E04ZCF – NAG Fortran Library Routine Document

Note. Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

1 Purpose

E04ZCF checks that user-supplied routines for evaluating an objective function, constraint functions and their first derivatives, produce derivative values which are consistent with the function and constraint values calculated.

2 Specification

```
 SUBROUTINE E04ZCF(N, NCNLN, NROWJ, CONFUN, OBJFUN, C, CJAC, OBJF,
1                  OBJGRD, X, WORK, LWORK, IFAIL)
 INTEGER           N, NCNLN, NROWJ, LWORK, IFAIL
 real              C(NCNLN), CJAC(NROWJ,N), OBJF, OBJGRD(N), X(N),
1                  WORK(LWORK)
 EXTERNAL          CONFUN, OBJFUN
```

3 Description

Routines for minimizing a function of several variables subject to general equality and/or inequality constraints may require the user to provide subroutines to evaluate the objective function $F(x_1, x_2, \ldots, x_n)$, constraint functions $c_i(x_1, x_2, \ldots, x_n)$, for $i = 1, 2, \ldots, m$, and their first derivatives. E04ZCF is designed to check the derivatives calculated by such user-supplied routines. As well as the routines to be checked (CONFUN and OBJFUN), the user must supply a point $x = (x_1, x_2, \ldots, x_n)^T$ at which the checks will be made.

To check the first derivatives of F, E04ZCF first calls OBJFUN to evaluate F and its first derivatives $g_j = \frac{\partial F}{\partial x_j}$, for $j = 1, 2, \ldots, n$ at x. The components of the user-supplied derivatives along two orthogonal directions (defined by unit vectors p_1 and p_2, say) are then calculated; these will be $g^T p_1$ and $g^T p_2$ respectively. The same components are also estimated by finite differences, giving quantities

$$v_k = \frac{F(x + hp_k) - F(x)}{h}, \qquad k = 1, 2$$

where h is a small positive scalar. If the relative difference between v_1 and $g^T p_1$ or between v_2 and $g^T p_2$ is judged too large, an error indicator is set.

When $n = 1$ only p_1 and v_1 are generated.

Similar checks are made of whether components of the first derivatives

$$\frac{\partial c_i}{\partial x_j}, \qquad i = 1, 2, \ldots, m; j = 1, 2, \ldots, n$$

(as calculated by CONFUN at x) are consistent with difference approximations to the same quantities.

4 References

[1] Gill P E, Murray W and Wright M H (1981) *Practical Optimization* Academic Press

5 Parameters

1: N — INTEGER *Input*

On entry: the number n of independent variables in the objective and constraint functions.

Constraint: N ≥ 1.

2: NCNLN — INTEGER *Input*

On entry: the number m of constraint functions.

Constraint: NCNLN ≥ 0.

3: NROWJ — INTEGER *Input*

On entry: the first dimension of the array CJAC as declared in the (sub)program from which E04ZCF is called.

Constraint: NROWJ $\geq \max(1, \text{NCNLN})$.

4: CONFUN — SUBROUTINE, supplied by the user. *External Procedure*

CONFUN must calculate the vector $c(x)$ of nonlinear constraint functions and its Jacobian for a specified n-vector x. If there are no nonlinear constraints (NCNLN = 0), CONFUN will not be called by E04ZCF and CONFUN may be the dummy routine E04VDM. (E04VDM is included in the NAG Fortran Library and so need not be supplied by the user. Its name may be implementation-dependent: see the Users' Note for your implementation for details.) If there are nonlinear constraints, E04ZCF always calls CONFUN and OBJFUN together, in that order.

Its specification is:

```
SUBROUTINE CONFUN(MODE, NCNLN, N, NROWJ, X, C, CJAC, NSTATE)
INTEGER           MODE, NCNLN, N, NROWJ, NSTATE
real              X(N), C(NROWJ), CJAC(NROWJ,N)
```

1: MODE — INTEGER *Input/Output*

MODE is a flag that the user may set within CONFUN to indicate a failure in the evaluation of the nonlinear constraints.

On entry: MODE is always nonnegative.

On exit: if MODE is negative on exit from CONFUN, the execution of E04ZCF will be terminated with IFAIL containing the negative value of MODE.

2: NCNLN — INTEGER *Input*

On entry: the number m of nonlinear constraints, as input to E04ZCF.

3: N — INTEGER *Input*

On entry: the number n of variables, as input to E04ZCF.

