1 Purpose

nag_zgesvx (f07apc) uses the LU factorization to compute the solution to a complex system of linear equations

\[ AX = B \quad \text{or} \quad A^TX = B \quad \text{or} \quad A^H X = B, \]

where \( A \) is an \( n \) by \( n \) matrix and \( X \) and \( B \) are \( n \) by \( r \) matrices. Error bounds on the solution and a condition estimate are also provided.

2 Specification

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

void nag_zgesvx (Nag_OrderType order, Nag_FactoredFormType fact,
                  Nag_TransType trans, Integer n, Integer nrhs, Complex a[], Integer pda,
                  Complex af[], Integer pdaf, Integer ipiv[],
                  Nag_EquilibrationType *equed, double r[], double c[], Complex b[],
                  Integer pdb, Complex x[], Integer pdx, double *rcond, double ferr[],
                  double berr[], double *recip_growth_factor, NagError *fail)
```

3 Description

nag_zgesvx (f07apc) performs the following steps:

1. \textbf{Equilibration}

   The linear system to be solved may be badly scaled. However, the system can be equilibrated as a first stage by setting `fact = Nag_EquilibrateAndFactor`. In this case, real scaling factors are computed and these factors then determine whether the system is to be equilibrated. Equilibrated forms of the systems \( AX = B \), \( A^TX = B \) and \( A^H X = B \) are

   \[
   (D_R^AD_C)(D_C^{-1}X) = D_RB, \\
   (D_R^AD_C)^T(D_R^{-1}X) = D_CB, \\
   (D_R^AD_C)^H(D_R^{-1}X) = D_CB,
   \]

   respectively, where \( D_R \) and \( D_C \) are diagonal matrices, with positive diagonal elements, formed from the computed scaling factors.

   When equilibration is used, \( A \) will be over written by \( D_R^AD_C \) and \( B \) will be over written by \( D_RB \) (or \( D_CB \) when the solution of \( A^TX = B \) or \( A^H X = B \) is sought).

2. \textbf{Factorization}

   The matrix \( A \), or its scaled form, is copied and factored using the LU decomposition

   \[ A = PLU, \]

   where \( P \) is a permutation matrix, \( L \) is a unit lower triangular matrix, and \( U \) is upper triangular.
This stage can be by-passed when a factored matrix (with scaled matrices and scaling factors) are supplied; for example, as provided by a previous call to nag_zgesvx (f07apc) with the same matrix A.

3. **Condition Number Estimation**

The $LU$ factorization of $A$ determines whether a solution to the linear system exists. If some diagonal element of $U$ is zero, then $U$ is exactly singular, no solution exists and the function returns with a failure. Otherwise the factorized form of $A$ is used to estimate the condition number of the matrix $A$. If the reciprocal of the condition number is less than *machine precision* then a warning code is returned on final exit.

4. **Solution**

The (equilibrated) system is solved for $X$ ($D_C^{-1}X$ or $D_R^{-1}X$) using the factored form of $A$ ($D_R A D_C$).

5. **Iterative Refinement**

Iterative refinement is applied to improve the computed solution matrix and to calculate error bounds and backward error estimates for the computed solution.

6. **Construct Solution Matrix $X$**

If equilibration was used, the matrix $X$ is premultiplied by $D_C$ (if $\text{trans} = \text{Nag\_NoTrans}$) or $D_R$ (if $\text{trans} = \text{Nag\_Trans}$ or $\text{Nag\_ConjTrans}$) so that it solves the original system before equilibration.

### References


### 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 $\text{order} = \text{Nag\_RowMajor}$. See Section 3.2.1.3 in the Essential Introduction for a more detailed explanation of the use of this argument.

   *Constraint:* $\text{order} = \text{Nag\_RowMajor}$ or $\text{Nag\_ColMajor}$.

2. **fact** – Nag_FactoredFormType

   *Input*

   On entry: specifies whether or not the factorized form of the matrix $A$ is supplied on entry, and if not, whether the matrix $A$ should be equilibrated before it is factorized.

