NAG C 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 $A^T X = B$. It improves the solution by iterative refinement, in order to reduce the backward error as much as possible.

2 Specification

```c
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 $A^T X = B$. 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:

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

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

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

4 References


5 Parameters

1: order – Nag_OrderType

On entry: the order parameter 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 2.2.1.4 of the Essential Introduction for a more detailed explanation of the use of this parameter.

Constraint: order = Nag_RowMajor or Nag_ColMajor.

2: trans – Nag_TransType

On entry: indicates the form of the linear equations for which $X$ is the computed solution as follows:

[NP3645/7]
if \( \text{trans} = \text{Nag\_NoTrans} \), then the linear equations are of the form \( AX = B \);
if \( \text{trans} = \text{Nag\_Trans} \) or \( \text{Nag\_ConjTrans} \), then the linear equations are of the form \( A^T X = B \).

Constraint: \( \text{trans} = \text{Nag\_NoTrans}, \text{Nag\_Trans} \) or \( \text{Nag\_ConjTrans} \).

3: \( n \) – Integer  
\text{Input}  
\text{On entry: } n, \text{ the order of the matrix } A.  
\text{Constraint: } n \geq 0.

4: \( \text{nrhs} \) – Integer  
\text{Input}  
\text{On entry: } r, \text{ the number of right-hand sides.}  
\text{Constraint: } \text{nrhs} \geq 0.

5: \( a[dim] \) – const double  
\text{Input}  
\text{Note: } \text{the dimension, } dim, \text{ of the array } a \text{ must be at least } \max(1, \text{pda} \times n).

If \( \text{order} = \text{Nag\_ColMajor} \), the \((i,j)\)th element of the matrix \( A \) is stored in \( a[(j-1) \times \text{pda} + i - 1] \) and if \( \text{order} = \text{Nag\_RowMajor} \), the \((i,j)\)th element of the matrix \( A \) is stored in \( a[(i-1) \times \text{pda} + j - 1] \).
\text{On entry: } \text{the } n \text{ by } n \text{ original matrix } A \text{ as supplied to nag_dgetrf (f07adc)}.

6: \( \text{pda} \) – Integer  
\text{Input}  
\text{On entry: } \text{the stride separating matrix row or column elements (depending on the value of order) in the array } a.  
\text{Constraint: } \text{pda} \geq \max(1, n).

7: \( af[dim] \) – const double  
\text{Input}  
\text{Note: } \text{the dimension, } dim, \text{ of the array } af \text{ must be at least } \max(1, \text{pdaf} \times n).

If \( \text{order} = \text{Nag\_ColMajor} \), the \((i,j)\)th element of the matrix is stored in \( af[(j-1) \times \text{pdaf} + i - 1] \) and if \( \text{order} = \text{Nag\_RowMajor} \), the \((i,j)\)th element of the matrix is stored in \( af[(i-1) \times \text{pdaf} + j - 1] \).
\text{On entry: } \text{the } LU \text{ factorization of } A, \text{ as returned by nag_dgetrf (f07adc)}.

8: \( \text{pdaf} \) – Integer  
\text{Input}  
\text{On entry: } \text{the stride separating matrix row or column elements (depending on the value of order) in the array } af.  
\text{Constraint: } \text{pdaf} \geq \max(1, n).

9: \( \text{ipiv[dim]} \) – const Integer  
\text{Input}  
\text{Note: } \text{the dimension, } dim, \text{ of the array } ipiv \text{ must be at least } \max(1, n).
\text{On entry: } \text{the pivot indices, as returned by nag_dgetrf (f07adc)}.

10: \( b[dim] \) – const double  
\text{Input}  
\text{Note: } \text{the dimension, } dim, \text{ of the array } b \text{ must be at least } \max(1, \text{pdb} \times \text{nrhs}) \text{ when } \text{order} = \text{Nag\_ColMajor} \text{ and at least } \max(1, \text{pdb} \times n) \text{ when } \text{order} = \text{Nag\_RowMajor}.  
\text{If } \text{order} = \text{Nag\_ColMajor}, \text{ the } (i,j)\text{th element of the matrix } B \text{ is stored in } b[(j-1) \times \text{pdb} + i - 1] \text{ and if } \text{order} = \text{Nag\_RowMajor}, \text{ the } (i,j)\text{th element of the matrix } B \text{ is stored in } b[(i-1) \times \text{pdb} + j - 1].  
\text{On entry: } \text{the } n \text{ by } r \text{ right-hand side matrix } B.
11: \( \text{pdb} \) – Integer

\textit{Input}

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

\textit{Constraints:}

\begin{align*}
\text{if } \text{order} &= \text{Nag\_ColMajor}, \text{pdb} \geq \max(1, n); \\
\text{if } \text{order} &= \text{Nag\_RowMajor}, \text{pdb} \geq \max(1, \text{nrhs}).