4: NROWJ — INTEGER *Input*

On entry: the first dimension of the array CJAC and the length of the array C, as input to E04ZCF.

5: X(N) — ***real*** array *Input*

On entry: the vector x of variables at which the constraint functions are to be evaluated.

6: C(NROWJ) — ***real*** array *Output*

On exit: C must contain the NCNLN nonlinear constraint values, with the value of the jth nonlinear constraint in C(j).

7: CJAC(NROWJ,N) — ***real*** array *Output*

On exit: CJAC must contain the Jacobian of the nonlinear constraint functions with the ith row of CJAC containing the gradient of the ith nonlinear constraint, i.e., CJAC(i, j) must contain the partial derivative of c_i with respect to x_j. If CJAC contains any constant elements, a saving in computation can be made by setting them once only, when NSTATE = 1 (see below).

8: NSTATE — INTEGER *Input*

On entry: NSTATE will be 1 on the first call to CONFUN by E04ZCF, and is 0 for the two subsequent calls. Thus, if the user wishes, NSTATE may be tested within CONFUN in order to perform certain calculations once only. For example, the user may read data or initialise COMMON blocks when NSTATE = 1. In addition, the constant elements of CJAC can be set in CONFUN when NSTATE = 1, and need not be defined on subsequent calls.

CONFUN must be declared as EXTERNAL in the (sub)program from which E04ZCF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: OBJFUN — SUBROUTINE, supplied by the user. *External Procedure*

OBJFUN must calculate the objective function $F(x)$ and its gradient for a specified n-element vector x.

Its specification is:

```
SUBROUTINE OBJFUN(MODE, N, X, OBJF, OBJGRD, NSTATE)
INTEGER           MODE, N, NSTATE
real              X(N), OBJF, OBJGRD(N)
```

1: MODE — INTEGER *Input/Output*

MODE is a flag that the user may set within OBJFUN to indicate a failure in the evaluation of the objective function.

On entry: MODE is always non-negative.

On exit: if MODE is negative on exit from OBJFUN, the execution of E04ZCF will be terminated with IFAIL set to MODE.

2: N — INTEGER *Input*

On entry: the number n of variables as input to E04ZCF.

3: X(N) — ***real*** array *Input*

On entry: the vector x of variables at which the objective function is to be evaluated.

4: OBJF — ***real*** *Output*

On exit: OBJF must be set to the value of the objective function.

5: OBJGRD(N) — ***real*** array *Output*

On exit: OBJGRD must contain the gradient vector of the objective function, with OBJGRD(j) containing the partial derivative of F with respect to x_j.

6: NSTATE — INTEGER *Input*

On entry: NSTATE will be 1 on the first call to OBJFUN by E04ZCF, and is 0 on the two subsequent calls. Thus, if the user wishes, NSTATE may be tested in order to perform certain calculations only on the first call of OBJFUN – e.g., read data or initialise COMMON blocks. Note that if there are any nonlinear constraints, CONFUN and OBJFUN are called together, in that order.N

OBJFUN must be declared as EXTERNAL in the (sub)program from which E04ZCF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: C(NCNLN) — ***real*** array *Output*

On exit: unless the user sets MODE negative in the first call of CONFUN, C(i) contains the value of $c_i(x)$ at the point given by the user in x, for $i = 1, 2, \ldots,$ NCNLN. If NCNLN is zero, C is not referenced.

7: CJAC(NROWJ,N) — ***real*** array *Output*

On exit: unless the user sets MODE negative in the first call of CONFUN, CJAC(i, j) contains the value of the derivative $\frac{\partial c_i}{\partial x_j}$ at the point given in X, as calculated by CONFUN, for $j = 1, 2, \ldots, n$; $i = 1, 2, \ldots,$ NCNLN.

If NCNLN is zero, CJAC is not referenced.

8: OBJF — ***real*** *Output*

On exit: unless the user sets MODE negative in the first call of OBJFUN, OBJF contains the value of the objective function $F(x)$ at the point given by the user in X.

9: OBJGRD(N) — ***real*** array *Output*

On exit: unless the user sets MODE negative in the first call of OBJFUN, OBJGRD(j) contains the value of the derivative $\frac{\partial F}{\partial x_j}$ at the point given in X, as calculated by OBJFUN, for $j = 1, 2, \ldots, n$.

