NAG Library Function Document

nag_zgeequ (f07atc)

1 Purpose

nag_zgeequ (f07atc) computes real diagonal scaling matrices D_R and D_C intended to equilibrate a complex m by n matrix A and reduce its condition number.

2 Specification

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

```
void nag_zgeequ (Nag_OrderType order, Integer m, Integer n,
     const Complex a[], Integer pda, double r[], double c[], double *rowcnd,
     double *colcnd, double *amax, NagError *fail)
```

3 Description

nag_zgeequ (f07atc) computes the diagonal scaling matrices. The diagonal scaling matrices are chosen to try to make the elements of largest absolute value in each row and column of the matrix B given by

$$B = D_R A D_C$$

have absolute value 1. The diagonal elements of D_R and D_C are restricted to lie in the safe range $(\delta, 1/\delta)$, where δ is the value returned by function nag_real_safe_small_number (X02AMC). Use of these scaling factors is not guaranteed to reduce the condition number of A but works well in practice.

4 References

None.

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: **m** – Integer *Input*

On entry: m, the number of rows of the matrix A.

Constraint: **m** ≥ 0.

3: **n** – Integer *Input*

On entry: n, the number of columns of the matrix A.

Constraint: **n** ≥ 0.

4: **a**[dim] – const Complex *Input*

Note: the dimension, *dim*, of the array **a** must be at least

$\max(1, \mathbf{pda} \times \mathbf{n})$ when **order** = Nag_ColMajor;
$\max(1, \mathbf{m} \times \mathbf{pda})$ when **order** = Nag_RowMajor.

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

$\mathbf{a}[(j-1) \times \mathbf{pda} + i - 1]$ when **order** = Nag_ColMajor;
$\mathbf{a}[(i-1) \times \mathbf{pda} + j - 1]$ when **order** = Nag_RowMajor.

On entry: the matrix A whose scaling factors are to be computed.

5: **pda** – Integer *Input*

On entry: the stride separating row or column elements (depending on the value of **order**) in the array **a**.

Constraints:

if **order** = Nag_ColMajor, $\mathbf{pda} \geq \max(1, \mathbf{m})$;
if **order** = Nag_RowMajor, $\mathbf{pda} \geq \max(1, \mathbf{n})$.

6: **r**[**m**] – double *Output*

On exit: if **fail.code** = NE_NOERROR or **fail.code** = NE_MAT_COL_ZERO, **r** contains the row scale factors, the diagonal elements of D_R. The elements of **r** will be positive.

7: **c**[**n**] – double *Output*

On exit: if **fail.code** = NE_NOERROR, **c** contains the column scale factors, the diagonal elements of D_C. The elements of **c** will be positive.

8: **rowcnd** – double * *Output*

On exit: if **fail.code** = NE_NOERROR or **fail.code** = NE_MAT_COL_ZERO, **rowcnd** contains the ratio of the smallest value of $\mathbf{r}[i-1]$ to the largest value of $\mathbf{r}[i-1]$. If $\mathbf{rowcnd} \geq 0.1$ and **amax** is neither too large nor too small, it is not worth scaling by D_R.

9: **colcnd** – double * *Output*

On exit: if **fail.code** = NE_NOERROR, **colcnd** contains the ratio of the smallest value of $\mathbf{c}[i-1]$ to the largest value of $\mathbf{c}[i-1]$.

If $\mathbf{colcnd} \geq 0.1$, it is not worth scaling by D_C.

10: **amax** – double * *Output*

On exit: $\max|a_{ij}|$. If **amax** is very close to overflow or underflow, the matrix A should be scaled.

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, $\mathbf{m} = \langle value \rangle$.
Constraint: $\mathbf{m} \geq 0$.

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 $\mathbf{m} = \langle value \rangle$.
Constraint: $\mathbf{pda} \geq \max(1, \mathbf{m})$.

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.

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_COL_ZERO

Column $\langle value \rangle$ of A is exactly zero.

NE_MAT_ROW_ZERO

Row $\langle value \rangle$ of A is exactly zero.

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_dgeequ (f07afc).

10 Example

This example equilibrates the general matrix A given by

$$A = \begin{pmatrix} -1.34 + 2.55i & (0.28 + 3.17i) \times 10^{10} & -6.39 - 2.20i \\ -1.70 - 1.41i & (3.31 - 0.15i) \times 10^{10} & -0.15 + 1.34i \\ (2.41 + 0.39i) \times 10^{-10} & -0.56 + 1.47i & (-0.83 - 0.69i) \times 10^{-10} \end{pmatrix}.$$

