E04UFF – 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.

Note. *This routine uses* optional parameters *to define choices in the problem specification and in the details of the algorithm. If you wish to use* default *settings for all of the optional parameters, you need only read Section 1 to Section 9 of this document. Refer to the additional Section 10, Section 11 and Section 12 for a detailed description of the algorithm, the specification of the optional parameters and a description of the monitoring information produced by the routine.*

1 Purpose

E04UFF is designed to minimize an arbitrary smooth function subject to constraints (which may include simple bounds on the variables, linear constraints and smooth nonlinear constraints) using a sequential quadratic programming (SQP) method. As many first derivatives as possible should be supplied by the user; any unspecified derivatives are approximated by finite differences. It is not intended for large sparse problems.

E04UFF may also be used for unconstrained, bound-constrained and linearly constrained optimization.

E04UFF uses **reverse communication** for evaluating the objective function, the nonlinear constraint functions, and any of their derivatives.

2 Specification

```
 SUBROUTINE E04UFF(IREVCM, N, NCLIN, NCNLN, LDA, LDCJ, LDR, A, BL,
1                  BU, ITER, ISTATE, C, CJAC, CLAMDA, OBJF, OBJGRD,
2                  R, X, NEEDC, IWORK, LIWORK, WORK, LWORK, IFAIL)
 INTEGER           IREVCM, N, NCLIN, NCNLN, LDA, LDCJ, LDR, ITER,
1                  ISTATE(N+NCLIN+NCNLN), NEEDC(*), IWORK(LIWORK),
2                  LIWORK, LWORK, IFAIL
 real              A(LDA,*), BL(N+NCLIN+NCNLN), BU(N+NCLIN+NCNLN),
1                  C(*), CJAC(LDCJ,*), CLAMDA(N+NCLIN+NCNLN), OBJF,
2                  OBJGRD(N), R(LDR,N), X(N), WORK(LWORK)
```

3 Description

E04UFF is designed to solve the nonlinear programming problem – the minimization of a smooth nonlinear function subject to a set of constraints on the variables. The problem is assumed to be stated in the following form:

$$\underset{x \in R^n}{\text{Minimize}} \; F(x) \;\text{ subject to }\; l \leq \begin{Bmatrix} x \\ A_L x \\ c(x) \end{Bmatrix} \leq u, \tag{1}$$

where $F(x)$ (the *objective function*) is a nonlinear function, A_L is an n_L by n constant matrix, and $c(x)$ is an n_N element vector of nonlinear constraint functions. (The matrix A_L and the vector $c(x)$ may be empty.) The objective function and the constraint functions are assumed to be smooth, i.e., at least twice-continuously differentiable. (The method of E04UFF will usually solve (1) if there are only isolated discontinuities away from the solution.)

Note that although the bounds on the variables could be included in the definition of the linear constraints, we prefer to distinguish between them for reasons of computational efficiency. For the same reason, the linear constraints should **not** be included in the definition of the nonlinear constraints. Upper and lower bounds are specified for all the variables and for all the constraints. An *equality* constraint can be specified by setting $l_i = u_i$. If certain bounds are not present, the associated elements of l or u can be set to special values that will be treated as $-\infty$ or $+\infty$. (See the description of the optional parameter **Infinite Bound Size** in Section 11.2).

If there are no nonlinear constraints in (1) and F is linear or quadratic, then it will generally be more efficient to use one of E04MFF, E04NCF or E04NFF, or E04NKF if the problem is large and sparse. If

the problem is large and sparse and does have nonlinear constraints, the MINOS package (see Murtagh and Saunders [12]) should be used, since E04UFF treats all matrices as dense.

E04UFF uses reverse communication for evaluating $F(x)$, $c(x)$ and as many of their first partial derivatives as possible; any remaining derivatives are approximated by finite differences; see Section 11.2 for a discussion of the optional parameter **Derivative Level**.

On initial entry, the user must supply an initial estimate of the solution to (1).

On intermediate exits, the calling program must compute appropriate values for the objective function, the nonlinear constraints or their derivatives, as specified by the parameter IREVCM, and then re-enter the routine. Just before an intermediate exit when derivatives are required, each element of the current array of derivatives OBJGRD or CJAC is initialised to a special value. On re-entry to the routine, any element that retains this value is estimated by finite differences.

For maximum reliability, it is preferable for the user to provide all partial derivatives (see Chapter 8 of Gill *et al.* [10], for a detailed discussion). If they cannot all be provided, it is advisable to provide as many as possible. While developing code to evaluate the objective function and the constraints, the optional parameter **Verify** (see Section 11.2) should be used to check the calculation of any known derivatives.

The method used by E04UFF is described in detail in Section 10.

E04UCF is an alternative routine which uses exactly the same method, but with **forward communication**: that is, the objective and constraint functions are evaluated by subroutines, supplied as parameters to the routine.

4 References

[1] Dennis J E Jr and Moré J J (1977) Quasi–Newton methods, motivation and theory *SIAM Rev.* **19** 46–89

[2] Dennis J E Jr and Schnabel R B (1981) A new derivation of symmetric positive-definite secant updates *nonlinear programming* (ed O L Mangasarian, R R Meyer and S M Robinson) **4** Academic Press 167–199

[3] Dennis J E Jr and Schnabel R B (1983) *Numerical Methods for Unconstrained Optimization and Nonlinear Equations* Prentice–Hall

[4] Fletcher R (1987) *Practical Methods of Optimization* Wiley (2nd Edition)

[5] Gill P E, Murray W, Saunders M A and Wright M H (1984) User's guide for SOL/QPSOL version 3.2 *Report SOL 84–5* Department of Operations Research, Stanford University

[6] Gill P E, Murray W, Saunders M A and Wright M H (1984) Procedures for optimization problems with a mixture of bounds and general linear constraints *ACM Trans. Math. Software* **10** 282–298

[7] Gill P E, Hammarling S, Murray W, Saunders M A and Wright M H (1986) User's guide for LSSOL (Version 1.0) *Report SOL 86–1* Department of Operations Research, Stanford University

[8] Gill P E, Murray W, Saunders M A and Wright M H (1986) Some theoretical properties of an augmented Lagrangian merit function *Report SOL 86–6R* Department of Operations Research, Stanford University

[9] Gill P E, Murray W, Saunders M A and Wright M H (1986) User's guide for NPSOL (Version 4.0) *Report SOL 86–2* Department of Operations Research, Stanford University

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

[11] Hock W and Schittkowski K (1981) *Test Examples for Nonlinear Programming Codes. Lecture Notes in Economics and Mathematical Systems* **187** Springer-Verlag

[12] Murtagh B A and Saunders M A (1983) MINOS 5.0 user's guide *Report SOL 83–20* Department of Operations Research, Stanford University

[13] Powell M J D (1974) Introduction to constrained optimization *Numerical Methods for Constrained Optimization* (ed P E Gill and W Murray) Academic Press 1–28

[14] Powell M J D (1983) Variable metric methods in constrained optimization *Mathematical Programming: The State of the Art* (ed A Bachem, M Grötschel and B Korte) Springer-Verlag 288–311

5 Parameters

Note: this routine uses **reverse communication**. Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the **parameter IREVCM**. Between intermediate exits and re-entries, **all parameters must remain unchanged except those specified by the value of IREVCM**.

1: IREVCM — INTEGER *Input/Output*

On initial entry: IREVCM must be set to 0.

On intermediate exit: IREVCM specifies what values the calling program must assign to parameters of E04UFF before re-entering the routine:

if IREVCM = 1, set OBJF to the value of the objective function $F(x)$;
if IREVCM = 2, set OBJGRD(j) to the value $\dfrac{\partial F}{\partial x_j}$ if available, for $j = 1, 2, \ldots, n$;
if IREVCM = 3, set OBJF and OBJGRD(j) as for IREVCM = 1 and IREVCM = 2;
if IREVCM = 4, set C(i) to the value of the constraint function $c_i(x)$, for each i such that NEEDC(i) > 0;
if IREVCM = 5, set CJAC(i, j) to the value $\dfrac{\partial c_i}{\partial x_j}$ if available, for each i such that NEEDC(i) > 0 and $j = 1, 2, \ldots, n$;
if IREVCM = 6, set C(i) and CJAC(i, j) as for IREVCM = 4 and IREVCM = 5;

On intermediate re-entry: IREVCM **must remain unchanged**, unless the user wishes to terminate the solution to the current problem: in this case IREVCM may be set to a negative value and then E04UFF will take a final exit with IFAIL set to this value of IREVCM.

On final exit: IREVCM = 0.

Constraint: IREVCM ≤ 6.

2: N — INTEGER *Input*

On initial entry: n, the number of variables.

Constraint: N > 0.

3: NCLIN — INTEGER *Input*

On initial entry: n_L, the number of general linear constraints.

Constraint: NCLIN ≥ 0.

4: NCNLN — INTEGER *Input*

On initial entry: n_N, the number of nonlinear constraints.

Constraint: NCNLN ≥ 0.

5: LDA — INTEGER *Input*

On initial entry: the first dimension of the array A as declared in the (sub)program from which E04UFF is called.

Constraint: LDA $\geq$ max(1,NCLIN).

6: LDCJ — INTEGER *Input*

On initial entry: the first dimension of the array CJAC as declared in the (sub)program from which E04UFF is called.

Constraint: LDCJ $\geq$ max(1,NCNLN).

7: LDR — INTEGER *Input*

On initial entry: the first dimension of the array R as declared in the (sub)program from which E04UFF is called.

Constraint: LDR $\geq$ N.

8: A(LDA,$*$) — ***real*** array *Input*

Note: the second dimension of the array A must be at least N when NCLIN > 0, and at least 1 otherwise.

On initial entry: the ith row of the array A must contain the ith row of the matrix A_L of general linear constraints in (1). That is, the ith row contains the coefficients of the ith general linear constraint, for $i = 1, 2, \ldots,$NCLIN.

If NCLIN $= 0$ then the array A is not referenced.

9: BL(N+NCLIN+NCNLN) — ***real*** array *Input*

10: BU(N+NCLIN+NCNLN) — ***real*** array *Input*

On initial entry: BL must contain the lower bounds and BU the upper bounds, for all the constraints in the following order. The first n elements of each array must contain the bounds on the variables, the next n_L elements the bounds for the general linear constraints (if any) and the next n_N elements the bounds for the general nonlinear constraints (if any). To specify a non-existent lower bound (i.e., $l_j = -\infty$), set BL(j) $\leq -bigbnd$, and to specify a non-existent upper bound (i.e., $u_j = +\infty$), set BU(j) $\geq bigbnd$; the default value of $bigbnd$ is 10^{20}, but this may be changed by the optional parameter **Infinite Bound Size** (see Section 11.2). To specify the jth constraint as an *equality*, set BL(j) = BU(j) = β, say, where $|\beta| < bigbnd$.

Constraints:

BL(j) $\leq$BU(j), for $j = 1, 2, \ldots,$N+NCLIN+NCNLN,
$|\beta| < bigbnd$ when BL(j) = BU(j) = β.

11: ITER — INTEGER *Output*

On final exit: the number of major iterations performed.

12: ISTATE(N+NCLIN+NCNLN) — INTEGER array *Input/Output*

On initial entry: ISTATE need not be set if the (default) **Cold Start** option is used.