10: X(N) — ***real*** array *Input*

On entry: X(j), for $j = 1, 2, \ldots,$ N must be set to the co-ordinates of a suitable point x at which to check the derivatives calculated by CONFUN and OBJFUN. 'Obvious' settings such as 0 or 1, should not be used since, at such points, incorrect terms may take correct values (particularly zero), so that errors could go undetected. Similarly, it is preferable that no two elements of x should be the same.

11: WORK(LWORK) — ***real*** array *Workspace*

12: LWORK — INTEGER *Input*

On entry: the length of WORK as declared in the (sub)program from which E04ZCF is called.

Constraint: LWORK $\geq 4 \times$ N + NCNLN + N $\times$ NROWJ.

13: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. Users who are unfamiliar with this parameter should refer to Chapter P01 for details.

On exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq 0$ on exit, users are recommended to set IFAIL to -1 before entry. **It is then essential to test the value of IFAIL on exit**. To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6 Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL < 0

A negative value of IFAIL indicates an exit from E04ZCF because the user has set MODE negative in OBJFUN or CONFUN. The value of IFAIL will be the same as the user's setting of MODE. The checks on OBJFUN and CONFUN will not have been completed.

IFAIL = 1

On entry, N < 1,
or NCNLN < 0,
or NROWJ $<$ max(1, NCNLN),
or LWORK $< 4 \times$ N + NCNLN + N $\times$ NROWJ.

IFAIL = 2

The user should check carefully the derivation and programming of expressions for the derivatives of $F(x)$, because it is very unlikely that OBJFUN is calculating them correctly.

IFAIL $= 2 + i$, for $i = 1, 2, \ldots,$ NCNLN

The user should check carefully the derivation and programming of expressions for the derivatives of $c_i(x)$, because it is very unlikely that CONFUN is calculating them correctly. See Section 7.

7 Accuracy

IFAIL is set to 2 if

$$(v_k - g^T p_k)^2 \geq h \times ((g^T p_k)^2 + 1)$$

for $k = 1$ or 2. (See Section 3 for definitions of the quantities involved.) The scalar h is set equal to $\sqrt{\epsilon}$, where ϵ is the ***machine precision*** as given by X02AJF.

IFAIL is set to $2 + i$ if a relation analogous to that given above holds for c_i and its calculated derivatives.

8 Further Comments

The user-supplied routines CONFUN and OBJFUN are both called 3 times, unless NCNLN = 0 in which case CONFUN is not called.

Before using E04ZCF to check the calculation of first derivatives, the user should be confident that CONFUN and OBJFUN are calculating F and the c_i correctly. The usual way of checking the calculation of these function values is to compare values of $F(x)$ and the $c_i(x)$ calculated by OBJFUN and CONFUN at non-trivial points x with values calculated independently. ('Non-trivial' means that, as when setting x before calling E04ZCF, co-ordinates such as 0 or 1 should be avoided.)

9 Example

The example problem has nine variables, finite bounds on six of the variables, four general linear constraints, and fifteen nonlinear constraints.

The objective function is

$$F(x) = -x_2x_6 + x_1x_7 - x_3x_7 - x_5x_8 + x_4x_9 + x_3x_8$$

and the fifteen nonlinear constraint functions are

$$\begin{aligned}
c_1(x) &= x_1^2 + x_6^2, \\
c_2(x) &= (x_2 - x_1)^2 + (x_7 - x_6)^2, \\
c_3(x) &= (x_3 - x_1)^2 + x_6^2, \\
c_4(x) &= (x_1 - x_4)^2 + (x_6 - x_8)^2, \\
c_5(x) &= (x_1 - x_5)^2 + (x_6 - x_9)^2, \\
c_6(x) &= x_2^2 + x_7^2, \\
c_7(x) &= (x_3 - x_2)^2 + x_7^2, \\
c_8(x) &= (x_4 - x_2)^2 + (x_8 - x_7)^2, \\
c_9(x) &= (x_2 - x_5)^2 + (x_7 - x_9)^2, \\
c_{10}(x) &= x_3^2, \\
c_{11}(x) &= (x_4 - x_3)^2 + x_8^2, \\
c_{12}(x) &= (x_5 - x_3)^2 + x_9^2, \\
c_{13}(x) &= x_4^2 + x_8^2, \\
c_{14}(x) &= (x_4 - x_5)^2 + (x_9 - x_8)^2, \\
c_{15}(x) &= x_5^2 + x_9^2.
\end{aligned}$$

The example checks the gradients at two separate points.

