07ftc) computes a diagonal scaling matrix  $S$  intended to equilibrate a complex  $n$  by  $n$  Hermitian positive definite matrix  $A$  and reduce its condition number.

### 2 Specification

```
#include <nag.h>
#include <nagf07.h>

void nag_zpoequ (Nag_OrderType order, Integer n, const Complex a[],
                Integer pda, double s[], double *scond, double *amax, NagError *fail)
```

### 3 Description

nag\_zpoequ (f07ftc) 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

Higham N J (2002) *Accuracy and Stability of Numerical Algorithms* (2nd Edition) SIAM, Philadelphia

### 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: **n** – Integer *Input*

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

*Constraint:*  $n \geq 0$ .

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

**Note:** the dimension, *dim*, of the array **a** must be at least  $\max(1, \mathbf{pda} \times \mathbf{n})$ .

The ( $i, j$ )th element of the matrix  $A$  is stored in

$$\begin{aligned} &\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1] \text{ when } \mathbf{order} = \text{Nag\_ColMajor}; \\ &\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1] \text{ when } \mathbf{order} = \text{Nag\_RowMajor}. \end{aligned}$$

*On entry:* the matrix  $A$  whose scaling factors are to be computed. Only the diagonal elements of the array  $\mathbf{a}$  are referenced.

- 4: **pda** – Integer *Input*  
*On entry:* the stride separating row or column elements (depending on the value of **order**) in the array  $\mathbf{a}$ .  
*Constraint:*  $\mathbf{pda} \geq \max(1, \mathbf{n})$ .
- 5: **s[n]** – double *Output*  
*On exit:* if **fail.code** = NE\_NOERROR,  $\mathbf{s}$  contains the diagonal elements of the scaling matrix  $S$ .
- 6: **scond** – double \* *Output*  
*On exit:* if **fail.code** = NE\_NOERROR, **scond** contains the ratio of the smallest value of  $\mathbf{s}$  to the largest value of  $\mathbf{s}$ . If  $\mathbf{scond} \geq 0.1$  and **amax** is neither too large nor too small, it is not worth scaling by  $S$ .
- 7: **amax** – double \* *Output*  
*On exit:*  $\max |a_{ij}|$ . If **amax** is very close to overflow or underflow, the matrix  $A$  should be scaled.
- 8: **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  $\langle value \rangle$  had an illegal value.

### NE\_INT

On entry,  $\mathbf{n} = \langle value \rangle$ .

Constraint:  $\mathbf{n} \geq 0$ .

On entry,  $\mathbf{pda} = \langle value \rangle$ .

Constraint:  $\mathbf{pda} > 0$ .

### NE\_INT\_2

On entry,  $\mathbf{pda} = \langle value \rangle$  and  $\mathbf{n} = \langle value \rangle$ .

Constraint:  $\mathbf{pda} \geq \max(1, \mathbf{n})$ .

### 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\_MAT\_NOT\_POS\_DEF

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

## 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\_dpoequ (f07ffc).

## 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_zpoequ (f07ftc) Example Program.
 *
 * Copyright 2008 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */

#include <stdio.h>
#include <nag.h>
#include <nagx04.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagx02.h>

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

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

#ifdef NAG_COLUMN_MAJOR
#define A(I, J) a[(J-1)*pda + I - 1]
    order = Nag_ColMajor;
#else
#define A(I, J) a[(I-1)*pda + J - 1]
    order = Nag_RowMajor;
#endif

    INIT_FAIL(fail);

    printf("nag_zpoequ (f07ftc) Example Program Results\n\n");

    /* Skip heading in data file */
    scanf("%*[\n]");
    scanf("%ld%*[\n]", &n);

    pda = n;
    /* Allocate memory */

```

```

if (!(a = NAG_ALLOC(n * n, Complex)) ||
    !(s = NAG_ALLOC(n, double)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read the upper triangular part of the matrix A from data file */
for (i = 1; i <= n; ++i)
    for (j = i; j <= n; ++j)
        scanf(" (%lf , %lf )", &A(i, j).re, &A(i, j).im);
scanf("%*[\n]");

/* Print the matrix A using nag_gen_complx_mat_print_comp (x04dbc). */
fflush(stdout);
nag_gen_complx_mat_print_comp(order, Nag_UpperMatrix, Nag_NonUnitDiag, n, n,
                              a, pda, 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_gen_complx_mat_print_comp (x04dbc).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}
printf("\n");

/* Compute diagonal scaling factors using nag_zpoequ (f07ftc). */
nag_zpoequ(order, n, a, pda, s, &scond, &amax, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_zpoequ (f07ftc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print scond, amax and the scale factors */
printf("scond = %10.1e, amax = %10.1e\n", scond, amax);
printf("\nDiagonal scaling factors\n");
for (i = 0; i < n; ++i) printf("%11.1e%s", s[i], i%7 == 6?"\n":" ");
printf("\n\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);
big = 1.0 / small;
if (scond < 0.1 || amax < small || amax > big)
{
    /* Scale A */
    for (j = 1; j <= n; ++j)
        for (i = 1; i <= j; ++i)
        {
            A(i, j).re *= s[i-1]*s[j-1];
            A(i, j).im *= s[i-1]*s[j-1];
        }

    /* Print the scaled matrix using
     * nag_gen_complx_mat_print_comp (x04dbc).
     */
    fflush(stdout);
    nag_gen_complx_mat_print_comp(order, Nag_UpperMatrix, Nag_NonUnitDiag, n,
                                  n, a, pda, Nag_BracketForm, 0,
                                  "Scaled matrix", Nag_IntegerLabels, 0,
                                  Nag_IntegerLabels, 0, 80, 0, 0, &fail);

    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_gen_complx_mat_print_comp (x04dbc).\n%s\n",
              fail.message);
        exit_status = 1;
        goto END;
    }
}

```

```

        fail.message);
        exit_status = 1;
        goto END;
    }
}

END:
    NAG_FREE(a);
    NAG_FREE(s);

    return exit_status;
}
#undef A

```

## 10.2 Program Data

```

nag_zpoequ (f07ftc) Example Program Data
4
( 3.23, 0.00) ( 1.51,-1.92) ( 1.90e+05, 0.84e+05) ( 0.42 , 2.50 ) : n
                ( 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\_zpoequ (f07ftc) Example Program Results

Matrix A

```

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

1 3 ( 1.90e+05, 8.40e+04) ( 4.20e-01, 2.50e+00)
2 4 ( -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 ( 1.0000, 0.0000) ( 0.4441, -0.5646) ( 0.5227, 0.2311)
2 2 ( 1.0000, 0.0000) ( -0.0601, 0.2901)
3 ( 1.0000, 0.0000)
4

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

```

---