   $\text{fact} = \text{Nag\_Factored}$

   $\text{af}$ and $\text{ipiv}$ contain the factorized form of $A$. If $\text{equed} \neq \text{Nag\_NoEquilibration}$, the matrix $A$ has been equilibrated with scaling factors given by $\text{r}$ and $\text{c}$, $\text{a}$, $\text{af}$ and $\text{ipiv}$ are not modified.

   $\text{fact} = \text{Nag\_NotFactored}$

   The matrix $A$ will be copied to $\text{af}$ and factorized.

   $\text{fact} = \text{Nag\_EquilibrateAndFactor}$

   The matrix $A$ will be equilibrated if necessary, then copied to $\text{af}$ and factorized.

   *Constraint:* $\text{fact} = \text{Nag\_Factored}$, $\text{Nag\_NotFactored}$ or $\text{Nag\_EquilibrateAndFactor}$. 
trans – Nag_TransType

On entry: specifies the form of the system of equations.

trans = Nag_NoTrans
AX = B (No transpose).

trans = Nag_Trans
A^T X = B (Transpose).

trans = Nag_ConjTrans
A^H X = B (Conjugate transpose).

Constraint: trans = Nag_NoTrans, Nag_Trans or Nag_ConjTrans.

n – Integer

On entry: n, the number of linear equations, i.e., the order of the matrix A.

Constraint: n ≥ 0.

nrhs – Integer

On entry: r, the number of right-hand sides, i.e., the number of columns of the matrix B.

Constraint: nrhs ≥ 0.

a[dim] – Complex

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

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

a[(j - 1) × pda + i - 1] when order = Nag_ColMajor;

a[(i - 1) × pda + j - 1] when order = Nag_RowMajor.

On entry: the n by n matrix A.

If fact = Nag_Factored and equed ≠ Nag_NoEquilibration, a must have been equilibrated by the scaling factors in r and/or c.

On exit: if fact = Nag_Factored or Nag_NotFactored, or if fact = Nag_EquilibrateAndFactor and equed = Nag_NoEquilibration, a is not modified.

If fact = Nag_EquilibrateAndFactor or equed ≠ Nag_NoEquilibration, A is scaled as follows:

if equed = Nag_RowEquilibration, A = DRA;

if equed = Nag_ColumnEquilibration, A = ADC;

if equed = Nag_RowAndColumnEquilibration, A = DRAADC.

pda – Integer

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

Constraint: pda ≥ max(1, n).

af[dim] – Complex

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

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

af[(j - 1) × pdaf + i - 1] when order = Nag_ColMajor;

af[(i - 1) × pdaf + j - 1] when order = Nag_RowMajor.

On entry: if fact = Nag_Factored, af contains the factors L and U from the factorization A = PLU as computed by nag_zgetrf (f07arc). If equed ≠ Nag_NoEquilibration, af is the factorized form of the equilibrated matrix A.
If fact = Nag_NotFactored or Nag_EquilibrateAndFactor, af need not be set.

On exit: if fact = Nag_NotFactored, af returns the factors $L$ and $U$ from the factorization $A = PLU$ of the original matrix $A$.

If fact = Nag_EquilibrateAndFactor, af returns the factors $L$ and $U$ from the factorization $A = PLU$ of the equilibrated matrix $A$ (see the description of a for the form of the equilibrated matrix).

If fact = Nag_Factored, af is unchanged from entry.

9: pdaf – Integer

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

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

10: ipiv[dim] – Integer

On entry: if fact = Nag_Factored, ipiv contains the pivot indices from the factorization $A = PLU$ as computed by nag_zgetrf (f07arc); at the $i$th step row $i$ of the matrix was interchanged with row $ipiv[i-1]$. $ipiv[i-1] = i$ indicates a row interchange was not required.

If fact = Nag_NotFactored or Nag_EquilbrateAndFactor, ipiv need not be set.

On exit: if fact = Nag_NotFactored, ipiv contains the pivot indices from the factorization $A = PLU$ of the original matrix $A$.

If fact = Nag_EquilbrateAndFactor, ipiv contains the pivot indices from the factorization $A = PLU$ of the equilibrated matrix $A$.

If fact = Nag_Factored, ipiv is unchanged from entry.

11: equed – Nag_EquilibrationType *

On entry: if fact = Nag_NotFactored or Nag_EquilbrateAndFactor, equed need not be set.