9.1 Program Text

Note. The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*     E04ZCF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
```

```
*     .. Local Scalars ..
      real             OBJF
      INTEGER          I, IFAIL, J, K, LWORK, N, NCNLN, NROWJ
*     .. Local Arrays ..
      real             C(20), CJAC(20,9), OBJGRD(9), WORK(351), X(9)
*     .. External Subroutines ..
      EXTERNAL         CONFUN, E04ZCF, OBJFUN
*     .. Data statements ..
      DATA             NROWJ/20/, LWORK/351/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04ZCF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      N = 9
      NCNLN = 15
*     Read in two points and check the derivatives at each point.
      DO 20 K = 1, 2
         READ (NIN,99999) (X(J),J=1,N)
         IFAIL = 1
*
         CALL E04ZCF(N,NCNLN,NROWJ,CONFUN,OBJFUN,C,CJAC,OBJF,OBJGRD,X,
     +              WORK,LWORK,IFAIL)
*
         WRITE (NOUT,*)
         IF (IFAIL.EQ.0) THEN
            WRITE (NOUT,*) 'Derivatives probably correct at the point'
            WRITE (NOUT,99998) (X(J),J=1,N)
         ELSE IF (IFAIL.EQ.1) THEN
            WRITE (NOUT,*) 'Incorrect parameter supplied to E04ZCF'
            STOP
         ELSE IF (IFAIL.EQ.2) THEN
            WRITE (NOUT,*)
     +        'Probable error in derivative of objective function'
            WRITE (NOUT,99998) (X(J),J=1,N)
            WRITE (NOUT,*) 'The computed gradients are'
            WRITE (NOUT,99998) (OBJGRD(J),J=1,N)
         ELSE
            I = IFAIL - 2
            WRITE (NOUT,99997)
     +        'Probable error in derivative of constraint', I,
     +        '  at the point'
            WRITE (NOUT,99998) (X(J),J=1,N)
            WRITE (NOUT,*)
     +        'The computed gradients of this constraint are'
            WRITE (NOUT,99998) (CJAC(I,J),J=1,N)
         END IF
   20 CONTINUE
      STOP
*
99999 FORMAT (1X,9F6.2)
99998 FORMAT (1X,1P,5e12.4)
99997 FORMAT (1X,A,I4,A)
      END
*
      SUBROUTINE OBJFUN(MODE,N,X,OBJF,OBJGRD,NSTATE)
*     .. Scalar Arguments ..
      real              OBJF
      INTEGER           MODE, N, NSTATE
```

```
*     .. Array Arguments ..
      real              OBJGRD(N), X(N)
*     .. Executable Statements ..
      OBJF = X(2)*X(6) - X(1)*X(7) + X(3)*X(7) + X(5)*X(8) - X(4)*X(9) -
     +       X(3)*X(8)
      OBJF = -OBJF
      OBJGRD(1) = X(7)
      OBJGRD(2) = -X(6)
      OBJGRD(3) = -X(7) + X(8)
      OBJGRD(4) = X(9)
      OBJGRD(5) = -X(8)
      OBJGRD(6) = -X(2)
      OBJGRD(7) = -X(3) + X(1)
      OBJGRD(8) = -X(5) + X(3)
      OBJGRD(9) = X(4)
      RETURN
      END
*
      SUBROUTINE CONFUN(MODE,NCNLN,N,NROWJ,X,C,CJAC,NSTATE)
*     .. Parameters ..
      real              ZERO, TWO
      PARAMETER         (ZERO=0.0e0,TWO=2.0e0)
*     .. Scalar Arguments ..
      INTEGER           MODE, N, NCNLN, NROWJ, NSTATE
*     .. Array Arguments ..
      real              C(NROWJ), CJAC(NROWJ,N), X(N)
*     .. Local Scalars ..
      INTEGER           I, J
*     .. Executable Statements ..
*     The zero elements of Jacobian matrix are set only once. This
*     occurs during the first call to CONFUN (NSTATE = 1).
      IF (NSTATE.EQ.1) THEN
         DO 40 J = 1, N
            DO 20 I = 1, NCNLN
               CJAC(I,J) = ZERO
   20       CONTINUE
   40    CONTINUE
      END IF
      C(1) = X(1)**2 + X(6)**2
      CJAC(1,1) = TWO*X(1)
      CJAC(1,6) = TWO*X(6)
      C(2) = (X(2)-X(1))**2 + (X(7)-X(6))**2
      CJAC(2,1) = -TWO*(X(2)-X(1))
      CJAC(2,2) = TWO*(X(2)-X(1))
      CJAC(2,6) = -TWO*(X(7)-X(6))
      CJAC(2,7) = TWO*(X(7)-X(6))
      C(3) = (X(3)-X(1))**2 + X(6)**2
      CJAC(3,1) = -TWO*(X(3)-X(1))
      CJAC(3,3) = TWO*(X(3)-X(1))
      CJAC(3,6) = TWO*X(6)
      C(4) = (X(1)-X(4))**2 + (X(6)-X(8))**2
      CJAC(4,1) = TWO*(X(1)-X(4))
      CJAC(4,4) = -TWO*(X(1)-X(4))
      CJAC(4,6) = TWO*(X(6)-X(8))
      CJAC(4,8) = -TWO*(X(6)-X(8))
      C(5) = (X(1)-X(5))**2 + (X(6)-X(9))**2
      CJAC(5,1) = TWO*(X(1)-X(5))
      CJAC(5,5) = -TWO*(X(1)-X(5))
```