If the **Warm Start** option has been chosen (see Section 11.2), the elements of ISTATE corresponding to the bounds and linear constraints define the initial working set for the procedure that finds a feasible point for the linear constraints and bounds. The active set at the conclusion of this procedure and the elements of ISTATE corresponding to nonlinear constraints then define the initial working set for the first QP subproblem. More precisely, the first n elements of ISTATE refer to the upper and lower bounds on the variables, the next n_L elements refer to the upper and lower bounds on $A_L x$, and the next n_N elements refer to the upper and lower bounds on $c(x)$. Possible values for ISTATE(j) are as follows:

ISTATE(j)	**Meaning**
0	The corresponding constraint is *not* in the initial QP working set.
1	This inequality constraint should be in the working set at its lower bound.
2	This inequality constraint should be in the working set at its upper bound.
3	This equality constraint should be in the initial working set. This value must not be specified unless BL(j) = BU(j).

The values -2, -1 and 4 are also acceptable but will be modified by the routine. If E04UFF has been called previously with the same values of N, NCLIN and NCNLN, ISTATE already contains satisfactory information. (See also the description of the optional parameter **Warm Start** in Section 11.2). The routine also adjusts (if necessary) the values supplied in X to be consistent with ISTATE.

Constraint: $-2 \leq$ ISTATE(j) ≤ 4, for $j = 1, 2, \ldots,$N+NCLIN+NCNLN.

On final exit: the status of the constraints in the QP working set at the point returned in X. The significance of each possible value of ISTATE(j) is as follows:

ISTATE(j)	**Meaning**
-2	This constraint violates its lower bound by more than the appropriate feasibility tolerance (see the optional parameters **Linear Feasibility Tolerance** and **Nonlinear Feasibility Tolerance** in Section 11.2). This value can occur only when no feasible point can be found for a QP subproblem.
-1	This constraint violates its upper bound by more than the appropriate feasibility tolerance (see the optional parameters **Linear Feasibility Tolerance** and **Nonlinear Feasibility Tolerance** in Section 11.2). This value can occur only when no feasible point can be found for a QP subproblem.
0	The constraint is satisfied to within the feasibility tolerance, but is not in the QP working set.
1	This inequality constraint is included in the QP working set at its lower bound.
2	This inequality constraint is included in the QP working set at its upper bound.
3	This constraint is included in the QP working set as an equality. This value of ISTATE can occur only when BL(j) = BU(j).

13: C(∗) — ***real*** array *Input/Output*

Note: the dimension of the array C must be at least max(1,NCNLN).

On initial entry: C need not be set.

On intermediate re-entry: if IREVCM = 4 or 6 and NEEDC(i) > 0, C(i) must contain the value of the ith constraint at x. The remaining elements of C, corresponding to the non-positive elements of NEEDC, are ignored.

On final exit: if NCNLN > 0, C(i) contains the value of the ith nonlinear constraint function c_i at the final iterate, for $i = 1, 2, \ldots,$NCNLN.

If NCNLN = 0 then the array C is not referenced.

14: CJAC(LDCJ,∗) — ***real*** array *Input/Output*

Note: the second dimension of the array CJAC must be at least N when NCNLN > 0, and at least 1 otherwise.

On initial entry: in general, CJAC need not be initialised before the call to E04UFF. However, if the optional parameter **Derivative Level** = 2 or 3 (default value = 3; see Section 11.2), the user may optionally set the constant elements of CJAC. Such constant elements need not be re-assigned on subsequent intermediate exits.

If all elements of the constraint Jacobian are known (i.e., **Derivative Level** = 2 or 3), any constant elements may be assigned to CJAC one time only at the start of the optimization. An element of CJAC that is not subsequently assigned during an intermediate exit will retain its initial value throughout. Constant elements may be loaded into CJAC either before the call to E04UFF or during the first intermediate exit. The ability to preload constants is useful when many Jacobian elements are identically zero, in which case CJAC may be initialised to zero and non-zero elements may be reset during intermediate exits.

On intermediate re-entry: if IREVCM = 5 or 6 and NEEDC(i) > 0, the ith row of CJAC must contain the available elements of the vector ∇c_i given by

$$\nabla c_i = \left(\frac{\partial c_i}{\partial x_1}, \frac{\partial c_i}{\partial x_2}, \ldots, \frac{\partial c_i}{\partial x_n} \right)^T,$$

where $\frac{\partial c_i}{\partial x_j}$ is the partial derivative of the ith constraint with respect to the jth variable, evaluated at the point x. The remaining rows of CJAC, corresponding to non-positive elements of NEEDC, are ignored.

Note that constant non-zero elements do affect the values of the constraints. Thus, if CJAC(i, j) is set to a constant value, it need not be reset during subsequent intermediate exits, but the value

CJAC(i, j)*X(j) must nonetheless be added to C(i). For example, if CJAC(1,1) = 2 and CJAC(1,2) = −5, then the term 2*X(1) − 5*X(2) must be included in the definition of C(1).

It must be emphasized that, if **Derivative Level** = 0 or 1, unassigned elements of CJAC are not treated as constant; they are estimated by finite differences, at non-trivial expense. If the user does not supply a value for the optional parameter **Difference Interval** (the default; see Section 11.2), an interval for each element of x is computed automatically at the start of the optimization. The automatic procedure can usually identify constant elements of CJAC, which are then computed once only by finite differences.

See also the optional parameter **Verify** in Section 11.2.

On final exit: if NCNLN > 0, CJAC contains the Jacobian matrix of the nonlinear constraint functions at the final iterate, i.e., CJAC(i, j) contains the partial derivative of the ith constraint function with respect to the jth variable, for $i = 1, 2, \ldots$,NCNLN; $j = 1, 2, \ldots$,N.

If NCNLN = 0 then the array CJAC is not referenced.

15: CLAMDA(N+NCLIN+NCNLN) — ***real*** array *Input/Output*

On initial entry: CLAMDA need not be set if the (default) **Cold Start** option is used.

If the **Warm Start** option has been chosen (see Section 11.2), CLAMDA(j) must contain a multiplier estimate for each nonlinear constraint with a sign that matches the status of the constraint specified by the ISTATE array (as above), for j = N+NCLIN+1,N+NCLIN+2,. . .,N+NCLIN+NCNLN. The remaining elements need not be set. Note that if the jth constraint is defined as 'inactive' by the initial value of the ISTATE array (i.e. ISTATE(j) = 0), CLAMDA(j) should be zero; if the jth constraint is an inequality active at its lower bound (i.e. ISTATE(j) = 1), CLAMDA(j) should be non-negative; if the jth constraint is an inequality active at its upper bound (i.e. ISTATE(j) = 2), CLAMDA(j) should be non-positive. If necessary, the routine will modify CLAMDA to match these rules.

On final exit: the values of the QP multipliers from the last QP subproblem. CLAMDA(j) should be non-negative if ISTATE(j) = 1 and non-positive if ISTATE(j) = 2.

16: OBJF — ***real*** *Input/Output*

On initial entry: OBJF need not be set.

On intermediate re-entry: if IREVCM = 1 or 3, OBJF must be set to the value of the objective function at x.

On final exit: the value of the objective function at the final iterate.

17: OBJGRD(N) — ***real*** array *Input/Output*

On initial entry: OBJGRD need not be set.

On intermediate re-entry: if IREVCM = 2 or 3, OBJGRD must contain the available elements of the gradient evaluated at x.

See also the optional parameter **Verify** in Section 11.2.

On final exit: the gradient of the objective function at the final iterate (or its finite difference approximation).

18: R(LDR,N) — ***real*** array *Input/Output*

On initial entry: R need not be initialised if the (default) **Cold Start** option is used.

If the **Warm Start** option has been chosen (see Section 11.2), R must contain the upper triangular Cholesky factor R of the initial approximation of the Hessian of the Lagrangian function, with the variables in the natural order. Elements not in the upper triangular part of R are assumed to be zero and need not be assigned.

On final exit: if **Hessian** = **No** (the default; see Section 11.2), R contains the upper triangular Cholesky factor R of $Q^T\tilde{H}Q$, an estimate of the transformed and re-ordered Hessian of the Lagrangian at x (see (6) in Section 10.1). If **Hessian** = **Yes**, R contains the upper triangular Cholesky factor R of H, the approximate (untransformed) Hessian of the Lagrangian, with the variables in the natural order.

19: X(N) — ***real*** array *Input/Output*

On initial entry: an initial estimate of the solution.

On intermediate exit: the point x at which the objective function, constraint functions or their derivatives are to be evaluated.

On final exit: the final estimate of the solution.

20: NEEDC(∗) — INTEGER array *Output*

Note: the dimension of the array NEEDC must be at least max(1,NCNLN).

On intermediate exit: if IREVCM ≥ 4, NEEDC specifies the indices of the elements of C and/or CJAC that must be assigned. If NEEDC(i) > 0, then the ith element of C and/or the available elements of the ith row of CJAC must be evaluated at x.

21: IWORK(LIWORK) — INTEGER array *Workspace*

22: LIWORK — INTEGER *Input*

On initial entry: the dimension of the array IWORK as declared in the (sub)program from which E04UFF is called.

Constraint: LIWORK ≥ 3×N+ NCLIN + 2×NCNLN.

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

24: LWORK — INTEGER *Input*

On initial entry: the dimension of the array WORK as declared in the (sub)program from which E04UFF is called.

Constraints:

if NCNLN = 0 and NCLIN = 0, then LWORK ≥ 21×N + 2;
if NCNLN = 0 and NCLIN > 0, then LWORK ≥ 2×N^2 + 21×N + 11×NCLIN + 2;
if NCNLN > 0 and NCLIN ≥ 0, then LWORK ≥ 2×N^2 + N×NCLIN + 2×N×NCNLN + 21×N + 11×NCLIN + 22×NCNLN + 1.

The amounts of workspace provided and required are (by default) output on the current advisory message unit (as defined by X04ABF). As an alternative to computing LIWORK and LWORK from the formulas given above, the user may prefer to obtain appropriate values from the output of a preliminary run with LIWORK and LWORK set to 1. (E04UFF will then terminate with IFAIL = 9.)

25: IFAIL — INTEGER *Input/Output*

On initial 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 final 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 ≠ 0 on final exit, users are recommended to set IFAIL to −1 before initial entry. **It is then essential to test the value of IFAIL on final exit**.

E04UFF returns with IFAIL = 0 if the iterates have converged to a point x that satisfies the first-order Kuhn–Tucker (see Section 10.1) conditions to the accuracy requested by the optional parameter **Optimality Tolerance** (default value $=\epsilon_R^{0.8}$, where ϵ_R is the value of the optional parameter **Function Precision** (default value $=\epsilon^{0.9}$, where ϵ is the ***machine precision***; see Section 11.2)), i.e., the projected gradient and active constraint residuals are negligible at x.

The user should check whether the following four conditions are satisfied:

(i) the final value of `Norm Gz` (see Section 8.1) is significantly less than that at the starting point;
(ii) during the final major iterations, the values of `Step` and `Mnr` (see Section 8.1) are both one;
(iii) the last few values of both `Norm Gz` and `Violtn` (see Section 8.1) become small at a fast linear rate; and
(iv) `Cond Hz` (see Section 8.1) is small.

If all these conditions hold, x is almost certainly a local minimum of (1) .

6 Error Indicators and Warnings

If on entry IFAIL = 0 or −1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings specified by the routine:

IFAIL < 0

A negative value of IFAIL indicates an exit from E04UFF because the user set IREVCM< 0 during an intermediate exit. The value of IFAIL will be the same as the user's setting of IREVCM.

