NAG C Library Function Document

nag_zgerfs (f07avc)

1 Purpose

nag_zgerfs (f07avc) returns error bounds for the solution of a complex system of linear equations with multiple right-hand sides, \( AX = B \), \( A^T X = B \) or \( A^H X = B \). It improves the solution by iterative refinement, in order to reduce the backward error as much as possible.

2 Specification

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

3 Description

nag_zgerfs (f07avc) returns the backward errors and estimated bounds on the forward errors for the solution of a complex system of linear equations with multiple right-hand sides \( AX = B \), \( A^T X = B \) or \( A^H X = B \). The function handles each right-hand side vector (stored as a column of the matrix \( B \)) independently, so we describe the function of nag_zgerfs (f07avc) 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 Parameters

1: order – Nag_OrderType

*Input*

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

- if \( \text{trans} = \text{Nag\_NoTrans} \), the linear equations are of the form \( AX = B \);
- if \( \text{trans} = \text{Nag\_Trans} \), the linear equations are of the form \( A^T X = B \);
- if \( \text{trans} = \text{Nag\_ConjTrans} \), the linear equations are of the form \( A^H X = B \).

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

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

Constraint: \( n \geq 0 \).

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

Constraint: \( nrhs \geq 0 \).

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

On entry: the \( n \) by \( n \) original matrix \( A \) as supplied to nag_zgetrf (f07arc).

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

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

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

On entry: the \( LU \) factorization of \( A \), as returned by nag_zgetrf (f07arc).

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

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

On entry: the pivot indices, as returned by nag_zgetrf (f07arc).

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

On entry: the \( i, j \)th element of the matrix \( B \) is stored in \( b[(j - 1) \times pdb + i - 1] \) and

Constraint: \( \text{trans} = \text{Nag\_NoTrans}, \text{Nag\_Trans} \) or \( \text{Nag\_ConjTrans} \).
On entry: the \( n \) by \( r \) right-hand side matrix \( B \).

11: \( \text{pdb} \) – Integer

*Input*

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

*Constraints*:

if \( \text{order} = \text{Nag\_ColMajor} \), \( \text{pdb} \geq \max(1, n) \);

if \( \text{order} = \text{Nag\_RowMajor} \), \( \text{pdb} \geq \max(1, \text{nrhs}) \).

12: \( \text{x}[\text{dim}] \) – Complex

*Input/Output*

Note: the dimension, \( \text{dim} \), of the array \( \text{x} \) 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 \( \text{order} = \text{Nag\_RowMajor} \).

If \( \text{order} = \text{Nag\_ColMajor} \), the \((i, j)\)th element of the matrix \( X \) is stored in \( \text{x}[((j - 1) \times \text{pdx} + i - 1)] \) and if \( \text{order} = \text{Nag\_RowMajor} \), the \((i, j)\)th element of the matrix \( X \) is stored in \( \text{x}[(i - 1) \times \text{pdx} + j - 1] \).

On entry: the \( n \) by \( r \) solution matrix \( X \), as returned by \text{nag\_zgetrs} (f07asc).

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

13: \( \text{pdx} \) – Integer

*Input*

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

*Constraints*:

if \( \text{order} = \text{Nag\_ColMajor} \), \( \text{pdx} \geq \max(1, n) \);

if \( \text{order} = \text{Nag\_RowMajor} \), \( \text{pdx} \geq \max(1, \text{nrhs}) \).

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

*Output*

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

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

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

*Output*

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

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*

*Output*

The NAG error parameter (see the Essential Introduction).

6 Error Indicators and Warnings

**NE\(_\text{\_\_INT}\)**

On entry, \( n = \langle \text{value} \rangle \).

Constraint: \( n \geq 0 \).

On entry, \( \text{nrhs} = \langle \text{value} \rangle \).

Constraint: \( \text{nrhs} \geq 0 \).

On entry, \( \text{pda} = \langle \text{value} \rangle \).

Constraint: \( \text{pda} > 0 \).

On entry, \( \text{pdaf} = \langle \text{value} \rangle \).

Constraint: \( \text{pdaf} > 0 \).
On entry, \( \text{pdb} = \langle \text{value} \rangle \).
Constraint: \( \text{pdb} > 0 \).

On entry, \( \text{pdx} = \langle \text{value} \rangle \).
Constraint: \( \text{pdx} > 0 \).

\textbf{NE_INT_2}

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

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

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

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

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

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

\textbf{NE_ALLOC_FAIL}

Memory allocation failed.

\textbf{NE_BAD_PARAM}

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

\textbf{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 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 \( 16n^2 \) real floating-point operations. Each step of iterative refinement involves an additional \( 24n^2 \) real 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 \( A^H x = b \); the number is usually 5 and never more than 11. Each solution involves approximately \( 8n^2 \) real operations.

