NAG Library Function Document

nag_dgerfs (f07ahc)

1 Purpose

nag_dgerfs (f07ahc) returns error bounds for the solution of a real system of linear equations with multiple right-hand sides, $AX = B$ or $A^TX = B$. It improves the solution by iterative refinement, in order to reduce the backward error as much as possible.

2 Specification

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

void nag_dgerfs (Nag_OrderType order, Nag_TransType trans, Integer n,
                 Integer nrhs, const double a[], Integer pda, const double af[],
                 Integer pdaf, const Integer ipiv[], const double b[], Integer pdb,
                 double x[], Integer pdx, double ferr[], double berr[], NagError *fail)
```

3 Description

nag_dgerfs (f07ahc) returns the backward errors and estimated bounds on the forward errors for the solution of a real system of linear equations with multiple right-hand sides $AX = B$ or $A^TX = B$. The function handles each right-hand side vector (stored as a column of the matrix $B$) independently, so we describe the function of nag_dgerfs (f07ahc) in terms of a single right-hand side $b$ and solution $x$.

Given a computed solution $x$, the function computes the component-wise backward error $\beta$. This is the size of the smallest relative perturbation in each element of $A$ and $b$ such that $x$ is the exact solution of a perturbed system

$$(A + \delta A)x = b + \delta b$$

$$|\delta a_{ij}| \leq \beta|a_{ij}| \quad \text{and} \quad |\delta b_i| \leq \beta|b_i|.$$ 

Then the function estimates a bound for the component-wise forward error in the computed solution, defined by:

$$\max_i |x_i - \hat{x}_i| / \max_i |x_i|$$

where $\hat{x}$ is the true solution.

For details of the method, see the f07 Chapter Introduction.

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.
trans – Nag_TransType

Input

On entry: indicates the form of the linear equations for which \( X \) is the computed solution.

\[
\text{trans} = \text{Nag}_\text{NoTrans}
\]

The linear equations are of the form \( AX = B \).

\[
\text{trans} = \text{Nag}_\text{Trans} \text{ or } \text{Nag}_\text{ConjTrans}
\]

The linear equations are of the form \( A^T X = B \).

Constraint: \( \text{trans} = \text{Nag}_\text{NoTrans}, \text{Nag}_\text{Trans} \text{ or } \text{Nag}_\text{ConjTrans} \).

n – Integer

Input

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

Constraint: \( n \geq 0 \).

nrhs – Integer

Input

On entry: \( r \), the number of right-hand sides.

Constraint: \( nrhs \geq 0 \).

\[
a[\text{dim}] \text{ – const double}
\]

Input

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

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

\[
\begin{align*}
a[(j - 1) \times \text{pda} + i - 1] & \quad \text{when order = Nag}_\text{ColMajor}; \\
a[(i - 1) \times \text{pda} + j - 1] & \quad \text{when order = Nag}_\text{RowMajor}.
\end{align*}
\]

On entry: the \( n \) by \( n \) original matrix \( A \) as supplied to \( \text{Nag}_\text{dgetrf}(f07adc) \).

\[
pda \text{ – Integer}
\]

Input

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

Constraint: \( pda \geq \max(1, n) \).

\[
a[\text{dim}] \text{ – const double}
\]

Input

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

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

\[
\begin{align*}
af[(j - 1) \times \text{pdaf} + i - 1] & \quad \text{when order = Nag}_\text{ColMajor}; \\
af[(i - 1) \times \text{pdaf} + j - 1] & \quad \text{when order = Nag}_\text{RowMajor}.
\end{align*}
\]

On entry: the \( LU \) factorization of \( A \), as returned by \( \text{Nag}_\text{dgetrf}(f07adc) \).

\[
pdaf \text{ – Integer}
\]

Input

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

Constraint: \( pdaf \geq \max(1, n) \).

\[
\text{ipiv}[\text{dim}] \text{ – const Integer}
\]

Input

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

On entry: the pivot indices, as returned by \( \text{Nag}_\text{dgetrf}(f07adc) \).
Note: the dimension, \( dim \), of the array \( b \) must be at least
\[
\max(1, \text{pdb} \times \text{nrhs}) \quad \text{when order} = \text{Nag\_ColMajor};
\]
\[
\max(1, \text{n} \times \text{pdb}) \quad \text{when order} = \text{Nag\_RowMajor}.
\]
The \((i,j)\)th element of the matrix \( B \) is stored in
\[
b[j - 1] \times \text{pdb} + i - 1] \quad \text{when order} = \text{Nag\_ColMajor};
\]
\[
b[i - 1] \times \text{pdb} + j - 1] \quad \text{when order} = \text{Nag\_RowMajor}.
\]

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

Note: the dimension, \( dim \), of the array \( x \) must be at least
\[
\max(1, \text{pdx} \times \text{nrhs}) \quad \text{when order} = \text{Nag\_ColMajor};
\]
\[
\max(1, \text{n} \times \text{pdx}) \quad \text{when order} = \text{Nag\_RowMajor}.
\]
The \((i,j)\)th element of the matrix \( X \) is stored in
\[
x[j - 1] \times \text{pdx} + i - 1] \quad \text{when order} = \text{Nag\_ColMajor};
\]
\[
x[i - 1] \times \text{pdx} + j - 1] \quad \text{when order} = \text{Nag\_RowMajor}.
\]

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