IFAIL = 1

The final iterate x satisfies the first-order Kuhn–Tucker conditions (see Section 10.1) to the accuracy requested, but the sequence of iterates has not yet converged. E04UFF was terminated because no further improvement could be made in the merit function (see Section 8.1).

This value of IFAIL may occur in several circumstances. The most common situation is that the user asks for a solution with accuracy that is not attainable with the given precision of the problem (as specified by the optional parameter **Function Precision** (default value = $\epsilon^{0.9}$, where ϵ is the ***machine precision***; see Section 11.2)). This condition will also occur if, by chance, an iterate is an 'exact' Kuhn–Tucker point, but the change in the variables was significant at the previous iteration. (This situation often happens when minimizing very simple functions, such as quadratics.)

If the four conditions listed in Section 5 for IFAIL = 0 are satisfied, x is likely to be a solution of (1) even if IFAIL = 1.

IFAIL = 2

E04UFF has terminated without finding a feasible point for the linear constraints and bounds, which means that either no feasible point exists for the given value of the optional parameter **Linear Feasibility Tolerance** (default value = $\sqrt{\epsilon}$, where ϵ is the ***machine precision***; see Section 11.2), or no feasible point could be found in the number of iterations specified by the optional parameter **Minor Iteration Limit** (default value = $\max(50, 3(n + n_L + n_N))$; see Section 11.2). The user should check that there are no constraint redundancies. If the data for the constraints are accurate only to an absolute precision σ, the user should ensure that the value of the optional parameter **Linear Feasibility Tolerance** is greater than σ. For example, if all elements of A_L are of order unity and are accurate to only three decimal places, **Linear Feasibility Tolerance** should be at least 10^{-3}.

IFAIL = 3

No feasible point could be found for the nonlinear constraints. The problem may have no feasible solution. This means that there has been a sequence of QP subproblems for which no feasible point could be found (indicated by `I` at the end of each line of intermediate printout produced by the major iterations; see Section 8.1). This behaviour will occur if there is no feasible point for the nonlinear constraints. (However, there is no general test that can determine whether a feasible point exists for a set of nonlinear constraints.) If the infeasible subproblems occur from the very first major iteration, it is highly likely that no feasible point exists. If infeasibilities occur when earlier subproblems have been feasible, small constraint inconsistencies may be present. The user should check the validity of constraints with negative values of ISTATE. If the user is convinced that a feasible point does exist, E04UFF should be restarted at a different starting point.

IFAIL = 4

The limiting number of iterations (as determined by the optional parameter **Major Iteration Limit** (default value = $\max(50, 3(n + n_L) + 10n_N)$; see Section 11.2)) has been reached.

If the algorithm appears to be making satisfactory progress, then **Major Iteration Limit** may be too small. If so, either increase its value and rerun E04UFF or, alternatively, rerun E04UFF using the **Warm Start** option (see Section 11.2). If the algorithm seems to be making little or no progress however, then the user should check for incorrect gradients or ill-conditioning as described below under IFAIL = 6.

Note that ill-conditioning in the working set is sometimes resolved automatically by the algorithm, in which case performing additional iterations may be helpful. However, ill-conditioning in the Hessian approximation tends to persist once it has begun, so that allowing additional iterations without altering R is usually inadvisable. If the quasi-Newton update of the Hessian approximation was reset during the latter major iterations (i.e., an `R` occurs at the end of each line of intermediate printout; see Section 8.1), it may be worthwhile to try a **Warm Start** at the final point as suggested above.

IFAIL = 5

Not used by this routine.

IFAIL = 6

x does not satisfy the first-order Kuhn–Tucker conditions (see Section 10.1), and no improved point for the merit function (see Section 8.1) could be found during the final line search.

This sometimes occurs because an overly stringent accuracy has been requested, i.e., the value of the optional parameter **Optimality Tolerance** (default value = $\epsilon_R^{0.8}$, where ϵ_R is the value of the optional parameter **Function Precision** (default value = $\epsilon^{0.9}$, where ϵ is the ***machine precision***; see Section 11.2)) is too small. In this case the user should apply the four tests described above under IFAIL = 0 to determine whether or not the final solution is acceptable (see Gill *et al.* [10], for a discussion of the attainable accuracy).

If many iterations have occurred in which essentially no progress has been made and E04UFF has failed completely to move from the initial point, then values set by the calling program for the objective or constraint functions or their derivatives during intermediate exits may be incorrect. The user should refer to comments below under IFAIL = 7 and check the gradients using the optional parameter **Verify** (default value = 0; see Section 11.2). Unfortunately, there may be small errors in the objective and constraint gradients that cannot be detected by the verification process. Finite difference approximations to first derivatives are catastrophically affected by even small inaccuracies. An indication of this situation is a dramatic alteration in the iterates if the finite difference interval is altered. One might also suspect this type of error if a switch is made to central differences even when `Norm Gz` and `Violtn` (see Section 8.1) are large.

Another possibility is that the search direction has become inaccurate because of ill-conditioning in the Hessian approximation or the matrix of constraints in the working set; either form of ill-conditioning tends to be reflected in large values of `Mnr` (the number of iterations required to solve each QP subproblem; see Section 8.1).

If the condition estimate of the projected Hessian (`Cond Hz`; see Section 8.1) is extremely large, it may be worthwhile rerunning E04UFF from the final point with the **Warm Start** option (see Section 11.2). In this situation, ISTATE and CLAMDA should be left unaltered and R should be reset to the identity matrix.

If the matrix of constraints in the working set is ill-conditioned (i.e., `Cond T` is extremely large; see Section 12), it may be helpful to run E04UFF with a relaxed value of the **Feasibility Tolerance** (default value = $\sqrt{\epsilon}$, where ϵ is the ***machine precision***; see Section 11.2). (Constraint dependencies are often indicated by wide variations in size in the diagonal elements of the matrix T, whose diagonals will be printed if **Major Print Level** ≥ 30 (default value = 10; see Section 11.2)).

IFAIL = 7

The user-provided derivatives of the objective function and/or nonlinear constraints appear to be incorrect.

Large errors were found in the derivatives of the objective function and/or nonlinear constraints. This value of IFAIL will occur if the verification process indicated that at least one gradient or Jacobian element had no correct figures. The user should refer to the printed output to determine which elements are suspected to be in error.

As a first-step, the user should check that the code for the objective and constraint values is correct – for example, by computing the function at a point where the correct value is known. However, care should be taken that the chosen point fully tests the evaluation of the function. It is remarkable how often the values $x = 0$ or $x = 1$ are used in such a test, and how often the special properties of these numbers make the test meaningless.

Special care should be used in the test if computation of the objective function involves subsidiary data communicated in COMMON storage. Although the first evaluation of the function may be correct, subsequent calculations may be in error because some of the subsidiary data has accidently been overwritten.

Gradient checking will be ineffective if the objective function uses information computed by the constraints, since they are not necessarily computed prior to each function evaluation.

Errors in programming the function may be quite subtle in that the function value is 'almost' correct. For example, the function may not be accurate to full precision because of the inaccurate calculation of a subsidiary quantity, or the limited accuracy of data upon which the function depends. A common error on machines where numerical calculations are usually performed in double precision is to include even one single precision constant in the calculation of the function; since some compilers do not convert such constants to double precision, half the correct figures may be lost by such a seemingly trivial error.

IFAIL = 8

Not used by this routine.

IFAIL = 9

An input parameter is invalid.

Overflow

If the printed output before the overflow error contains a warning about serious ill-conditioning in the working set when adding the jth constraint, it may be possible to avoid the difficulty by increasing the magnitude of the optional parameter **Linear Feasibility Tolerance** (default value $= \sqrt{\epsilon}$, where ϵ is the ***machine precision***; see Section 11.2) and/or the optional parameter **Nonlinear Feasibility Tolerance** (default value $= \epsilon^{0.33}$ or $\sqrt{\epsilon}$; see Section 11.2), and rerunning the program. If the message recurs even after this change, the offending linearly dependent constraint (with index 'j') must be removed from the problem. If overflow occurs during function or constraint evaluation (e.g., if the nonlinear functions involve exponentials or singularities), it may help to specify tighter bounds for some of the variables (i.e., reduce the gap between the appropriate l_j and u_j).

7 Accuracy

If IFAIL = 0 on final exit, then the vector returned in the array X is an estimate of the solution to an accuracy of approximately **Optimality Tolerance** (default value $= \epsilon^{0.8}$, where ϵ is the ***machine precision***; see Section 11.2).

8 Further Comments

8.1 Description of Printed Output

This section describes the (default) intermediate printout and final printout produced by E04UFF. The intermediate printout is a subset of the monitoring information produced by the routine at every iteration (see Section 12). The level of printed output can be controlled by the user (see the description of the optional parameter **Major Print Level** in Section 11.2). Note that the intermediate printout and final printout are produced only if **Major Print Level** ≥ 10 (the default).

The following line of summary output (< 80 characters) is produced at every major iteration. In all cases, the values of the quantities printed are those in effect *on completion* of the given iteration.

`Maj` is the major iteration count.

`Mnr` is the number of minor iterations required by the feasibility and optimality phases of the QP subproblem. Generally, `Mnr` will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution (see Section 10).

Note that `Mnr` may be greater than the **Minor Iteration Limit** (default value $= \max(50, 3(n + n_L + n_N))$; see Section 11.2) if some iterations are required for the feasibility phase.

`Step` is the step taken along the computed search direction. On reasonably well-behaved problems, the unit step will be taken as the solution is approached.

`Merit Function` is the value of the augmented Lagrangian merit function (12) at the current iterate. This function will decrease at each iteration unless it was necessary to increase the penalty parameters (see Section 10.3). As the solution is approached, `Merit Function` will converge to the value of the objective function at the solution.

If the QP subproblem does not have a feasible point (signified by `I` at the end of the current output line), the merit function is a large multiple of the constraint violations, weighted by the penalty parameters. During a sequence of major iterations with infeasible subproblems, the sequence of `Merit Function` values will decrease monotonically until either a feasible subproblem is obtained or E04UFF terminates with IFAIL = 3 (no feasible point could be found for the nonlinear constraints).

If no nonlinear constraints are present (i.e., NCNLN = 0), this entry contains `Objective`, the value of the objective function $F(x)$. The objective function will decrease monotonically to its optimal value when there are no nonlinear constraints.

`Norm Gz` is $\|Z^T g_{\mathrm{FR}}\|$, the Euclidean norm of the projected gradient (see Section 10.2). `Norm Gz` will be approximately zero in the neighbourhood of a solution.

`Violtn` is the Euclidean norm of the residuals of constraints that are violated or in the predicted active set (not printed if NCNLN is zero). `Violtn` will be approximately zero in the neighbourhood of a solution.

`Cond Hz` is a lower bound on the condition number of the projected Hessian approximation H_Z ($H_Z = Z^T H_{\mathrm{FR}} Z = R_Z^T R_Z$; see (6) in Section 10.1 and (11) in Section 10.2). The larger this number, the more difficult the problem.

`M` is printed if the quasi-Newton update has been modified to ensure that the Hessian approximation is positive-definite (see Section 10.4).

`I` is printed if the QP subproblem has no feasible point.

`C` is printed if central differences have been used to compute the unspecified objective and constraint gradients. If the value of `Step` is zero, the switch to central differences was made because no lower point could be found in the line search. (In this case, the QP subproblem is re-solved with the central difference gradient and Jacobian.) If the value of `Step` is non-zero, central differences were computed because `Norm Gz` and `Violtn` imply that x is close to a Kuhn–Tucker point (see Section 10.1).