If fact = Nag_Factored, equed must specify the form of the equilibration that was performed as follows:

- if equed = Nag_NoEquilibration, no equilibration;
- if equed = Nag_RowEquilibration, row equilibration, i.e., $A$ has been premultiplied by $D_R$;
- if equed = Nag_ColumnEquilibration, column equilibration, i.e., $A$ has been postmultiplied by $D_C$;
- if equed = Nag_RowAndColumnEquilibration, both row and column equilibration, i.e., $A$ has been replaced by $D_R D_C$.

On exit: if fact = Nag_Factored, equed is unchanged from entry.

Otherwise, if no constraints are violated, equed specifies the form of equilibration that was performed as specified above.

Constraint: if fact = Nag_Factored, equed = Nag_NoEquilibration, Nag_RowEquilibration, Nag_ColumnEquilibration or Nag_RowAndColumnEquilibration.

12: r[dim] – double

Note: the dimension, dim, of the array r must be at least max(1, n).

On entry: if fact = Nag_NotFactored or Nag_EquilbrateAndFactor, r need not be set.

If fact = Nag_Factored and equed = Nag_RowEquilibration or Nag_RowAndColumnEquilibration, r must contain the row scale factors for $A$, $D_R$; each element of r must be positive.
On exit: if fact = Nag_Factored, r is unchanged from entry.
Otherwise, if no constraints are violated and equed = Nag_RowEquilibration or Nag_RowAndColumnEquilibration, r contains the row scale factors for A, \( D_R \), such that A is multiplied on the left by \( D_R \); each element of r is positive.

13: \( c[dim] \) – double

**Input/Output**

**Note:** the dimension, \( dim \), of the array \( c \) must be at least \( \max(1, n) \).

On entry: if fact = Nag_NotFactored or Nag_EquilibrateAndFactor, \( c \) need not be set.
If fact = Nag_Factored or equed = Nag_ColumnEquilibration or Nag_RowAndColumnEquilibration, \( c \) must contain the column scale factors for \( A \), \( D_C \); each element of \( c \) is positive.

On exit: if fact = Nag_Factored, \( c \) is unchanged from entry.
Otherwise, if no constraints are violated and equed = Nag_ColumnEquilibration or Nag_RowAndColumnEquilibration, \( c \) contains the row scale factors for \( A \), \( D_C \); each element of \( c \) is positive.

14: \( b[dim] \) – Complex

**Input/Output**

**Note:** the dimension, \( dim \), of the array \( b \) must be at least \( \max(1, \text{pdb} \times \text{nrhs}) \) when order = Nag_ColMajor;
\( \max(1, n \times \text{pdb}) \) when order = Nag_RowMajor.

The \( (i, j) \)th element of the matrix \( B \) is stored in
\[
\begin{align*}
  b[(j-1) \times \text{pdb} + i - 1] & \quad \text{when order = Nag_ColMajor;} \\
  b[(i-1) \times \text{pdb} + j - 1] & \quad \text{when order = Nag_RowMajor.}
\end{align*}
\]

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

On exit: if equed = Nag_NoEquilibration, \( b \) is not modified.
If trans = Nag_NoTrans and equed = Nag_RowEquilibration or Nag_RowAndColumnEquilibration, \( b \) is overwritten by \( D_R B \).
If trans = Nag_Trans or Nag_ConjTrans and equed = Nag_ColumnEquilibration or Nag_RowAndColumnEquilibration, \( b \) is overwritten by \( D_C B \).

15: \( \text{pdb} \) – Integer

**Input**

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

Constraints:
\[
\begin{align*}
  \text{if order = Nag_ColMajor, } & \quad \text{pdb} \geq \max(1, n) ; \\
  \text{if order = Nag_RowMajor, } & \quad \text{pdb} \geq \max(1, \text{nrhs}) .
\end{align*}
\]

16: \( x[dim] \) – Complex

**Output**

**Note:** the dimension, \( dim \), of the array \( x \) must be at least \( \max(1, \text{pdx} \times \text{nrhs}) \) when order = Nag_ColMajor;
\( \max(1, n \times \text{pdx}) \) when order = Nag_RowMajor.

