1 Purpose

nag_zgbequ (f07btc) computes diagonal scaling matrices $D_R$ and $D_C$ intended to equilibrate a complex $m$ by $n$ band matrix $A$ of band width $(k_l + k_u + 1)$, and reduce its condition number.

2 Specification

```c
#include <nag.h>
#include <nagf07.h>
void nag_zgbequ (Nag_OrderType order, Integer m, Integer n, Integer kl,
                Integer ku, const Complex ab[], Integer pdab, double r[], double c[],
                double *rowcnd, double *colcnd, double *amax, NagError *fail)
```

3 Description

nag_zgbequ (f07btc) 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

$$ B = D_RADF_C $$

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 $\delta$ 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

   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

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

   Constraint: $m \geq 0$.

3: n – Integer

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

   Constraint: $n \geq 0$.

4: kl – Integer

   On entry: $k_l$, the number of subdiagonals of the matrix $A$.

   Constraint: $k_l \geq 0$. 

Mark 25
5: \(\mathbf{ku}\) – Integer

*Input*

*On entry:* \(\mathbf{ku}\), the number of superdiagonals of the matrix \(\mathbf{A}\).

*Constraint:* \(\mathbf{ku} \geq 0\).

6: \(\mathbf{ab}[\text{dim}]\) – const Complex

*Input*

*Note:* the dimension, \(\text{dim}\), of the array \(\mathbf{ab}\) must be at least

\[
\max(1, \mathbf{pdab} \times \mathbf{n}) \text{ when } \mathbf{order} = \text{Nag\_ColMajor};
\]

\[
\max(1, \mathbf{m} \times \mathbf{pdab}) \text{ when } \mathbf{order} = \text{Nag\_RowMajor}.
\]

*On entry:* the \(m\) by \(n\) band matrix \(\mathbf{A}\) whose scaling factors are to be computed. This is stored as a notional two-dimensional array with row elements or column elements stored contiguously. The storage of elements \(\mathbf{A}_{ij}\), for row \(i = 1, \ldots, m\) and column \(j = \max(1, i - k\mathbf{u}), \ldots, \min(n, i + k\mathbf{u})\), depends on the \(\mathbf{order}\) argument as follows:

\[
\begin{align*}
\text{if } \mathbf{order} = \text{Nag\_ColMajor}, & \quad \mathbf{A}_{ij} \text{ is stored as } \mathbf{ab}[j - 1] \times \mathbf{pdab} + \mathbf{ku} + i - j; \\
\text{if } \mathbf{order} = \text{Nag\_RowMajor}, & \quad \mathbf{A}_{ij} \text{ is stored as } \mathbf{ab}[i - 1] \times \mathbf{pdab} + \mathbf{kl} + j - i.
\end{align*}
\]

*See Section 9 in nag_zgbsv (f07bnc) for further details.*

7: \(\mathbf{pdab}\) – Integer

*Input*

*On entry:* the stride separating row or column elements (depending on the value of \(\mathbf{order}\)) of the matrix \(\mathbf{A}\) in the array \(\mathbf{ab}\).

*Constraint:* \(\mathbf{pdab} \geq \mathbf{kl} + \mathbf{ku} + 1\).

8: \(\mathbf{r}[\mathbf{m}]\) – double

*Output*

*On exit:* if \(\mathbf{fail.code} = \text{NE\_NOERROR}\) or \(\mathbf{fail.code} = \text{NE\_MAT\_COL\_ZERO}\), \(\mathbf{r}\) contains the row scale factors, the diagonal elements of \(\mathbf{D}_R\). The elements of \(\mathbf{r}\) will be positive.

9: \(\mathbf{c}[\mathbf{n}]\) – double

*Output*

*On exit:* if \(\mathbf{fail.code} = \text{NE\_NOERROR}\), \(\mathbf{c}\) contains the column scale factors, the diagonal elements of \(\mathbf{D}_C\). The elements of \(\mathbf{c}\) will be positive.