`L` is printed if the line search has produced a relative change in x greater than the value defined by the optional parameter **Step Limit** (default value = 2.0; see Section 11.2). If this output occurs frequently during later iterations of the run, **Step Limit** should be set to a larger value.

`R` is printed if the approximate Hessian has been refactorized. If the diagonal condition estimator of R indicates that the approximate Hessian is badly conditioned, the approximate Hessian is refactorized using column interchanges. If necessary, R is modified so that its diagonal condition estimator is bounded.

The final printout includes a listing of the status of every variable and constraint.

The following describes the printout for each variable. A full stop (.) is printed for any numerical value that is zero.

`Varbl` gives the name (`V`) and index j, for $j = 1, 2, \ldots, n$ of the variable.

`State` gives the state of the variable (`FR` if neither bound is in the active set, `EQ` if a fixed variable, `LL` if on its lower bound, `UL` if on its upper bound). If `Value` lies outside the upper or lower bounds by more than the **Feasibility Tolerance** (default value $= \sqrt{\epsilon}$, where ϵ is the ***machine precision***; see Section 11.2), `State` will be `++` or `--` respectively. (The latter situation can occur only when there is no feasible point for the bounds and linear constraints.)

A key is sometimes printed before `State` to give some additional information about the state of a variable.

- `A` *Alternative optimum possible.* The variable is active at one of its bounds, but its Lagrange multiplier is essentially zero. This means that if the variable were allowed to start moving away from its bound, there would be no change to the objective function. The values of the other free variables *might* change, giving a genuine alternative solution. However, if there are any degenerate variables (labelled `D`), the actual change might prove to be zero, since one of them could encounter a bound immediately. In either case the values of the Lagrange multipliers might also change.
- `D` *Degenerate.* The variable is free, but it is equal to (or very close to) one of its bounds.
- `I` *Infeasible.* The variable is currently violating one of its bounds by more than the **Feasibility Tolerance**.

`Value` is the value of the variable at the final iterate.

`Lower Bound` is the lower bound specified for the variable. `None` indicates that $\mathrm{BL}(j) \leq -bigbnd$.

`Upper Bound` is the upper bound specified for the variable. `None` indicates that $\mathrm{BU}(j) \geq bigbnd$.

`Lagr Mult` is the Lagrange multiplier for the associated bound. This will be zero if `State` is `FR` unless $\mathrm{BL}(j) \leq -bigbnd$ and $\mathrm{BU}(j) \geq bigbnd$, in which case the entry will be blank. If x is optimal, the multiplier should be non-negative if `State` is `LL`, and non-positive if `State` is `UL`.

`Slack` is the difference between the variable `Value` and the nearer of its (finite) bounds $\mathrm{BL}(j)$ and $\mathrm{BU}(j)$. A blank entry indicates that the associated variable is not bounded (i.e., $\mathrm{BL}(j) \leq -bigbnd$ and $\mathrm{BU}(j) \geq bigbnd$).

The meaning of the printout for linear and nonlinear constraints is the same as that given above for variables, with 'variable' replaced by 'constraint', $\mathrm{BL}(j)$ and $\mathrm{BU}(j)$ are replaced by $\mathrm{BL}(n+j)$ and $\mathrm{BU}(n+j)$ respectively, and with the following changes in the heading:

`L Con` gives the name (`L`) and index j, for $j = 1, 2, \ldots, n_L$ of the linear constraint.

`N Con` gives the name (`N`) and index $(j - n_L)$, for $j = n_L + 1, n_L + 2, \ldots, n_L + n_N$ of the nonlinear constraint.

Note that movement off a constraint (as opposed to a variable moving away from its bound) can be interpreted as allowing the entry in the `Slack` column to become positive.

Numerical values are output with a fixed number of digits; they are not guaranteed to be accurate to this precision.

9 Example

This is based on Problem 71 in Hock and Schittkowski [12] and involves the minimization of the nonlinear function

$$F(x) = x_1x_4(x_1 + x_2 + x_3) + x_3$$

subject to the bounds

$$\begin{aligned} 1 \leq x_1 \leq 5 \\ 1 \leq x_2 \leq 5 \\ 1 \leq x_3 \leq 5 \\ 1 \leq x_4 \leq 5 \end{aligned}$$

to the general linear constraint

$$x_1 + x_2 + x_3 + x_4 \leq 20,$$

and to the nonlinear constraints

$$\begin{aligned} x_1^2 + x_2^2 + x_3^2 + x_4^2 \leq 40, \\ x_1x_2x_3x_4 \geq 25. \end{aligned}$$

The initial point, which is infeasible, is

$$x_0 = (1, 5, 5, 1)^T,$$

and $F(x_0) = 16$.

The optimal solution (to five figures) is

$$x^* = (1.0, 4.7430, 3.8211, 1.3794)^T,$$

and $F(x^*) = 17.014$. One bound constraint and both nonlinear constraints are active at the solution.

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.

```
*     E04UFF Example Program Text
*     Mark 18 Release. NAG Copyright 1997.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          NMAX, NCLMAX, NCNMAX
      PARAMETER        (NMAX=10,NCLMAX=10,NCNMAX=10)
      INTEGER          LDA, LDCJ, LDR
      PARAMETER        (LDA=NCLMAX,LDCJ=NCNMAX,LDR=NMAX)
      INTEGER          LIWORK, LWORK
      PARAMETER        (LIWORK=100,LWORK=1000)
      real             ZERO, ONE, TWO
      PARAMETER        (ZERO=0.0e+0,ONE=1.0e+0,TWO=2.0e+0)
*     .. Local Scalars ..
      real             OBJF
      INTEGER          I, IFAIL, IREVCM, ITER, J, N, NCLIN, NCNLN
*     .. Local Arrays ..
      real             A(LDA,NMAX), BL(NMAX+NCLMAX+NCNMAX),
     +                 BU(NMAX+NCLMAX+NCNMAX), C(NCNMAX),
     +                 CJAC(LDCJ,NMAX), CLAMDA(NMAX+NCLMAX+NCNMAX),
     +                 OBJGRD(NMAX), R(LDR,NMAX), WORK(LWORK), X(NMAX)
      INTEGER          ISTATE(NMAX+NCLMAX+NCNMAX), IWORK(LIWORK),
     +                 NEEDC(NCNMAX)
*     .. External Subroutines ..
      EXTERNAL         E04UFF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'E04UFF Example Program Results'
```

```
*     Skip heading in data file.
      READ (NIN,*)
      READ (NIN,*) N, NCLIN, NCNLN
      IF (N.LE.NMAX .AND. NCLIN.LE.NCLMAX .AND. NCNLN.LE.NCNMAX) THEN
*
*        Read A, BL, BU and X from data file.
*
         IF (NCLIN.GT.0) READ (NIN,*) ((A(I,J),J=1,N),I=1,NCLIN)
         READ (NIN,*) (BL(I),I=1,N+NCLIN+NCNLN)
         READ (NIN,*) (BU(I),I=1,N+NCLIN+NCNLN)
         READ (NIN,*) (X(I),I=1,N)
*
*        Set all constraint Jacobian elements to zero.
*        Note that this will only work when 'Derivative Level = 3'
*        (the default; see Section 11.2).
*
         DO 40 J = 1, N
            DO 20 I = 1, NCNLN
               CJAC(I,J) = ZERO
   20       CONTINUE
   40    CONTINUE
*
*        Solve the problem.
*
         IFAIL = -1
         IREVCM = 0
*
   60    CALL E04UFF(IREVCM,N,NCLIN,NCNLN,LDA,LDCJ,LDR,A,BL,BU,ITER,
     +              ISTATE,C,CJAC,CLAMDA,OBJF,OBJGRD,R,X,NEEDC,IWORK,
     +              LIWORK,WORK,LWORK,IFAIL)
*
         IF (IFAIL.EQ.-1 .AND. IREVCM.GT.0) THEN
            IF (IREVCM.EQ.1 .OR. IREVCM.EQ.3) THEN
*              Evaluate the objective function.
               OBJF = X(1)*X(4)*(X(1)+X(2)+X(3)) + X(3)
            END IF
            IF (IREVCM.EQ.2 .OR. IREVCM.EQ.3) THEN
*              Evaluate the objective gradient.
               OBJGRD(1) = X(4)*(TWO*X(1)+X(2)+X(3))
               OBJGRD(2) = X(1)*X(4)
               OBJGRD(3) = X(1)*X(4) + ONE
               OBJGRD(4) = X(1)*(X(1)+X(2)+X(3))
            END IF
*
            IF (IREVCM.EQ.4 .OR. IREVCM.EQ.6) THEN
*              Evaluate the nonlinear constraint functions.
               IF (NEEDC(1).GT.0) C(1) = X(1)**2 + X(2)**2 + X(3)**2 +
     +                                   X(4)**2
               IF (NEEDC(2).GT.0) C(2) = X(1)*X(2)*X(3)*X(4)
            END IF
            IF (IREVCM.EQ.5 .OR. IREVCM.EQ.6) THEN
*              Evaluate the constraint Jacobian.
               IF (NEEDC(1).GT.0) THEN
                  CJAC(1,1) = TWO*X(1)
                  CJAC(1,2) = TWO*X(2)
                  CJAC(1,3) = TWO*X(3)
                  CJAC(1,4) = TWO*X(4)
               END IF
```

```
               IF (NEEDC(2).GT.0) THEN
                  CJAC(2,1) = X(2)*X(3)*X(4)
                  CJAC(2,2) = X(1)*X(3)*X(4)
                  CJAC(2,3) = X(1)*X(2)*X(4)
                  CJAC(2,4) = X(1)*X(2)*X(3)
               END IF
            END IF
            GO TO 60
         END IF
      END IF
*
      STOP
      END
```

9.2 Program Data

```
E04UFF Example Program Data
  4   1   2                              :Values of N, NCLIN and NCNLN
  1.0   1.0   1.0   1.0                                :End of matrix A
  1.0   1.0   1.0   1.0  -1.0E+25  -1.0E+25  25.0      :End of BL
  5.0   5.0   5.0   5.0  20.0      40.0       1.0E+25  :End of BU
  1.0   5.0   5.0   1.0                                :End of X
```