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

10.1 Program Text

```
/* nag_zgeequ (f07atc) Example Program.
 *
 * Copyright 2014 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, colcnd, rowcnd, small;
  Integer       i, j, m, n, pda;
  Integer       exit_status = 0;

  /* Arrays */
  Complex       *a = 0;
  double        *c = 0, *r = 0;

  /* Nag Types */
  NagError      fail;
  Nag_OrderType order;
  Nag_Boolean   scaled = Nag_FALSE;

#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_zgeequ (f07atc) 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"%*[^\n]", &n);
#else
  scanf("%"NAG_IFMT"%*[^\n]", &n);
#endif
  if (n < 0)
    {
      printf("Invalid n\n");
      exit_status = 1;
      return exit_status;
    }

  m = n;
  pda = n;

  /* Allocate memory */
  if (!(a = NAG_ALLOC(m*n, Complex)) ||
      !(c = NAG_ALLOC(n, double)) ||
      !(r = NAG_ALLOC(m, double)))
    {
      printf("Allocation failure\n");
```

```
      exit_status = -1;
      goto END;
    }

  /* Read the n by n matrix A from data file */
  for (i = 1; i <= n; ++i)
      for (j = 1; j <= n; ++j)
#ifdef _WIN32
        scanf_s(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#else
        scanf(" ( %lf , %lf )", &A(i, j).re, &A(i, j).im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif

  /* Print the matrix A using nag_gen_complx_mat_print_comp (x04dbc). */
  fflush(stdout);
  nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, 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 row and column scaling factors */
  nag_zgeequ(order, n, n, a, pda, r, c, &rowcnd, &colcnd, &amax, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_zgeequ (f07atc).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }

  /* Print rowcnd, colcnd, amax and the scale factors */
  printf("rowcnd = %10.1e, colcnd = %10.1e, amax = %10.1e\n\n", rowcnd,
          colcnd, amax);
  printf("Row scale factors\n");
  for (i = 1; i <= n; ++i) printf("%11.2e%s", r[i-1], i%7 == 0?"\n":" ");
  printf("\n\nColumn scale factors\n");
  for (i = 1; i <= n; ++i) printf("%11.2e%s", c[i-1], i%7 == 0?"\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 (colcnd < 0.1)
    {
      /* column scale A */
      scaled = Nag_TRUE;
      for (j = 1; j <= n; ++j)
        for (i = 1; i <= n; ++i) {
          A(i, j).re *= c[j - 1];
          A(i, j).im *= c[j - 1];
        }
    }
  if (rowcnd < 0.1 || amax < small || amax > big)
    {
      /* row scale A */
      scaled = Nag_TRUE;
```

```
      for (j = 1; j <= n; ++j)
        for (i = 1; i <= n; ++i) {
          A(i, j).re *= r[i - 1];
          A(i, j).im *= r[i - 1];
        }
    }
  if (scaled)
    {
      /* Print the scaled matrix using nag_gen_complx_mat_print_comp (x04dbc) */
      fflush(stdout);
      nag_gen_complx_mat_print_comp(order, Nag_GeneralMatrix, 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;
        }
    }

 END:
  NAG_FREE(a);
  NAG_FREE(c);
  NAG_FREE(r);

  return exit_status;
}
```

10.2 Program Data

```
nag_zgeequ (f07atc) Example Program Data

   3                                                                  : n

 (-1.34e+00, 2.55e+00) ( 0.28e+10, 3.17e+10) (-6.39e+00,-2.20e+00)
 (-1.70e+00,-1.41e+00) ( 3.31e+10,-0.15e+10) (-0.15e+00, 1.34e+00)
 ( 2.41e-10, 0.39e-10) (-0.56e+00, 1.47e+00) (-0.83e-10,-0.69e-10) : matrix A
```

10.3 Program Results

```
nag_zgeequ (f07atc) Example Program Results

 Matrix A
                             1                          2
 1  (   -1.34e+00,   2.55e+00)  (    2.80e+09,   3.17e+10)
 2  (   -1.70e+00,  -1.41e+00)  (    3.31e+10,  -1.50e+09)
 3  (    2.41e-10,   3.90e-11)  (   -5.60e-01,   1.47e+00)

                             3
 1  (   -6.39e+00,  -2.20e+00)
 2  (   -1.50e-01,   1.34e+00)
 3  (   -8.30e-11,  -6.90e-11)

rowcnd =    5.9e-11, colcnd =    1.4e-10, amax =    3.5e+10

Row scale factors
   2.90e-11    2.89e-11    4.93e-01

Column scale factors
   7.25e+09    1.00e+00    4.02e+09
```

```
 Scaled matrix
                      1                   2                   3
 1  ( -0.2816,  0.5359)  (  0.0812,  0.9188)  ( -0.7439, -0.2561)
 2  ( -0.3562, -0.2954)  (  0.9566, -0.0434)  ( -0.0174,  0.1555)
 3  (  0.8607,  0.1393)  ( -0.2759,  0.7241)  ( -0.1642, -0.1365)
```