10: \(\mathbf{rowcnd}\) – double *

*Output*

*On exit:* if \(\mathbf{fail.code} = \text{NE\_NOERROR}\) or \(\mathbf{fail.code} = \text{NE\_MAT\_COL\_ZERO}\), \(\mathbf{rowcnd}\) 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 \(\mathbf{amax}\) is neither too large nor too small, it is not worth scaling by \(\mathbf{D}_R\).

11: \(\mathbf{colcnd}\) – double *

*Output*

*On exit:* if \(\mathbf{fail.code} = \text{NE\_NOERROR}\), \(\mathbf{colcnd}\) 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 \(\mathbf{D}_C\).

12: \(\mathbf{amax}\) – double *

*Output*

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

13: \(\mathbf{fail}\) – 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, $kl = ⟨value⟩$.
Constraint: $kl \geq 0$.

On entry, $ku = ⟨value⟩$.
Constraint: $ku \geq 0$.

On entry, $m = ⟨value⟩$.
Constraint: $m \geq 0$.

On entry, $n = ⟨value⟩$.
Constraint: $n \geq 0$.

On entry, $pdab = ⟨value⟩$.
Constraint: $pdab > 0$.

**NE_INT_3**

On entry, $pdab = ⟨value⟩$, $kl = ⟨value⟩$ and $ku = ⟨value⟩$.
Constraint: $pdab \geq kl + ku + 1$.

**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 ⟨value⟩ of $A$ is exactly zero.

**NE_MAT_ROW_ZERO**

Row ⟨value⟩ 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_dgbequ (f07bfc).

10 Example

This example equilibrates the complex band matrix $A$ given by

$$A = \begin{pmatrix}
-1.65 + 2.26i & (-2.05 - 0.85i) \times 10^{-10} & 0.97 - 2.84i & 0 \\
0.00 + 6.30i & (-1.48 - 1.75i) \times 10^{-10} & -3.99 + 4.01i & 0.59 - 0.48i \\
0 & -0.77 + 2.83i & (-1.06 + 1.94i) \times 10^{10} & (3.33 - 1.04i) \times 10^{10} \\
0 & 0.48 - 1.09i & -0.46 - 1.72i & 0
\end{pmatrix}.$$  

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

10.1 Program Text

/* nag_zgbequ (f07btc) 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 exit_status = 0, i, j, kl, ku, n, pdab;

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

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

   #ifdef NAG_COLUMN_MAJOR
   #define AB(I, J) ab[(J-1)*pdab + ku + I - J]
   order = Nag_ColMajor;
   #else
   #define AB(I, J) ab[(I-1)*pdab + kl + J - I]
   order = Nag_RowMajor;
   #endif

   INIT_FAIL(fail);
   printf("nag_zgbequ (f07btc) 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"%NAG_IFMT"%*[\n]", &n, &kl, &ku);
   #else
   scanf("%"NAG_IFMT"%NAG_IFMT"%NAG_IFMT"%*[\n]", &n, &kl, &ku);
   #endif
#else
scanf("%"NAG_IFMT"%"NAG_IFMT"%"NAG_IFMT"%*[\n]", &n, &kl, &ku);
#endif
if (n<0 || kl<0 || ku<0)
{
  printf("Invalid n or kl or ku\n");
  exit_status = 1;
  goto END;
}
/* Allocate memory */
if (!(ab = NAG_ALLOC((kl+ku+1) * n, Complex)) ||
  !(c = NAG_ALLOC(n, double)) ||
  !(r = NAG_ALLOC(n, double)))
{
  printf("Allocation failure\n");
  exit_status = -1;
  goto END;
}
pdab = kl+ku+1;

/* Read the band matrix A from data file */
for (i = 1; i <= n; ++i)
  for (j = MAX(i - kl, 1); j <= MIN(i + ku, n); ++j)
#ifndef _WIN32
  scanf_s(" ( %lf , %lf )", &AB(i, j).re, &AB(i, j).im);
#else
  scanf(" ( %lf , %lf )", &AB(i, j).re, &AB(i, j).im);
#endif
#ifndef _WIN32
  scanf_s("%*[\n]);
#else
  scanf("%*[\n]);
#endif
#endif