9.3 Program Results

```
E04UFF Example Program Results

*** E04UFF
*** Start of NAG Library implementation details ***

Implementation title: Generalised Base Version
           Precision: FORTRAN double precision
        Product Code: FLBAS19D
                Mark: 19A

*** End of NAG Library implementation details ***

Parameters
----------

Linear constraints.....         1       Variables..............         4
Nonlinear constraints..         2

Infinite bound size....  1.00E+20       COLD start.............
Infinite step size.....  1.00E+20       EPS (machine precision)  1.11E-16
Step limit.............  2.00E+00       Hessian................        NO

Linear feasibility.....  1.05E-08       Crash tolerance........  1.00E-02
Nonlinear feasibility..  1.05E-08       Optimality tolerance...  3.26E-12
Line search tolerance..  9.00E-01       Function precision.....  4.37E-15

Derivative level.......         3       Monitoring file........        -1
Verify level...........         0

Major iterations limit.        50       Major print level......        10
Minor iterations limit.        50       Minor print level......         0
```

```
Workspace provided is     IWORK(     100),  WORK(    1000).
To solve problem we need  IWORK(      17),  WORK(     192).


Verification of the constraint gradients.
-----------------------------------------

The constraint Jacobian seems to be ok.

The largest relative error was    2.29E-07  in constraint    2



Verification of the objective gradients.
----------------------------------------

The objective gradients seem to be ok.

Directional derivative of the objective    8.15250000E-01
Difference approximation                   8.15249734E-01


 Maj  Mnr    Step Merit Function Norm Gz  Violtn Cond Hz
   0    4 0.0E+00   1.738281E+01 7.1E-01 1.2E+01 1.0E+00
   1    1 1.0E+00   1.703169E+01 4.6E-02 1.9E+00 1.0E+00
   2    1 1.0E+00   1.701442E+01 2.1E-02 8.8E-02 1.0E+00
   3    1 1.0E+00   1.701402E+01 3.1E-04 5.4E-04 1.0E+00
   4    1 1.0E+00   1.701402E+01 7.0E-06 9.9E-08 1.0E+00
   5    1 1.0E+00   1.701402E+01 1.1E-08 4.6E-11 1.0E+00

Exit from NP problem after     5 major iterations,
                               9 minor iterations.


Varbl State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack

V   1    LL    1.00000       1.00000       5.00000       1.088         .
V   2    FR    4.74300       1.00000       5.00000           .      0.2570
V   3    FR    3.82115       1.00000       5.00000           .       1.179
V   4    FR    1.37941       1.00000       5.00000           .      0.3794


L Con State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack

L   1    FR    10.9436          None       20.0000           .       9.056


N Con State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack

N   1    UL    40.0000          None       40.0000     -0.1615     -3.5264E-11
N   2    LL    25.0000       25.0000          None      0.5523     -2.8791E-11

Exit E04UFF - Optimal solution found.

Final objective value =    17.01402
```

The remainder of this document is intended for more advanced users. Section 10 contains a detailed description of the algorithm which may be needed in order to understand Section 11 and Section 12. Section 11 describes the optional parameters which may be set by calls to E04UDF and/or E04UEF. Section 12 describes the quantities which can be requested to monitor the course of the computation.

10 Algorithmic Details

This section contains a detailed description of the method used by E04UFF.

10.1 Overview

E04UFF is essentially identical to the subroutine NPSOL described in Gill *et al.* [9].

At a solution of (1), some of the constraints will be *active*, i.e., satisfied exactly. An active simple bound constraint implies that the corresponding variable is *fixed* at its bound, and hence the variables are partitioned into *fixed* and *free* variables. Let C denote the m by n matrix of gradients of the active general linear and nonlinear constraints. The number of fixed variables will be denoted by n_{FX}, with n_{FR} ($n_{\mathrm{FR}} = n - n_{\mathrm{FX}}$) the number of free variables. The subscripts 'FX' and 'FR' on a vector or matrix will denote the vector or matrix composed of the elements corresponding to fixed or free variables.

A point x is a *first-order Kuhn–Tucker point* for (1) (see, e.g.,, Powell [13]) if the following conditions hold:

(i) x is feasible;
(ii) there exist vectors ξ and λ (*the Lagrange multiplier vectors for the bound and general constraints*) such that
$$g = C^T\lambda + \xi \tag{2}$$
where g is the gradient of F evaluated at x, and $\xi_j = 0$ if the jth variable is free.
(iii) The Lagrange multiplier corresponding to an inequality constraint active at its lower bound must be non-negative, and non-positive for an inequality constraint active at its upper bound.

Let Z denote a matrix whose columns form a basis for the set of vectors orthogonal to the rows of C_{FR}; i.e., $C_{\mathrm{FR}}Z = 0$. An equivalent statement of the condition (2) in terms of Z is

$$Z^T g_{\mathrm{FR}} = 0.$$

The vector $Z^T g_{\mathrm{FR}}$ is termed the *projected gradient* of F at x. Certain additional conditions must be satisfied in order for a first-order Kuhn–Tucker point to be a solution of (1) (see, e.g.,, Powell [13]).

E04UFF implements a sequential quadratic programming (SQP) method. For an overview of SQP methods, see, for example, Fletcher [4], Gill *et al.* [10] and Powell [14].

The basic structure of E04UFF involves *major* and *minor* iterations. The major iterations generate a sequence of iterates $\{x_k\}$ that converge to x^*, a first-order Kuhn–Tucker point of (1). At a typical major iteration, the new iterate $\bar{x}$ is defined by
$$\bar{x} = x + \alpha p \tag{3}$$
where x is the current iterate, the non-negative scalar α is the *step length,* and p is the *search direction.* (For simplicity, we shall always consider a typical iteration and avoid reference to the index of the iteration.) Also associated with each major iteration are estimates of the Lagrange multipliers and a prediction of the active set.

The search direction p in (3) is the solution of a quadratic programming subproblem of the form

$$\underset{p}{\text{Minimize}}\ g^T p + \tfrac{1}{2}p^T H p \ \text{ subject to } \ \bar{l} \leq \begin{Bmatrix} p \\ A_L p \\ A_N p \end{Bmatrix} \leq \bar{u}, \tag{4}$$

where g is the gradient of F at x, the matrix H is a positive-definite quasi-Newton approximation to the Hessian of the Lagrangian function (see Section 10.4), and A_N is the Jacobian matrix of c evaluated at x. (Finite difference estimates may be used for g and A_N; see the optional parameter **Derivative Level** in

Section 11.2.) Let l in (1) be partitioned into three sections: l_B, l_L and l_N, corresponding to the bound, linear and nonlinear constraints. The vector $\bar{l}$ in (4) is similarly partitioned, and is defined as

$$\bar{l}_B = l_B - x, \quad \bar{l}_L = l_L - A_L x, \text{ and } \bar{l}_N = l_N - c,$$

where c is the vector of nonlinear constraints evaluated at x. The vector $\bar{u}$ is defined in an analogous fashion.

The estimated Lagrange multipliers at each major iteration are the Lagrange multipliers from the subproblem (4) (and similarly for the predicted active set). (The numbers of bounds, general linear and nonlinear constraints in the QP active set are the quantities `Bnd`, `Lin` and `Nln` in the monitoring file output of E04UFF; see Section 12.) In E04UFF, (4) is solved using E04NCF. Since solving a quadratic program is itself an iterative procedure, the *minor* iterations of E04UFF are the iterations of E04NCF. (More details about solving the subproblem are given in Section 10.2.)

Certain matrices associated with the QP subproblem are relevant in the major iterations. Let the subscripts 'FX' and 'FR' refer to the *predicted* fixed and free variables, and let C denote the m by n matrix of gradients of the general linear and nonlinear constraints in the predicted active set. First, we have available the TQ factorization of C_{FR}:

$$C_{\mathrm{FR}} Q_{\mathrm{FR}} = (0 \;\; T), \tag{5}$$

where T is a non-singular m by m reverse-triangular matrix (i.e., $t_{ij} = 0$ if $i+j < m$), and the non-singular n_{FR} by n_{FR} matrix Q_{FR} is the product of orthogonal transformations (see Gill *et al.* [5]). Second, we have the upper triangular Cholesky factor R of the *transformed and re-ordered* Hessian matrix

$$R^T R = H_Q \equiv Q^T \tilde{H} Q, \tag{6}$$

where $\tilde{H}$ is the Hessian H with rows and columns permuted so that the free variables are first, and Q is the n by n matrix

$$Q = \begin{pmatrix} Q_{\mathrm{FR}} & \\ & I_{\mathrm{FX}} \end{pmatrix} \tag{7}$$

with I_{FX} the identity matrix of order n_{FX}. If the columns of Q_{FR} are partitioned so that

$$Q_{\mathrm{FR}} = (Z \;\; Y),$$

the n_Z ($n_Z \equiv n_{\mathrm{FR}} - m$) columns of Z form a basis for the null space of C_{FR}. The matrix Z is used to compute the projected gradient $Z^T g_{\mathrm{FR}}$ at the current iterate. (The values `Nz` and `Norm Gz` printed by E04UFF give n_Z and the norm of $Z^T g_{\mathrm{FR}}$; see Section 12.)

A theoretical characteristic of SQP methods is that the predicted active set from the QP subproblem (4) is identical to the correct active set in a neighbourhood of x^*. In E04UFF, this feature is exploited by using the QP active set from the previous iteration as a prediction of the active set for the next QP subproblem, which leads in practice to optimality of the subproblems in only one iteration as the solution is approached. Separate treatment of bound and linear constraints in E04UFF also saves computation in factorizing C_{FR} and H_Q.

Once p has been computed, the major iteration proceeds by determining a step length α that produces a 'sufficient decrease' in an augmented Lagrangian *merit function* (see Section 10.3). Finally, the approximation to the transformed Hessian matrix H_Q is updated using a modified BFGS quasi-Newton update (see Section 10.4) to incorporate new curvature information obtained in the move from x to $\bar{x}$.

On entry to E04UFF, an iterative procedure from E04NCF is executed, starting with the user-provided initial point, to find a point that is feasible with respect to the bounds and linear constraints (using the tolerance specified by **Linear Feasibility Tolerance**; see Section 11.2). If no feasible point exists for the bound and linear constraints, (1) has no solution and E04UFF terminates. Otherwise, the problem functions will thereafter be evaluated only at points that are feasible with respect to the bounds and linear constraints. The only exception involves variables whose bounds differ by an amount comparable to the finite difference interval (see the discussion of **Difference Interval** in Section 11.2). In contrast to the bounds and linear constraints, it must be emphasised that *the nonlinear constraints will not generally be satisfied until an optimal point* is reached.

Facilities are provided to check whether the user-provided gradients appear to be correct (see the optional parameter **Verify** in Section 11.2). In general, the check is provided at the first point that is feasible with

respect to the linear constraints and bounds. However, the user may request that the check be performed at the initial point.

In summary, the method of E04UFF first determines a point that satisfies the bound and linear constraints. Thereafter, each iteration includes:

(a) the solution of a quadratic programming subproblem;
(b) a line search with an augmented Lagrangian merit function; and
(c) a quasi-Newton update of the approximate Hessian of the Lagrangian function.

These three procedures are described in more detail in Section 10.2 to Section 10.4.

10.2 Solution of the Quadratic Programming Subproblem

The search direction p is obtained by solving (4) using E04NCF (see Gill *et al.* [7]), which was specifically designed to be used within an SQP algorithm for nonlinear programming.

E04NCF is based on a two-phase (primal) quadratic programming method. The two phases of the method are: finding an initial feasible point by minimizing the sum of infeasibilities (the *feasibility phase*), and minimizing the quadratic objective function within the feasible region (the *optimality phase*). The computations in both phases are performed by the same subroutines. The two-phase nature of the algorithm is reflected by changing the function being minimized from the sum of infeasibilities to the quadratic objective function.

In general, a quadratic program must be solved by iteration. Let p denote the current estimate of the solution of (4); the new iterate $\bar{p}$ is defined by

$$\bar{p} = p + \sigma d \tag{8}$$

where, as in (3), σ is a non-negative step length and d is a search direction.

At the beginning of each iteration of E04NCF, a *working* set is defined of constraints (general and bound) that are satisfied exactly. The vector d is then constructed so that the values of constraints in the working set remain *unaltered* for any move along d. For a bound constraint in the working set, this property is achieved by setting the corresponding element of d to zero, i.e., by fixing the variable at its bound. As before, the subscripts 'FX' and 'FR' denote selection of the elements associated with the fixed and free variables.