\end{align*}

12: \( x[\text{dim}] \) – double

\textit{Input/Output}

\textit{Note:} the dimension, \( \text{dim} \), of the array \( x \) must be at least \( \max(1, \text{pdx} \times \text{nrhs}) \) when \( \text{order} = \text{Nag\_ColMajor} \) and at least \( \max(1, \text{pdx} \times n) \) when \( \text{order} = \text{Nag\_RowMajor} \).

\textit{On entry:} the \( n \) by \( r \) solution matrix \( X \), as returned by \text{nag\_dgetrs} (f07aec).

\textit{On exit:} the improved solution matrix \( X \).

13: \( \text{pdx} \) – Integer

\textit{Input}

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

\textit{Constraints:}

\begin{align*}
\text{if } \text{order} &= \text{Nag\_ColMajor}, \text{pdx} \geq \max(1, n); \\
\text{if } \text{order} &= \text{Nag\_RowMajor}, \text{pdx} \geq \max(1, \text{nrhs}).
\end{align*}

14: \( \text{ferr}[\text{dim}] \) – double

\textit{Output}

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

\textit{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 \( j = 1, 2, \ldots, r \).

15: \( \text{berr}[\text{dim}] \) – double

\textit{Output}

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

\textit{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 \( j = 1, 2, \ldots, r \).

16: \( \text{fail} \) – NagError *

\textit{Output}

The NAGError * parameter (see the Essential Introduction).

\section{Error Indicators and Warnings}

\textbf{NE\_INT}

\begin{itemize}
\item On entry, \( n = \langle \text{value} \rangle \).
\item Constraint: \( n \geq 0 \).
\item On entry, \( \text{nrhs} = \langle \text{value} \rangle \).
\item Constraint: \( \text{nrhs} \geq 0 \).
\item On entry, \( \text{pda} = \langle \text{value} \rangle \).
\item Constraint: \( \text{pda} > 0 \).
\item On entry, \( \text{pdaf} = \langle \text{value} \rangle \).
\item Constraint: \( \text{pdaf} > 0 \).
\item On entry, \( \text{pdb} = \langle \text{value} \rangle \).
\item Constraint: \( \text{pdb} > 0 \).
\end{itemize}
On entry, \( \text{pdx} = \langle \text{value} \rangle \).
Constraint: \( \text{pdx} > 0 \).

**NE_INT_2**

On entry, \( \text{pda} = \langle \text{value} \rangle \), \( n = \langle \text{value} \rangle \).
Constraint: \( \text{pda} \geq \max(1, n) \).

On entry, \( \text{pdaf} = \langle \text{value} \rangle \), \( n = \langle \text{value} \rangle \).
Constraint: \( \text{pdaf} \geq \max(1, n) \).

On entry, \( \text{pdb} = \langle \text{value} \rangle \), \( n = \langle \text{value} \rangle \).
Constraint: \( \text{pdb} \geq \max(1, n) \).

On entry, \( \text{pdb} = \langle \text{value} \rangle \), \( \text{nrhs} = \langle \text{value} \rangle \).
Constraint: \( \text{pdb} \geq \max(1, \text{nrhs}) \).

On entry, \( \text{pdx} = \langle \text{value} \rangle \), \( n = \langle \text{value} \rangle \).
Constraint: \( \text{pdx} \geq \max(1, n) \).

On entry, \( \text{pdx} = \langle \text{value} \rangle \), \( \text{nrhs} = \langle \text{value} \rangle \).
Constraint: \( \text{pdx} \geq \max(1, \text{nrhs}) \).

**NE_ALLOC_FAIL**

Memory allocation failed.

**NE_BAD_PARAM**

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

**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 consult NAG for assistance.

7 Accuracy

The bounds returned in \( \text{ferr} \) are not rigorous, because they are estimated, not computed exactly; but in practice they almost always overestimate the actual error.

8 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 5 steps of iterative refinement are performed, but usually only 1 or 2 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).

9 Example

To solve the system of equations \( AX = B \) using iterative refinement and to compute the forward and backward error bounds, where

\[
A = \begin{pmatrix}
1.80 & 2.88 & 2.05 & -0.89 \\
5.25 & -2.95 & -0.95 & -3.80 \\
1.58 & -2.69 & -2.90 & -1.04 \\
-1.11 & -0.66 & -0.59 & 0.80
\end{pmatrix}
\quad \text{and} \quad
B = \begin{pmatrix}
9.52 & 18.47 \\
24.35 & 22.05 \\
0.77 & -13.28 \\
-6.22 & -6.21
\end{pmatrix}.