/* Print the matrix A using nag_band_complex_mat_print (x04dec). */
fflush(stdout);
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_band_complex_mat_print (x04dec).\n", fail.message);
  exit_status = 1;
  goto END;
}
printf("\n");

/* Compute row and column scaling factors using nag_zgbequ (f07btc). */
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_zgbequ (f07btc).\n", fail.message);
  exit_status = 1;
  goto END;
}

/* Print rowcnd, colcnd, amax and the scale factors */
printf("Row scale factors\n");
for (i = 1; i <= n; ++i) printf("%11.2e\n", r[i-1], i%7 == 0 ? "": "");

/* 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. / small;

if (colcnd < 0.1)
{
  scaled = Nag_TRUE;
  /* column scale A */
  for (j = 1; j <= n; ++j)
    for (i = MAX(1, j - ku); i <= MIN(n, j + kl); ++i)
      {
        AB(i, j).re *= c[j - 1];
        AB(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 = MAX(1, j - ku); i <= MIN(n, j + kl); ++i)
      {
        AB(i, j).re *= r[i-1];
        AB(i, j).im *= r[i-1];
      }
}

if (scaled)
{
  /* Print the row and column scaled matrix using
   * nag_band_complex_mat_print (x04dec).
   */
  fflush(stdout);
  nag_band_complex_mat_print(order, n, n, kl, ku, ab, pdab, "Scaled matrix",
                             0, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_band_complex_mat_print (x04dec).\n%s\n", fail.message);
      exit_status = 1;
      goto END;
    }
}

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

return exit_status;
}
#define AB

10.2 Program Data

nag_zgbequ (f07btc) Example Program Data

4 1 2 : n, kl and ku
(-1.65, 2.26) (-2.05e-10, -8.50e-11) ( 9.70e-01, -2.84e+00)
( 0.00, 6.30) (-1.48e-10, -1.75e-10) (-3.99e+00, 4.01e+00)
(-7.70e-01, 2.83e+00) (-1.06e+10, 1.94e+10)
( 3.33e+10, -1.04e+10)
(-0.46e+00, -1.72e+00) : matrix A
10.3 Program Results

nag_zgbequ (f07btc) Example Program Results

Matrix A

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1.6500e+00</td>
<td>-2.0500e-10</td>
<td>9.7000e-01</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.2600e+00</td>
<td>-8.5000e-11</td>
<td>-2.8400e+00</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.0000e+00</td>
<td>-1.4800e-10</td>
<td>-3.9900e+00</td>
<td>5.9000e-01</td>
</tr>
<tr>
<td></td>
<td>6.3000e+00</td>
<td>-1.7500e-10</td>
<td>4.0100e+00</td>
<td>-4.8000e-01</td>
</tr>
<tr>
<td>3</td>
<td>-7.7000e-01</td>
<td>-1.0600e+10</td>
<td>3.3300e+10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.8300e+00</td>
<td>1.9400e+10</td>
<td>-1.0400e+10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4.4800e+00</td>
<td>-4.6000e-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-1.0900e+00</td>
<td>-1.7200e+00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

rowcnd = 8.9e-11, colcnd = 8.2e-11, amax = 4.4e+10

Row scale factors
2.56e-01  1.25e-01  2.29e-11  1.80e-01

Column scale factors
1.00e+00  1.21e+10  1.00e+00  1.00e+00

Scaled matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.4220</td>
<td>-0.6364</td>
<td>0.2481</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.5780</td>
<td>-0.2639</td>
<td>-0.7263</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.0000</td>
<td>-0.2246</td>
<td>-0.4988</td>
<td>0.0737</td>
</tr>
<tr>
<td></td>
<td>0.7875</td>
<td>-0.2655</td>
<td>0.5012</td>
<td>-0.0600</td>
</tr>
<tr>
<td>3</td>
<td>-0.2139</td>
<td>-0.2426</td>
<td>0.7620</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.7861</td>
<td>0.4439</td>
<td>-0.2380</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.8043</td>
<td>-0.0826</td>
<td>-0.1957</td>
<td>-0.3088</td>
</tr>
</tbody>
</table>