Let C denote the sub-matrix of rows of

$$\begin{pmatrix} A_L \\ A_N \end{pmatrix}$$

corresponding to general constraints in the working set. The general constraints in the working set will remain unaltered if

$$C_{\mathrm{FR}} d_{\mathrm{FR}} = 0, \tag{9}$$

which is equivalent to defining d_{FR} as

$$d_{\mathrm{FR}} = Z d_Z \tag{10}$$

for some vector d_Z, where Z is the matrix associated with the TQ factorization (5) of C_{FR}.

The definition of d_Z in (10) depends on whether the current p is feasible. If not, d_Z is zero except for an element γ in the jth position, where j and γ are chosen so that the sum of infeasibilities is decreasing along d. (For further details, see Gill *et al.* [7].) In the feasible case, d_Z satisfies the equations

$$R_Z^T R_Z d_Z = -Z^T q_{\mathrm{FR}}, \tag{11}$$

where R_Z is the Cholesky factor of $Z^T H_{\mathrm{FR}} Z$ and q is the gradient of the quadratic objective function ($q = g + Hp$). (The vector $Z^T q_{\mathrm{FR}}$ is the projected gradient of the QP.) With (11), $p + d$ is the minimizer of the quadratic objective function subject to treating the constraints in the working set as equalities.

If the QP projected gradient is zero, the current point is a constrained stationary point in the subspace defined by the working set. During the feasibility phase, the projected gradient will usually be zero only at a vertex (although it may vanish at non-vertices in the presence of constraint dependencies). During the optimality phase, a zero projected gradient implies that p minimizes the quadratic objective function when the constraints in the working set are treated as equalities. In either case, Lagrange multipliers are

computed. Given a positive constant δ of the order of the ***machine precision***, the Lagrange multiplier μ_j corresponding to an inequality constraint in the working set is said to be *optimal* if $\mu_j \leq \delta$ when the jth constraint is at its *upper bound*, or if $\mu_j \geq -\delta$ when the associated constraint is at its *lower bound*. If any multiplier is non-optimal, the current objective function (either the true objective or the sum of infeasibilities) can be reduced by deleting the corresponding constraint from the working set.

If optimal multipliers occur during the feasibility phase and the sum of infeasibilities is non-zero, no feasible point exists. The QP algorithm will then continue iterating to determine the minimum sum of infeasibilities. At this point, the Lagrange multiplier μ_j will satisfy $-(1+\delta) \leq \mu_j \leq \delta$ for an inequality constraint at its upper bound, and $-\delta \leq \mu_j \leq (1+\delta)$ for an inequality at its lower bound. The Lagrange multiplier for an equality constraint will satisfy $|\mu_j| \leq 1+\delta$.

The choice of step length σ in the QP iteration (8) is based on remaining feasible with respect to the satisfied constraints. During the optimality phase, if $p+d$ is feasible, σ will be taken as unity. (In this case, the projected gradient at $\bar{p}$ will be zero.) Otherwise, σ is set to σ_{M}, the step to the 'nearest' constraint, which is added to the working set at the next iteration.

Each change in the working set leads to a simple change to C_{FR}: if the status of a general constraint changes, a *row* of C_{FR} is altered; if a bound constraint enters or leaves the working set, a *column* of C_{FR} changes. Explicit representations are recurred of the matrices T, Q_{FR} and R, and of the vectors $Q^T q$ and $Q^T g$.

10.3 The Merit Function

After computing the search direction as described in Section 10.2, each major iteration proceeds by determining a step length α in (3) that produces a 'sufficient decrease' in the augmented Lagrangian merit function

$$L(x,\lambda,s) = F(x) - \sum_i \lambda_i(c_i(x)-s_i) + \tfrac{1}{2}\sum_i \rho_i(c_i(x)-s_i)^2, \tag{12}$$

where x, λ and s vary during the line search. The summation terms in (12) involve only the *nonlinear* constraints. The vector λ is an estimate of the Lagrange multipliers for the nonlinear constraints of (1). The non-negative *slack variables* $\{s_i\}$ allow nonlinear inequality constraints to be treated without introducing discontinuities. The solution of the QP subproblem (4) provides a vector triple that serves as a direction of search for the three sets of variables. The non-negative vector ρ of *penalty parameters* is initialised to zero at the beginning of the first major iteration. Thereafter, selected elements are increased whenever necessary to ensure descent for the merit function. Thus, the sequence of norms of ρ (the printed quantity `Penalty`; see Section 12) is generally non-decreasing, although each ρ_i may be reduced a limited number of times.

The merit function (12) and its global convergence properties are described in Gill *et al.* [8].

10.4 The Quasi-Newton Update

The matrix H in (4) is a *positive-definite quasi-Newton* approximation to the Hessian of the Lagrangian function. (For a review of quasi-Newton methods, see Dennis and Schnabel [3].) At the end of each major iteration, a new Hessian approximation $\bar{H}$ is defined as a rank-two modification of H. In E04UFF, the BFGS (Broyden–Fletcher–Goldfarb–Shanno) quasi-Newton update is used:

$$\bar{H} = H - \frac{1}{s^T H s} H s s^T H + \frac{1}{y^T s} y y^T, \tag{13}$$

where $s = \bar{x} - x$ (the change in x).

In E04UFF, H is required to be positive-definite. If H is positive-definite, $\bar{H}$ defined by (13) will be positive-definite if and only if $y^T s$ is positive (see, e.g., Dennis and Moré [1]). Ideally, y in (13) would be taken as y_L, the change in gradient of the Lagrangian function

$$y_L = \bar{g} - \bar{A}_N^T \mu_N - g + A_N^T \mu_N, \tag{14}$$

where μ_N denotes the QP multipliers associated with the nonlinear constraints of the original problem. If $y_L^T s$ is not sufficiently positive, an attempt is made to perform the update with a vector y of the form

$$y = y_L + \sum_{i=1}^{m_N} \omega_i (a_i(\hat{x})c_i(\hat{x}) - a_i(x)c_i(x)),$$

where $\omega_i \geq 0$. If no such vector can be found, the update is performed with a scaled y_L; in this case, `M` is printed to indicate that the update was modified.

Rather than modifying H itself, the Cholesky factor of the *transformed Hessian* H_Q (6) is updated, where Q is the matrix from (5) associated with the active set of the QP subproblem. The update (13) is equivalent to the following update to H_Q:

$$\bar{H}_Q = H_Q - \frac{1}{s_Q^T H_Q s_Q} H_Q s_Q s_Q^T H_Q + \frac{1}{y_Q^T s_Q} y_Q y_Q^T, \tag{15}$$

where $y_Q = Q^T y$, and $s_Q = Q^T s$. This update may be expressed as a *rank-one* update to R (see Dennis and Schnabel [2]).

11 Optional Parameters

Several optional parameters in E04UFF define choices in the problem specification or the algorithm logic. In order to reduce the number of formal parameters of E04UFF these optional parameters have associated *default values* that are appropriate for most problems. Therefore the user need only specify those optional parameters whose values are to be different from their default values.

The remainder of this section can be skipped by users who wish to use the default values for all optional parameters. A complete list of optional parameters and their default values is given in Section 11.1.

Optional parameters may be specified by calling one, or both, of E04UDF and E04UEF prior to a call to E04UFF.

E04UDF reads options from an external options file, with `Begin` and `End` as the first and last lines respectively and each intermediate line defining a single optional parameter. For example,

```
Begin
  Print Level = 1
End
```

The call

```
CALL E04UDF (IOPTNS, INFORM)
```

can then be used to read the file on unit IOPTNS. INFORM will be zero on successful exit. E04UDF should be consulted for a full description of this method of supplying optional parameters.

E04UEF can be called to supply options directly, one call being necessary for each optional parameter. For example,

```
CALL E04UEF ('Print level = 1')
```

E04UEF should be consulted for a full description of this method of supplying optional parameters.

All optional parameters not specified by the user are set to their default values. Optional parameters specified by the user are unaltered by E04UFF (unless they define invalid values) and so remain in effect for subsequent calls to E04UFF, unless altered by the user.

11.1 Optional Parameter Checklist and Default Values

For easy reference, the following list shows all the valid keywords and their default values. The symbol ϵ represents the ***machine precision*** (see X02AJF).

Optional Parameters	**Default Values**
Central difference interval	Computed automatically
Cold/Warm start	**Cold start**
Crash tolerance	0.01
Defaults	
Derivative level	3
Difference interval	Computed automatically
Feasibility tolerance	$\sqrt{\epsilon}$
Function precision	$\epsilon^{0.9}$
Hessian	**No**
Infinite bound size	10^{20}
Infinite step size	10^{20}
Line search tolerance	0.9
Linear feasibility tolerance	$\sqrt{\epsilon}$
List/Nolist	**List**
Major iteration limit	$\max(50, 3(n + n_L) + 10n_N)$
Major print level	10
Minor iteration limit	$\max(50, 3(n + n_L + n_N))$
Minor print level	0
Monitoring file	-1
Nonlinear feasibility tolerance	$\epsilon^{0.33}$ or $\sqrt{\epsilon}$
Optimality tolerance	$\epsilon_R^{0.8}$
Step limit	2.0
Start objective check	1
Start constraint check	1
Stop objective check	n
Stop constraint check	n
Verify level	0

11.2 Description of the Optional Parameters

The following list (in alphabetical order) gives the valid options. For each option, we give the keyword, any essential optional qualifiers, the default value, and the definition. The minimum abbreviation of each keyword is underlined. If no characters of an optional qualifier are underlined, the qualifier may be omitted. The letter a denotes a phrase (character string) that qualifies an option. The letters i and r denote INTEGER and ***real*** values required with certain options. The number ϵ is a generic notation for ***machine precision*** (see X02AJF), and ϵ_R denotes the relative precision of the objective function (the optional parameter **Function Precision**; see below).

Central Difference Interval r Default values are computed

If the algorithm switches to central differences because the forward-difference approximation is not sufficiently accurate, the value of r is used as the difference interval for every element of x. The switch to central differences is indicated by `C` at the end of each line of intermediate printout produced by the major iterations (see Section 8.1). The use of finite differences is discussed further below under the optional parameter **Difference Interval**.

Cold Start Default = **Cold Start**

Warm Start

This option controls the specification of the initial working set in both the procedure for finding a feasible point for the linear constraints and bounds, and in the first QP subproblem thereafter. With a **Cold Start**, the first working set is chosen by E04UFF based on the values of the variables and constraints at the initial point. Broadly speaking, the initial working set will include equality constraints and bounds or inequality constraints that violate or 'nearly' satisfy their bounds (to within **Crash Tolerance**; see below).

With a **Warm Start**, the user must set the ISTATE array and define CLAMDA and R as discussed in Section 5. ISTATE values associated with bounds and linear constraints determine the initial working

set of the procedure to find a feasible point with respect to the bounds and linear constraints. ISTATE values associated with nonlinear constraints determine the initial working set of the first QP subproblem after such a feasible point has been found. E04UFF will override the user's specification of ISTATE if necessary, so that a poor choice of the working set will not cause a fatal error. For instance, any elements of ISTATE which are set to -2, -1 or 4 will be reset to zero, as will any elements which are set to 3 when the corresponding elements of BL and BU are not equal. A warm start will be advantageous if a good estimate of the initial working set is available – for example, when E04UFF is called repeatedly to solve related problems.