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

On exit: the stride separating row or column elements (depending on the value of \( order \)) in the array \( x \).

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

On exit: \( \text{ferr}[j - 1] \) contains an estimated error bound for the \( j \)th solution vector, that is, the \( j \)th column of \( X \), for \( j = 1, 2, \ldots, r \).

On exit: \( \text{berr}[j - 1] \) contains the component-wise backward error bound \( \beta \) for the \( j \)th solution vector, that is, the \( j \)th column of \( X \), for \( j = 1, 2, \ldots, r \).

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, n = ⟨value⟩.
Constraint: n ≥ 0.

On entry, nrhs = ⟨value⟩.
Constraint: nrhs ≥ 0.

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

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

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

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

**NE_INT_2**

On entry, pda = ⟨value⟩ and n = ⟨value⟩.
Constraint: pda ≥ max(1, n).

On entry, pdaf = ⟨value⟩ and n = ⟨value⟩.
Constraint: pdaf ≥ max(1, n).

On entry, pdb = ⟨value⟩ and n = ⟨value⟩.
Constraint: pdb ≥ max(1, n).

On entry, pdb = ⟨value⟩ and nrhs = ⟨value⟩.
Constraint: pdb ≥ max(1, nrhs).

On entry, pdx = ⟨value⟩ and n = ⟨value⟩.
Constraint: pdx ≥ max(1, n).

On entry, pdx = ⟨value⟩ and nrhs = ⟨value⟩.
Constraint: pdx ≥ 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.
7 Accuracy
The bounds returned in ferr are not rigorous, because they are estimated, not computed exactly; but in practice they almost always overestimate the actual error.

8 Parallelism and Performance
nag_dgerfs (f07ahc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dgerfs (f07ahc) 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
For each right-hand side, computation of the backward error involves a minimum of $4n^2$ floating-point operations. Each step of iterative refinement involves an additional $6n^2$ operations. At most five steps of iterative refinement are performed, but usually only one or two steps are required.

Estimating the forward error involves solving a number of systems of linear equations of the form $Ax = b$ or $A^T x = b$; the number is usually 4 or 5 and never more than 11. Each solution involves approximately $2n^2$ operations.

The complex analogue of this function is nag_zgerfs (f07avc).

10 Example
This example solves the system of equations $AX = B$ using iterative refinement and to compute the forward and backward error bounds, where

$$A = \begin{pmatrix} 1.80 & 2.88 & 2.05 & -0.89 \\ 5.25 & -2.95 & -0.95 & -3.80 \\ 1.58 & -2.69 & -2.90 & -1.04 \\ -1.11 & -0.66 & -0.59 & 0.80 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 9.52 & 18.47 \\ 24.35 & 2.25 \\ 0.77 & -13.28 \\ -6.22 & -6.21 \end{pmatrix}.$$ 

Here $A$ is nonsymmetric and must first be factorized by nag_dgetrf (f07adc).

10.1 Program Text
/* nag_dgerfs (f07ahc) Example Program. */
*/
* Copyright 2014 Numerical Algorithms Group.
*/
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf07.h>
#include <nagx04.h>

int main(void)
{
  /* Scalars */
  Integer    berr_len, i, ipiv_len, ferr_len, j, n, nrhs;
  Integer    pda, pdaf, pdb, pdx;
  Integer    exit_status = 0;
  NagError   fail;

  /* ... */
Nag_OrderType order;

/* Arrays */
double *a = 0, *af = 0, *b = 0, *berr = 0, *ferr = 0, *x = 0;
Integer *ipiv = 0;

#ifndef NAG_COLUMN_MAJOR
#define A(I, J) a[(J-1)*pda + I - 1]
#define AF(I, J) af[(J-1)*pdaf + I - 1]
#define B(I, J) b[(J-1)*pdb + I - 1]
#define X(I, J) x[(J-1)*pdx + I - 1]
#endif

order = Nag_ColMajor;
#else
#define A(I, J) a[(I-1)*pda + J - 1]
#define AF(I, J) af[(I-1)*pdaf + J - 1]
#define B(I, J) b[(I-1)*pdb + J - 1]
#define X(I, J) x[(I-1)*pdx + J - 1]
#endif

INIT_FAIL(fail);

printf("nag_dgerfs (f07ahc) Example Program Results\n\n");

#ifndef _WIN32
scanf("%*[^
] ");
#else
scanf("%*[\n ] ");
#endif

#ifndef _WIN32
scanf("%"NAG_IFMT"%"NAG_IFMT"%*[\n ] ", &n, &nrhs);
#else
scanf("%"NAG_IFMT"%"NAG_IFMT"%*[\n ] ", &n, &nrhs);
#endif

#ifndef NAG_COLUMN_MAJOR
pda = n;
pdaf = n;
pdb = n;
pdx = n;
#else
pda = n;
pdaf = n;
pdb = nrhs;
pdx = nrhs;
#endif
berr_len = nrhs;
ferr_len = nrhs;
ipiv_len = n;

/* Allocate memory */
if (!(a = NAG_ALLOC(n * n, double)) ||
   !(af = NAG_ALLOC(n * n, double)) ||
   !(b = NAG_ALLOC(n * nrhs, double)) ||
   !(berr = NAG_ALLOC(berr_len, double)) ||
   !(ferr = NAG_ALLOC(ferr_len, double)) ||
   !(x = NAG_ALLOC(n * nrhs, double)) ||
   !(ipiv = NAG_ALLOC(ipiv_len, Integer)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}
#endif

/* Read A and B from data file, and copy A to AF and B to X */
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= n; ++j)
#endif

f07ahc
```c
scanf_s("%lf", &A(i, j));
#else
scanf("%lf", &A(i, j));
#endif
}
#ifdef _WIN32
scanf_s("%*[\n ] ");
#else
scanf("%*[\n ] ");
#endif
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
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= n; ++j)
        AF(i, j) = A(i, j);
}
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= nrhs; ++j)
        X(i, j) = B(i, j);
}