The real analogue of this function is \( \text{nag_dgerfs (f07ahc)} \).

9 Example

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

\[ AX = B \]

NAG C Library Manual
Here $A$ is nonsymmetric and must first be factorized by nag_zgetrf (f07arc).

9.1 Program Text

/* nag_zgerfs (f07avc) Example Program. */
/* Copyright 2001 Numerical Algorithms Group. */
/* Mark 7, 2001. */
*/
#include <stdio.h>
#include <nag.h>
#include <nagf07.h>
#include <nagx04.h>

int main(void)
{
    /* Scalars */
    Integer berr_len, i, ferr_len, ipiv_len, j, n, nrhs;
    Integer pda, pdaf, pdb, pdx;
    Integer exit_status=0;
    NagError fail;
    Nag_OrderType order;
    /* Arrays */
    Complex *a=0, *af=0, *b=0, *x=0;
    double *berr=0, *ferr=0;
    Integer *ipiv=0;

    #ifdef 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]
    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[(J-1)*pdb+J-1]
    #define X(I,J) x[(I-1)*pdx+J-1]
    order = Nag_RowMajor;
    #endif

    INIT_FAIL(fail);
    Vprintf("f07avc Example Program Results\n\n");

    /* 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;
    #endif

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

/* Allocate memory */
if ( !(a = NAG_ALLOC(n * n, Complex)) ||
    !(af = NAG_ALLOC(n * n, Complex)) ||
    !(b = NAG_ALLOC(n * nrhs, Complex)) ||
    !(x = NAG_ALLOC(n * n, Complex)) ||
    !(berr = NAG_ALLOC(berr_len, double)) ||
    !(ferr = NAG_ALLOC(ferr_len, 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 , %lf )", &A(i,j).re, &A(i,j).im);
    Vscanf("%*\[\n\]");
}
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= nrhs; ++j)
        Vscanf(" ( %lf , %lf )", &B(i,j).re, &B(i,j).im);
    Vscanf("%*\[\n\]");
}

for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= n; ++j)
    {
        AF(i,j).re = A(i,j).re;
        AF(i,j).im = A(i,j).im;
    }
}
for (i = 1; i <= n; ++i)
{
    for (j = 1; j <= nrhs; ++j)
    {
        X(i,j).re = B(i,j).re;
        X(i,j).im = B(i,j).im;
    }
}

/* Factorize A in the array AF */
f07arc(order, n, n, af, pdaf, ipiv, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07arc.\n\", fail.message);
    exit_status = 1;
    goto END;
}

/* Compute solution in the array X */
f07asc(order, Nag_NoTrans, n, nrhs, af, pdaf, ipiv, x, pdx, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from f07asc.\n\", fail.message);
    exit_status = 1;
    goto END;
}

/* Improve solution, and compute backward errors and */
/* estimated bounds on the forward errors */
f07avc(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 f07avc.\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Print solution */
x04dbc(order, Nag_GeneralMatrix, Nag_NonUnitDiag, n, nrhs, x, pdx,
    Nag_BracketForm, "%7.4f", "Solution(s)", Nag_IntegerLabels,
    0, Nag_IntegerLabels, 0, 80, 0, &fail);
if (fail.code != NE_NOERROR)
{
    Vprintf("Error from x04dbc.\n", fail.message);
    exit_status = 1;
    goto END;
}

Vprintf("Backward errors (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    Vprintf("%11.1e\n", berr[j-1]);
Vprintf("Estimated forward error bounds (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j)
    Vprintf("%11.1e\n", ferr[j-1]);

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

9.2 Program Data

f07avc Example Program Data

4 2
(-1.34, 2.55) ( 0.28, 3.17) (-6.39,-2.20) ( 0.72,-0.92)
(-0.17,-1.41) ( 3.31,-0.15) (-0.15, 1.34) ( 1.29, 1.38)
(-3.29,-2.39) (-1.91, 4.42) (-0.14,-1.35) ( 1.72, 1.35)
 ( 2.41, 0.39) (-0.56, 1.47) (-0.83,-0.69) (-1.96, 0.67)
(26.26, 51.78) (31.32, -6.70)
( 6.43, -8.68) (15.86, -1.42)
(-5.75, 25.31) (-2.15, 30.19)
( 1.16, 2.57) (-2.56, 7.55)

9.3 Program Results

f07avc Example Program Results

Solution(s)

1 2
1 ( 1.0000, 1.0000) (-1.0000,-2.0000)
2 ( 2.0000,-3.0000) ( 5.0000, 1.0000)
3 (-4.0000,-5.0000) (-3.0000, 4.0000)
4 ( 0.0000, 6.0000) ( 2.0000,-3.0000)

Backward errors (machine-dependent)
9.1e-17 7.2e-17
Estimated forward error bounds (machine-dependent)
5.9e-14 7.6e-14