Crash Tolerance r Default = 0.01

This value is used in conjunction with the optional parameter **Cold Start** (the default value) when E04UFF selects an initial working set. If $0 \leq r \leq 1$, the initial working set will include (if possible) bounds or general inequality constraints that lie within r of their bounds. In particular, a constraint of the form $a_j^T x \geq l$ will be included in the initial working set if $|a_j^T x - l| \leq r(1 + |l|)$. If $r < 0$ or $r > 1$, the default value is used.

Defaults

This special keyword may be used to reset all optional parameters to their default values.

Derivative Level i Default = 3

This parameter indicates which derivatives are provided by the user during intermediate exits. The possible choices for i are the following.

i	**Meaning**
3	All elements of the objective gradient and the constraint Jacobian are provided.
2	All elements of the constraint Jacobian are provided, but some elements of the objective gradient are not specified.
1	All elements of the objective gradient are provided, but some elements of the constraint Jacobian are not specified.
0	Some elements of both the objective gradient and the constraint Jacobian are not specified.

The value $i = 3$ should be used whenever possible, since E04UFF is more reliable (and will usually be more efficient) when all derivatives are exact.

If $i = 0$ or 2, E04UFF will estimate the unspecified elements of the objective gradient, using finite differences. The computation of finite difference approximations usually increases the total run-time, since an intermediate exit to the calling program is required for each unspecified element. Furthermore, less accuracy can be attained in the solution (see Chapter 8 of Gill *et al.* [10], for a discussion of limiting accuracy).

If $i = 0$ or 1, E04UFF will approximate unspecified elements of the constraint Jacobian. One intermediate exit is needed for each variable for which partial derivatives are not available. For example, if the Jacobian has the form

$$\begin{pmatrix} * & * & * & * \\ * & ? & ? & * \\ * & * & ? & * \\ * & * & * & * \end{pmatrix}$$

where '$*$' indicates an element provided by the user and '?' indicates an unspecified element, E04UFF will make an intermediate exit to the calling program twice: once to estimate the missing element in column 2, and again to estimate the two missing elements in column 3. (Since columns 1 and 4 are known, they require no intermediate exits for information.)

At times, central differences are used rather than forward differences, in which case twice as many intermediate exits are needed. (The switch to central differences is not under the user's control.)

If $i < 0$ or $i > 3$, the default value is used.

Difference Interval r Default values are computed

This option defines an interval used to estimate derivatives by finite differences in the following circumstances:

(a) For verifying the objective and/or constraint gradients (see the description of **Verify**, below).
(b) For estimating unspecified elements of the objective gradient or the constraint Jacobian.

In general, a derivative with respect to the jth variable is approximated using the interval δ_j, where $\delta_j = r(1 + |\hat{x}_j|)$, with $\hat{x}$ the first point feasible with respect to the bounds and linear constraints. If the functions are well scaled, the resulting derivative approximation should be accurate to $O(r)$. See Gill *et al.* [10] for a discussion of the accuracy in finite difference approximations.

If a difference interval is not specified by the user, a finite difference interval will be computed automatically for each variable by a procedure that requires up to six intermediate exits for each element. This option is recommended if the function is badly scaled or the user wishes to have E04UFF determine constant elements in the objective and constraint gradients.

Feasibility Tolerance r Default = $\sqrt{\epsilon}$

The scalar r defines the maximum acceptable *absolute* violations in linear and nonlinear constraints at a 'feasible' point; i.e., a constraint is considered satisfied if its violation does not exceed r. If $r < \epsilon$ or $r \geq 1$, the default value is used. Using this keyword sets both optional parameters **Linear Feasibility Tolerance** and **Nonlinear Feasibility Tolerance** to r, if $\epsilon \leq r < 1$. (Additional details are given below under the descriptions of these parameters.)

Function Precision r Default = $\epsilon^{0.9}$

This parameter defines ϵ_R, which is intended to be a measure of the accuracy with which the problem functions $F(x)$ and $c(x)$ can be computed. If $r < \epsilon$ or $r \geq 1$, the default value is used.

The value of ϵ_R should reflect the relative precision of $1 + |F(x)|$; i.e., ϵ_R acts as a relative precision when $|F|$ is large, and as an absolute precision when $|F|$ is small. For example, if $F(x)$ is typically of order 1000 and the first six significant digits are known to be correct, an appropriate value for ϵ_R would be 10^{-6}. In contrast, if $F(x)$ is typically of order 10^{-4} and the first six significant digits are known to be correct, an appropriate value for ϵ_R would be 10^{-10}. The choice of ϵ_R can be quite complicated for badly scaled problems; see Chapter 8 of Gill *et al.* [10] for a discussion of scaling techniques. The default value is appropriate for most simple functions that are computed with full accuracy. However, when the accuracy of the computed function values is known to be significantly worse than full precision, the value of ϵ_R should be large enough so that E04UFF will not attempt to distinguish between function values that differ by less than the error inherent in the calculation.

Hessian **No** Default = **No**

Hessian **Yes**

This option controls the contents of the upper triangular matrix R (see Section 5). E04UFF works exclusively with the *transformed and re-ordered* Hessian H_Q (6), and hence extra computation is required to form the Hessian itself. If **Hessian** = **No**, R contains the Cholesky factor of the transformed and re-ordered Hessian. If **Hessian** = **Yes**, the Cholesky factor of the approximate Hessian itself is formed and stored in R. The user should select **Hessian** = **Yes** if a **Warm Start** will be used for the next call to E04UFF.

Infinite Bound Size r Default = 10^{20}

If $r > 0$, r defines the 'infinite' bound *bigbnd* in the definition of the problem constraints. Any upper bound greater than or equal to *bigbnd* will be regarded as plus infinity (and similarly any lower bound less than or equal to $-bigbnd$ will be regarded as minus infinity). If $r \leq 0$, the default value is used.

Infinite Step Size r Default = $\max(bigbnd, 10^{20})$

If $r > 0$, r specifies the magnitude of the change in variables that is treated as a step to an unbounded solution. If the change in x during an iteration would exceed the value of r, the objective function is considered to be unbounded below in the feasible region. If $r \leq 0$, the default value is used.

Iteration Limit i Default = $\max(50, 3(n + n_L) + 10n_N)$

See **Major Iteration Limit** below.

Line Search Tolerance r Default = 0.9

The value r ($0 \leq r < 1$) controls the accuracy with which the step α taken during each iteration approximates a minimum of the merit function along the search direction (the smaller the value of r,

the more accurate the line search). The default value $r = 0.9$ requests an inaccurate search, and is appropriate for most problems, particularly those with any nonlinear constraints.

If there are no nonlinear constraints, a more accurate search may be appropriate when it is desirable to reduce the number of major iterations – for example, if the objective function is cheap to evaluate, or if a substantial number of derivatives are unspecified. If $r < 0$ or $r \geq 1$, the default value is used.

Linear Feasibility Tolerance r_1 Default $= \sqrt{\epsilon}$

Nonlinear Feasibility Tolerance r_2 Default $= \epsilon^{0.33}$ or $\sqrt{\epsilon}$ (see below)

The default value of r_2 is $\epsilon^{0.33}$ if **Derivative Level** $= 0$ or 1, and $\sqrt{\epsilon}$ otherwise.

The scalars r_1 and r_2 define the maximum acceptable *absolute* violations in linear and nonlinear constraints at a 'feasible' point; i.e., a linear constraint is considered satisfied if its violation does not exceed r_1, and similarly for a nonlinear constraint and r_2. If $r_m < \epsilon$ or $r_m \geq 1$, the default value is used, for $m = 1, 2$.

On entry to E04UFF, an iterative procedure is executed in order to find a point that satisfies the linear constraints and bounds on the variables to within the tolerance r_1. All subsequent iterates will satisfy the linear constraints to within the same tolerance (unless r_1 is comparable to the finite difference interval).

For nonlinear constraints, the feasibility tolerance r_2 defines the largest constraint violation that is acceptable at an optimal point. Since nonlinear constraints are generally not satisfied until the final iterate, the value of **Nonlinear Feasibility Tolerance** acts as a partial termination criterion for the iterative sequence generated by E04UFF (see the discussion of **Optimality Tolerance**).

These tolerances should reflect the precision of the corresponding constraints. For example, if the variables and the coefficients in the linear constraints are of order unity, and the latter are correct to about 6 decimal digits, it would be appropriate to specify r_1 as 10^{-6}.

List Default = **List**

Nolist

Normally each optional parameter specification is printed as it is supplied. **Nolist** may be used to suppress the printing and **List** may be used to restore printing.

Major Iteration Limit i Default $= \max(50, 3(n + n_L) + 10n_N)$

Iteration Limit

Iters

Itns

The value of i specifies the maximum number of major iterations allowed before termination. Setting $i = 0$ and **Major Print Level** > 0 means that the workspace needed will be computed and printed, but no iterations will be performed. If $i < 0$, the default value is used.

Major Print Level i Default $= 10$

Print Level

The value of i controls the amount of printout produced by the major iterations of E04UFF, as indicated below. A detailed description of the printed output is given in Section 8.1 (summary output at each major iteration and the final solution) and Section 12 (monitoring information at each major iteration). (See also **Minor Print Level**, below.)

The following printout is sent to the current advisory message unit (as defined by X04ABF):

i	**Output**
0	No output.
1	The final solution only.
5	One line of summary output (< 80 characters; see Section 8.1) for each major iteration (no printout of the final solution).
≥ 10	The final solution and one line of summary output for each major iteration.

The following printout is sent to the logical unit number defined by the optional parameter **Monitoring File** (see below):

i	**Output**
< 5	No output.
≥ 5	One long line of output (> 80 characters; see Section 12) for each major iteration (no printout of the final solution).
≥ 20	At each major iteration, the objective function, the Euclidean norm of the nonlinear constraint violations, the values of the nonlinear constraints (the vector c), the values of the linear constraints (the vector $A_L x$), and the current values of the variables (the vector x).
≥ 30	At each major iteration, the diagonal elements of the matrix T associated with the TQ factorization (5) (see Section 10.1) of the QP working set, and the diagonal elements of R, the triangular factor of the transformed and re-ordered Hessian (6) (see Section 10.1).

If **Major Print Level** ≥ 5 and the unit number defined by **Monitoring File** is the same as that defined by X04ABF, then the summary output for each major iteration is suppressed.

Minor Iteration Limit i Default $= \max(50, 3(n + n_L + n_N))$

The value of i specifies the maximum number of iterations for finding a feasible point with respect to the bounds and linear constraints (if any). The value of i also specifies the maximum number of minor iterations for the optimality phase of each QP subproblem. If $i \leq 0$, the default value is used.

Minor Print Level i Default $= 0$

The value of i controls the amount of printout produced by the minor iterations of E04UFF (i.e., the iterations of the quadratic programming algorithm), as indicated below. A detailed description of the printed output is given in Section 8.2 of the document for E04NCF (summary output at each minor iteration and the final QP solution) and Section 12 of the document for E04NCF) (monitoring information at each minor iteration) of the document for E04NCF. (See also **Major Print Level**, above.)

The following printout is sent to the current advisory message unit (as defined by X04ABF):

i	**Output**
0	No output.
1	The final QP solution only.
5	One line of summary output (< 80 characters; see Section 8.2 of the document for E04NCF) for each minor iteration (no printout of the final QP solution).
≥ 10	The final QP solution and one line of summary output for each minor iteration.