/* Factorize A in the array AF */
/* nag_dgetrf (f07adc). *
* LU factorization of real m by n matrix *
*/
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dgetrf (f07adc).\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Compute solution in the array X */
/* nag_dgetrs (f07aec). *
* Solution of real system of linear equations, multiple *
* right-hand sides, matrix already factorized by nag_dgetrf *
* (f07adc) *
*/
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dgetrs (f07aec).\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
/* nag_dgerfs (f07ahc). *
* Refined solution with error bounds of real system of *
* linear equations, multiple right-hand sides *
*/
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dgerfs (f07ahc).\n", fail.message);
    exit_status = 1;
    goto END;
}
```
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_dgerfs (f07ahc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print solution */
/* 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,
                           x, pdx, "Solution(s)", 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;
}

printf("\nBackward errors (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    printf("%11.1e%s", berr[j-1], j%7 == 0?"\n":" ");
printf("\nEstimated forward error bounds (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    printf("%11.1e%s", ferr[j-1], j%7 == 0?"\n":" ");

END:
NAG_FREE(a);
NAG_FREE(af);
NAG_FREE(b);
NAG_FREE(berr);
NAG_FREE(ferr);
NAG_FREE(x);
NAG_FREE(ipiv);
return exit_status;

10.2 Program Data

nag_dgerfs (f07ahc) Example Program Data
4 2 :Values of N and NRHS
 1.80  2.88  2.05 -0.89
 5.25 -2.95 -0.95 -3.80
 1.58 -2.69 -2.90 -1.04
-1.11 -0.66 -0.59  0.80 :End of matrix A
 9.52 18.47
24.35  2.25
 0.77 -13.28
-6.22  -6.21 :End of matrix B

10.3 Program Results

nag_dgerfs (f07ahc) Example Program Results

Solution(s)
     1  2
 1 1.0000 3.0000
 2 -1.0000 2.0000
 3 3.0000 4.0000
 4 -5.0000 1.0000
### Backward errors (machine-dependent)

- 9.4e-17
- 3.7e-17

### Estimated forward error bounds (machine-dependent)

- 2.4e-14
- 3.3e-14