\]

Here \( A \) is nonsymmetric and must first be factorized by nag_dgetrf (f07adc).
#include <stdio.h>
#include <nag.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;

    INIT_FAIL(fail);
    Vprintf("f07ahc Example Program Results

");

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

    #ifdef 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(n, Integer)) )
    {
        exit_status = 1;
        Vprintf("Memory allocation failed.

");
    }

    order = Nag_RowMajor;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            A(i,j) = (i == j) ? 1 : 0;
        }
    }

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            AF(i,j) = (i == j) ? 1 : 0;
        }
    }

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < nrhs; j++)
        {
            B(i,j) = (i == j) ? 1 : 0;
        }
    }

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < nrhs; j++)
        {
            X(i,j) = (i == j) ? 1 : 0;
        }
    }

    for (i = 0; i < n; i++)
    {
        ipiv[i] = (i + 1);
    }

    status = nag_dgerfs(n, nrhs, af, pda, b, pdb, pdx, ferr, berr, &ferr);
}

9.1 Program Text
/* nag_dgerfs (f07ahc) Example Program.
  * Copyright 2001 Numerical Algorithms Group.
  */

#include <stdio.h>
#include <nag.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;

    INIT_FAIL(fail);
    Vprintf("f07ahc Example Program Results

");

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

    #ifdef 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(n, Integer)) )
    {
        exit_status = 1;
        Vprintf("Memory allocation failed.

");
    }

    order = Nag_RowMajor;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            A(i,j) = (i == j) ? 1 : 0;
        }
    }

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            AF(i,j) = (i == j) ? 1 : 0;
        }
    }

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < nrhs; j++)
        {
            B(i,j) = (i == j) ? 1 : 0;
        }
    }

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < nrhs; j++)
        {
            X(i,j) = (i == j) ? 1 : 0;
        }
    }

    for (i = 0; i < n; i++)
    {
        ipiv[i] = (i + 1);
    }

    status = nag_dgerfs(n, nrhs, af, pda, b, pdb, pdx, ferr, berr, &ferr);
}
!(x = NAG_ALLOC(n * nrhs, double)) ||
!(ipiv = NAG_ALLOC(ipiv_len, Integer)) )
{
  Vprintf("Allocation failure\n");
  exit_status = -1;
  goto END;
}

/* 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)
    Vscanf("%lf", &A(i,j));
}
Vscanf("%*[\n ]");
for (i = 1; i <= n; ++i)
{
  for (j = 1; j <= nrhs; ++j)
    Vscanf("%lf", &B(i,j));
}
Vscanf("%*[\n ]");
for (i = 1; i <= n; ++i)
{
  for (j = 1; j <= n; ++j)
    AF(i,j) = A(i,j);
  for (j = 1; j <= nrhs; ++j)
    X(i,j) = B(i,j);
}

/* Factorize A in the array AF */
f07adc(order, n, n, af, pdaf, ipiv, &fail);
if (fail.code != NE_NOERROR)
{
  Vprintf("Error from f07adc.\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}
Vprintf("\n");
/* Compute solution in the array X */
f07aec(order, Nag_NoTrans, n, nrhs, af, pdaf, ipiv, x, pdx,
&fail);
if (fail.code != NE_NOERROR)
{
  Vprintf("Error from f07aec.\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}
/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
f07ahc(order, Nag_NoTrans, n, nrhs, a, pda, af, pdaf, ipiv, b, pdb, x,
pdx, ferr, berr, &fail);
if (fail.code != NE_NOERROR)
{
  Vprintf("Error from f07ahc.\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}
/* Print solution */
x04cac(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, x, pdx,
"Solution(s)" , 0, &fail);
if (fail.code != NE_NOERROR)
{
  Vprintf("Error from x04cac.\n%s\n", fail.message);
  exit_status = 1;
  goto END;
Vprintf("\nBackward errors (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
  Vprintf("%11.1e%s", berr[j-1], j%7==0 ?"\n":" ");
Vprintf("\nEstimated forward error bounds (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
  Vprintf("%11.1e%s", ferr[j-1], j%7==0 ?"\n":" ");
Vprintf("\n");
END:
  if (a) NAG_FREE(a);
  if (af) NAG_FREE(af);
  if (b) NAG_FREE(b);
  if (berr) NAG_FREE(berr);
  if (ferr) NAG_FREE(ferr);
  if (x) NAG_FREE(x);
  if (ipiv) NAG_FREE(ipiv);
return exit_status;

9.2 Program Data

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

9.3 Program Results

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)
  5.6e-17  6.2e-17
Estimated forward error bounds (machine-dependent)
  2.4e-14  3.3e-14