The \( (i, j) \)th element of the matrix \( X \) is stored in
\[
\begin{align*}
  x[(j-1) \times \text{pdx} + i - 1] & \quad \text{when order = Nag_ColMajor;} \\
  x[(i-1) \times \text{pdx} + j - 1] & \quad \text{when order = Nag_RowMajor.}
\end{align*}
\]

On exit: if fail code = NE_NOERROR or NE_SINGULAR_WP, the \( n \) by \( r \) solution matrix \( X \) to the original system of equations. Note that the arrays \( A \) and \( B \) are modified on exit if equed \( \neq \) Nag_NoEquilibration, and the solution to the equilibrated system is \( D_C^{-1} X \) if
trans = Nag_NoTrans and equed = Nag_ColumnEquilibration or Nag_RowAndColumnEquilibration, or \( D^{-1} X \) if trans = Nag_Trans or Nag_ConjTrans and equed = Nag_RowEquilibration or Nag_RowAndColumnEquilibration.

17: pdx – Integer  
   Input
   On entry: the stride separating row or column elements (depending on the value of order) in the array x.
   Constraints:
   if order = Nag_ColMajor, pdx \geq \max(1, n);
   if order = Nag_RowMajor, pdx \geq \max(1, nrhs).

18: rcond – double *  
   Output
   On exit: if no constraints are violated, an estimate of the reciprocal condition number of the matrix A (after equilibration if that is performed), computed as \( rcond = 1 / (\|A\|_1 / \|A^{-1}\|_1) \).

19: ferr[nrhs] – double  
   Output
   On exit: if fail.code = NE_NOERROR or NE_SINGULAR_WP, an estimate of the forward error bound for each computed solution vector, such that \( \|\hat{x}_j - x_j\|_\infty / \|x_j\|_\infty \leq ferr[j - 1] \) where \( \hat{x}_j \) is the \( j \)th column of the computed solution returned in the array x and \( x_j \) is the corresponding column of the exact solution X. The estimate is as reliable as the estimate for rcond, and is almost always a slight overestimate of the true error.

20: berr[nrhs] – double  
   Output
   On exit: if fail.code = NE_NOERROR or NE_SINGULAR_WP, an estimate of the component-wise relative backward error of each computed solution vector \( \hat{x}_j \) (i.e., the smallest relative change in any element of A or B that makes \( \hat{x}_j \) an exact solution).

21: recip_growth_factor – double *  
   Output
   On exit: if fail.code = NE_NOERROR, the reciprocal pivot growth factor \( \|A\| / \|U\| \), where \( \|.\| \) denotes the maximum absolute element norm. If recip_growth_factor \ll 1, the stability of the LU factorization of (equilibrated) A could be poor. This also means that the solution x, condition estimate rcond, and forward error bound ferr could be unreliable. If the factorization fails with fail.code = NE_SINGULAR, then recip_growth_factor contains the reciprocal pivot growth factor for the leading fail.errnum columns of A.

22: 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 \( \langle value \rangle \) had an illegal value.

NE_INT
   On entry, \( n = \langle value \rangle \).
   Constraint: \( n \geq 0 \).
On entry, \( nrhs = \langle value \rangle \).
Constraint: \( nrhs \geq 0 \).

On entry, \( pda = \langle value \rangle \).
Constraint: \( pda > 0 \).

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

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

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

\textbf{NE\_INT\_2}

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

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

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

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

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

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

\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 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.

\textbf{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.

\textbf{NE\_SINGULAR}

Element \( \langle value \rangle \) of the diagonal is exactly zero. The factorization has been completed, but the factor \( U \) is exactly singular, so the solution and error bounds could not be computed. \( rcond = 0.0 \) is returned.

\textbf{NE\_SINGULAR\_WP}

\( U \) is nonsingular, but \( rcond \) is less than \textit{machine precision}, meaning that the matrix is singular to working precision. Nevertheless, the solution and error bounds are computed because there are a number of situations where the computed solution can be more accurate than the value of \( rcond \) would suggest.
7 Accuracy

For each right-hand side vector $b$, the computed solution $x$ is the exact solution of a perturbed system of equations $(A + E)x = b$, where

$$|E| \leq c(n)\epsilon P|L||U|,$$

$c(n)$ is a modest linear function of $n$, and $\epsilon$ is the machine precision. See Section 9.3 of Higham (2002) for further details.