The following printout is sent to the logical unit number defined by the optional parameter **Monitoring File** (see below):

i	**Output**
< 5	No output.
≥ 5	One long line of output (> 80 characters; see Section 12 of the document for E04NCF) for each minor iteration (no printout of the final QP solution).
≥ 20	At each minor iteration, the current estimates of the QP multipliers, the current estimate of the QP search direction, the QP constraint values, and the status of each QP constraint.
≥ 30	At each minor iteration, the diagonal elements of the matrix T associated with the TQ factorization (5) (see Section 10.1) of the QP working set, and the diagonal elements of the Cholesky factor R of the transformed Hessian (6) (see Section 10.1).

If **Minor Print Level** ≥ 5 and the unit number defined by **Monitoring File** is the same as that defined by X04ABF, then the summary output for each minor iteration is suppressed.

Monitoring File i Default $= -1$

If $i \geq 0$ and **Major Print Level**≥ 5 (see above) or $i \geq 0$ and **Minor Print Level** ≥ 5 (see above), monitoring information produced by E04UFF at every iteration is sent to a file with logical unit number i. If $i < 0$ and/or **Major Print Level** < 5 and **Minor Print Level** < 5, no monitoring information is produced.

Nonlinear Feasibility Tolerance r Default $= \epsilon^{0.33}$ or $\sqrt{\epsilon}$

See **Linear Feasibility Tolerance** above.

Optimality Tolerance r Default $= \epsilon_R^{0.8}$

The parameter r ($\epsilon_R \leq r < 1$) specifies the accuracy to which the user wishes the final iterate to approximate a solution of the problem. Broadly speaking, r indicates the number of correct figures desired in the objective function at the solution. For example, if r is 10^{-6} and E04UFF terminates successfully, the final value of F should have approximately six correct figures. If $r < \epsilon_R$ or $r \geq 1$, the default value is used.

E04UFF will terminate successfully if the iterative sequence of x-values is judged to have converged and the final point satisfies the first-order Kuhn–Tucker conditions (see Section 10.1). The sequence of iterates is considered to have converged at x if

$$\alpha\|p\| \leq \sqrt{r}(1 + \|x\|), \tag{16}$$

where p is the search direction and α the step length from (3). An iterate is considered to satisfy the first-order conditions for a minimum if

$$\|Z^T g_{\mathrm{FR}}\| \leq \sqrt{r}(1 + \max(1 + |F(x)|, \|g_{\mathrm{FR}}\|)) \tag{17}$$

and

$$|res_j| \leq ftol \text{ for all } j, \tag{18}$$

where $Z^T g_{\mathrm{FR}}$ is the projected gradient (see Section 10.1), g_{FR} is the gradient of $F(x)$ with respect to the free variables, res_j is the violation of the jth active nonlinear constraint, and $ftol$ is the **Nonlinear Feasibility Tolerance**.

Print Level

See **Major Print Level** above.

Start Objective Check At Variable i_1 Default $= 1$

Stop Objective Check At Variable i_2 Default $= n$

Start Constraint Check At Variable i_3 Default $= 1$

Stop Constraint Check At Variable i_4 Default $= n$

These keywords take effect only if **Verify Level** > 0 (see below). They may be used to control the verification of gradient elements and/or Jacobian elements computed by the calling program during intermediate exits. For example, if the first 30 elements of the objective gradient appeared to be correct in an earlier run, so that only element 31 remains questionable, it is reasonable to specify **Start Objective Check At Variable** 31. If the first 30 variables appear linearly in the objective, so that the corresponding gradient elements are constant, the above choice would also be appropriate.

If $i_{2m-1} \leq 0$ or $i_{2m-1} > \min(n, i_{2m})$, the default value is used, for $m = 1, 2$. If $i_{2m} \leq 0$ or $i_{2m} > n$, the default value is used, for $m = 1, 2$.

Step Limit r Default $= 2.0$

If $r > 0, r$ specifies the maximum change in variables at the first step of the line search. In some cases, such as $F(x) = ae^{bx}$ or $F(x) = ax^b$, even a moderate change in the elements of x can lead to floating-point overflow. The parameter r is therefore used to encourage evaluation of the problem functions at meaningful points. Given any major iterate x, the first point $\tilde{x}$ at which F and c are evaluated during the line search is restricted so that

$$\|\tilde{x} - x\|_2 \leq r(1 + \|x\|_2).$$

The line search may go on and evaluate F and c at points further from x if this will result in a lower value of the merit function (indicated by `L` at the end of each line of output produced by the major iterations; see Section 8.1). If `L` is printed for most of the iterations, r should be set to a larger value.

Wherever possible, upper and lower bounds on x should be used to prevent evaluation of nonlinear functions at wild values. The default value **Step Limit** $= 2.0$ should not affect progress on well-behaved functions, but values such as 0.1 or 0.01 may be helpful when rapidly varying functions are present. If a small value of **Step Limit** is selected, a good starting point may be required. An important application is to the class of nonlinear least-squares problems. If $r \leq 0$, the default value is used.

Stop Constraint Check At Variable

See **Start Constraint Check At Variable** above.

Stop Objective Check At Variable

See **Start Objective Check At Variable** above.

Verify Level	i	Default = 0
Verify No		
Verify Level	-1	
Verify Level	0	
Verify Objective Gradients		
Verify Level	1	
Verify Constraint Gradients		
Verify Level	2	
Verify		
Verify Yes		
Verify Gradients		
Verify Level	3	

These keywords refer to finite difference checks on the gradient elements computed by the calling program during intermediate exits. (Unspecified gradient elements are not checked.) It is possible to specify **Verify Level**s 0 – 3 in several ways, as indicated above. For example, the nonlinear objective gradient (if any) will be verified if either **Verify Objective Gradients** or **Verify Level** 1 is specified. Similarly, the objective and the constraint gradients will be verified if **Verify Yes** or **Verify Level** 3 or **Verify** is specified.

If $i = -1$, then no checking will be performed.

If $0 \leq i \leq 3$, gradients will be verified at the first point that satisfies the linear constraints and bounds. If $i = 0$, only a 'cheap' test will be performed, requiring one intermediate exit for the objective function gradients and (if appropriate) one intermediate exit for the partial derivatives of the constraints. If $1 \leq i \leq 3$, a more reliable (but more expensive) check will be made on individual gradient elements, within the ranges specified by the **Start** and **Stop** keywords described above. A result of the form `OK` or `BAD?` is printed by E04UFF to indicate whether or not each element appears to be correct.

If $10 \leq i \leq 13$, the action is the same as for $i - 10$, except that it will take place at the user-specified initial value of x.

If $i < -1$ or $4 \leq i \leq 9$ or $i > 13$, the default value is used.

We suggest that **Verify Level** $= 3$ be used whenever a new calling program is being developed.

Warm Start

See **Cold Start** above.

12 Description of Monitoring Information

This section describes the long line of output (> 80 characters) which forms part of the monitoring information produced by E04UFF. (See also the description of the optional parameters **Major Print Level**, **Minor Print Level** and **Monitoring File** in Section 11.2). The level of printed output can be controlled by the user.

When **Major Print Level** ≥ 5 and **Monitoring File** ≥ 0, the following line of output is produced at every major iteration of E04UFF on the unit number specified by **Monitoring File**. In all cases, the values of the quantities printed are those in effect *on completion* of the given iteration.

`Maj` is the major iteration count.

`Mnr` is the number of minor iterations required by the feasibility and optimality phases of the QP subproblem. Generally, `Mnr` will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution (see Section 10).

Note that `Mnr` may be greater than the **Minor Iteration Limit** if some iterations are required for the feasibility phase.

`Step` is the step taken along the computed search direction. On reasonably well-behaved problems, the unit step will be taken as the solution is approached.

`Nfun` is the cumulative number of evaluations of the objective function needed for the line search. Evaluations needed for the estimation of the gradients by finite differences are not included. `Nfun` is printed as a guide to the amount of work required for the line search.

`Merit Function` is the value of the augmented Lagrangian merit function (12) at the current iterate. This function will decrease at each iteration unless it was necessary to increase the penalty parameters (see Section 3). As the solution is approached, `Merit Function` will converge to the value of the objective function at the solution.

If the QP subproblem does not have a feasible point (signified by `I` at the end of the current output line), the merit function is a large multiple of the constraint violations, weighted by the penalty parameters. During a sequence of major iterations with infeasible subproblems, the sequence of `Merit Function` values will decrease monotonically until either a feasible subproblem is obtained or E04UFF terminates with IFAIL = 3 (no feasible point could be found for the nonlinear constraints).

If no nonlinear constraints are present (i.e., NCNLN = 0), this entry contains `Objective`, the value of the objective function $F(x)$. The objective function will decrease monotonically to its optimal value when there are no nonlinear constraints.

`Norm Gz` is $\|Z^T g_{\mathrm{FR}}\|$, the Euclidean norm of the projected gradient (see Section 10.2). `Norm Gz` will be approximately zero in the neighbourhood of a solution.

`Violtn` is the Euclidean norm of the residuals of constraints that are violated or in the predicted active set (not printed if NCNLN is zero). `Violtn` will be approximately zero in the neighbourhood of a solution.

`Nz` is the number of columns of Z (see Section 10.2). The value of `Nz` is the number of variables minus the number of constraints in the predicted active set; i.e., `Nz` $= n -$ (`Bnd` + `Lin` + `Nln`).

`Bnd` is the number of simple bound constraints in the predicted active set.

`Lin` is the number of general linear constraints in the predicted active set.

`Nln` is the number of nonlinear constraints in the predicted active set (not printed if NCNLN is zero).

`Penalty` is the Euclidean norm of the vector of penalty parameters used in the augmented Lagrangian merit function (not printed if NCNLN is zero).

`Cond H` is a lower bound on the condition number of the Hessian approximation H.

`Cond Hz` is a lower bound on the condition number of the projected Hessian approximation H_Z ($H_Z = Z^T H_{\mathrm{FR}} Z = R_Z^T R_Z$; see (6) in Section 10.1 and (11) in Section 10.2). The larger this number, the more difficult the problem.

`Cond T` is a lower bound on the condition number of the matrix of predicted active constraints.

`Conv` is a three-letter indication of the status of the three convergence tests (16)–(18) defined in the description of the optional parameter **Optimality Tolerance** in Section 11.2. Each letter is `T` if the test is satisfied, and `F` otherwise. The three tests indicate whether:

(a) the sequence of iterates has converged;
(b) the projected gradient (`Norm Gz`) is sufficiently small; and
(c) the norm of the residuals of constraints in the predicted active set (`Violtn`) is small enough.

If any of these indicators is `F` when E04UFF terminates with IFAIL = 0, the user should check the solution carefully.

`M` is printed if the quasi-Newton update has been modified to ensure that the Hessian approximation is positive-definite (see Section 10.4).

`I` is printed if the QP subproblem has no feasible point.

`C` is printed if central differences have been used to compute the unspecified objective and constraint gradients. If the value of `Step` is zero, the switch to central differences was made because no lower point could be found in the line search. (In this case, the QP subproblem is re-solved with the central difference gradient and Jacobian.) If the value of `Step` is non-zero, central differences were computed because `Norm Gz` and `Violtn` imply that x is close to a Kuhn–Tucker point (see Section 10.1).

`L` is printed if the line search has produced a relative change in x greater than the value defined by the optional parameter **Step Limit**. If this output occurs frequently during later iterations of the run, **Step Limit** should be set to a larger value.

`R` is printed if the approximate Hessian has been refactorized. If the diagonal condition estimator of R indicates that the approximate Hessian is badly conditioned, the approximate Hessian is refactorized using column interchanges. If necessary, R is modified so that its diagonal condition estimator is bounded.