```
      CJAC(5,6) = TWO*(X(6)-X(9))
      CJAC(5,9) = -TWO*(X(6)-X(9))
      C(6) = X(2)**2 + X(7)**2
      CJAC(6,2) = TWO*X(2)
      CJAC(6,7) = TWO*X(7)
      C(7) = (X(3)-X(2))**2 + X(7)**2
      CJAC(7,2) = -TWO*(X(3)-X(2))
      CJAC(7,3) = TWO*(X(3)-X(2))
      CJAC(7,7) = TWO*X(7)
      C(8) = (X(4)-X(2))**2 + (X(8)-X(7))**2
      CJAC(8,2) = -TWO*(X(4)-X(2))
      CJAC(8,4) = TWO*(X(4)-X(2))
      CJAC(8,7) = -TWO*(X(8)-X(7))
      CJAC(8,8) = TWO*(X(8)-X(7))
      C(9) = (X(2)-X(5))**2 + (X(7)-X(9))**2
      CJAC(9,2) = TWO*(X(2)-X(5))
      CJAC(9,5) = -TWO*(X(2)-X(5))
      CJAC(9,7) = TWO*(X(7)-X(9))
      CJAC(9,9) = -TWO*(X(7)-X(9))
      C(10) = X(3)**2
      CJAC(10,3) = TWO*X(3)
      C(11) = (X(4)-X(3))**2 + X(8)**2
      CJAC(11,3) = -TWO*(X(4)-X(3))
      CJAC(11,4) = TWO*(X(4)-X(3))
      CJAC(11,8) = TWO*X(8)
      C(12) = (X(5)-X(3))**2 + X(9)**2
      CJAC(12,3) = -TWO*(X(5)-X(3))
      CJAC(12,5) = TWO*(X(5)-X(3))
      CJAC(12,9) = TWO*X(9)
      C(13) = X(4)**2 + X(8)**2
      CJAC(13,4) = TWO*X(4)
      CJAC(13,8) = TWO*X(8)
      C(14) = (X(4)-X(5))**2 + (X(9)-X(8))**2
      CJAC(14,4) = TWO*(X(4)-X(5))
      CJAC(14,5) = -TWO*(X(4)-X(5))
      CJAC(14,8) = -TWO*(X(9)-X(8))
      CJAC(14,9) = TWO*(X(9)-X(8))
      C(15) = X(5)**2 + X(9)**2
      CJAC(15,5) = TWO*X(5)
      CJAC(15,9) = TWO*X(9)
      RETURN
      END
```

9.2 Program Data

```
E04ZCF Example Program Data
  1.10  1.20  1.30  1.40  1.50  1.60  1.70  1.80  1.90
  1.10  2.20  3.30  4.40  5.50  6.60  7.70  8.80  9.90
```

9.3 Program Results

```
E04ZCF Example Program Results

Derivatives probably correct at the point
  1.1000E+00  1.2000E+00  1.3000E+00  1.4000E+00  1.5000E+00
  1.6000E+00  1.7000E+00  1.8000E+00  1.9000E+00

Derivatives probably correct at the point
  1.1000E+00  2.2000E+00  3.3000E+00  4.4000E+00  5.5000E+00
  6.6000E+00  7.7000E+00  8.8000E+00  9.9000E+00
```