If $x$ is the true solution, then the computed solution $\hat{x}$ satisfies a forward error bound of the form

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} \leq w_c \text{cond}(A, \hat{x}, b)$$

where $\text{cond}(A, \hat{x}, b) = \frac{\|A^{-1}(|A||\hat{x}| + |b|)\|_\infty}{{\|\hat{x}\|_\infty}} \leq \text{cond}(A) = \frac{\|A^{-1}\|_\infty}{\|A\|_\infty} \leq \kappa_\infty(A)$. If $\hat{x}$ is the $j$th column of $X$, then $w_c$ is returned in $\text{berr}[j - 1]$ and a bound on $\frac{\|x - \hat{x}\|_\infty}{{\|\hat{x}\|_\infty}}$ is returned in $\text{ferr}[j - 1]$. See Section 4.4 of Anderson et al. (1999) for further details.

8 Parallelism and Performance

nag_zgesvx (f07apc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_zgesvx (f07apc) 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

The factorization of $A$ requires approximately $\frac{2}{3}n^3$ floating-point operations.

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 $8n^2$ operations.

In practice the condition number estimator is very reliable, but it can underestimate the true condition number; see Section 15.3 of Higham (2002) for further details.

The real analogue of this function is nag_dgesvx (f07abc).

10 Example

This example solves the equations

$$AX = B,$$

where $A$ is the general matrix

$$A = \begin{pmatrix}
-1.34 + 2.55i & 0.28 + 3.17i & -6.39 - 2.20i & 0.72 - 0.92i \\
-1.70 - 14.10i & 33.10 - 1.50i & -1.50 + 13.40i & 12.90 + 13.80i \\
-3.29 - 2.39i & -1.91 + 4.42i & -0.14 + 1.35i & 1.72 + 1.35i \\
2.41 + 0.92i & -0.56 + 1.47i & -0.83 - 0.69i & -1.96 + 0.67i
\end{pmatrix},$$

and

$$B = \begin{pmatrix}
26.26 + 51.78i & 31.32 - 6.70i \\
64.30 - 86.00i & 158.60 - 14.20i \\
-5.75 + 25.31i & -2.15 + 30.19i \\
1.16 + 2.57i & -2.56 + 7.55i
\end{pmatrix}.$$
Error estimates for the solutions, information on scaling, an estimate of the reciprocal of the condition number of the scaled matrix $A$ and an estimate of the reciprocal of the pivot growth factor for the factorization of $A$ are also output.

10.1 Program Text

/* nag_zgesvx (f07apc) 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>

int main(void)
{
    /* Scalars */
    double growth_factor, rcond;
    Integer exit_status = 0, i, j, n, nrhs, pda, pdaf, pdb, pdx;

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

    /* Nag Types */
    NagError fail;
    Nag_OrderType order;
    Nag_EquilibrationType equed;

    INIT_FAIL(fail);

    printf("nag_zgesvx (f07apc) 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"%*[\n] ", &n, &nrhs);
    #else
    scanf("%"NAG_IFMT"%"NAG_IFMT"%*[\n] ", &n, &nrhs);
    #endif
    if (n < 0 || nrhs < 0)
    {
        printf("Invalid n or nrhs\n");
        exit_status = 1;
        return exit_status;
    }

    pda = n;
pdaf = n;
#ifdef NAG_COLUMN_MAJOR
pdb = n;
pdx = n;
#else
pdb = nrhs;
pdx = nrhs;
#endif

/* 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 * nrhs, Complex)) ||
    !(!berr = NAG_ALLOC(nrhs, double)) ||
    !(!c = NAG_ALLOC(n, double)) ||
    !(!ferr = NAG_ALLOC(nrhs, double)) ||
    !(!r = NAG_ALLOC(n, double)) ||
    !(!ipiv = NAG_ALLOC(n, Integer)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Read A and B 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("%*[^
"]");
    #else
        scanf("%*[^
"]");
    #endif
for (i = 1; i <= n; ++i)
    for (j = 1; j <= nrhs; ++j)
        #ifdef _WIN32
            scanf_s("( %lf , %lf )", &B(i, j).re, &B(i, j).im);
        #else
            scanf("( %lf , %lf )", &B(i, j).re, &B(i, j).im);
        #endif
    #ifdef _WIN32
        scanf_s("%*[`
"]");
    #else
        scanf("%*[`
"]");
    #endif

    /* Solve the equations AX = B for X using nag_zgesvx (f07apc). */
    nag_zgesvx(order, Nag_EquilibrateAndFactor, Nag_NoTrans, n, nrhs, a, pda, af, 
        pdaf, ipiv, &equed, r, c, b, pdb, x, pdx, &rcond, ferr, berr, 
        &growth_factor, &fail);
    if (fail.code != NE_NOERROR && fail.code != NE_SINGULAR)
    {
        printf("Error from nag_zgesvx (f07apc).\n");
        exit_status = 1;
        goto END;
    }

    /* Print solution using nag_gen_complex_mat_print_comp (x04dbc). */
    fflush(stdout);
    nag_gen_complex_mat_print_comp(order, Nag_GeneralMatrix, Nag_NoUnitDiag, n, 
        nrhs, x, pdx, Nag_DenseColumnForm, "%7.4f", 
        "Solution(s)", 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;
}

/* Print error bounds, condition number, the form of equilibration
 * and the pivot growth factor */
printf("Backward errors (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j) printf("%11.1e%s", berr[j - 1], j%7 == 0?"\n":"");
printf("Estimated forward error bounds (machine-dependent)\n");
for (j = 1; j <= nrhs; ++j) printf("%11.1e%s", ferr[j - 1], j%7 == 0?"\n":"");

printf("Reciprocal condition number estimate of scaled matrix\n");
printf("%11.1e\n", rcond);
printf("Estimate of reciprocal pivot growth factor\n%11.1e\n", growth_factor);

if (fail.code == NE_SINGULAR)
{
    printf("Error from nag_zgesvx (f07apc).\n%s\n", fail.message);
    exit_status = 1;
}
END:
NAG_FREE(a);
NAG_FREE(af);
NAG_FREE(b);
NAG_FREE(x);
NAG_FREE(berr);
NAG_FREE(c);
NAG_FREE(ferr);
NAG_FREE(r);
NAG_FREE(ipiv);

return exit_status;
}

10.2 Program Data

nag_zgesvx (f07apc) Example Program Data

: n and nrhs

\begin{center}
\begin{tabular}{cccc}
  4 & 2 & \text{: n and nrhs} \\
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{cccccccc}
  (-1.34, 2.55) & (0.28, 3.17) & (-6.39, -2.20) & (0.72, -0.92) \\
  (-1.70, -14.10) & (33.10, -6.70) & (-1.50, 13.40) & (12.90, 13.80) \\
  (-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) \\
\end{tabular}
\end{center}

: matrix A

\begin{center}
\begin{tabular}{cccc}
  (26.26, 51.78) & (31.32, -6.70) \\
  (64.30, -86.80) & (158.60, -14.20) \\
  (-5.75, 25.31) & (-2.15, 30.19) \\
  (1.16, 2.57) & (-2.56, 7.55) \\
\end{tabular}
\end{center}

: matrix B
10.3 Program Results

nag_zgesvx (f07apc) Example Program Results

Solution(s)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( 1.0000, 1.0000)</td>
<td>(-1.0000, -2.0000)</td>
</tr>
<tr>
<td>2</td>
<td>( 2.0000, -3.0000)</td>
<td>( 5.0000, 1.0000)</td>
</tr>
<tr>
<td>3</td>
<td>(-4.0000, -5.0000)</td>
<td>(-3.0000, 4.0000)</td>
</tr>
<tr>
<td>4</td>
<td>( 0.0000, 6.0000)</td>
<td>( 2.0000, -3.0000)</td>
</tr>
</tbody>
</table>

Backward errors (machine-dependent)

|   | 5.3e-17 | 4.8e-17 |

Estimated forward error bounds (machine-dependent)

|   | 5.8e-14 | 7.4e-14 |

A has been row scaled as diag(R)*A

Reciprocal condition number estimate of scaled matrix

|   | 1.0e-02 |

Estimate of reciprocal pivot growth factor

|   | 8.3e-01 |