NAG Toolbox for MATLAB

e04us

1 Purpose

e04us is designed to minimize an arbitrary smooth sum of squares 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 you; any unspecified derivatives are approximated by finite differences. See the description of the optional parameter **Derivative Level**, in Section 11.2. It is not intended for large sparse problems.

e04us may also be used for unconstrained, bound-constrained and linearly constrained optimization.

2 Syntax

```
[iter, istate, c, cjac, f, fjac, clamda, objf, r, x, user, lwsav, iwsav,
rwsav, ifail] = e04us(a, bl, bu, y, confun, objfun, istate, cjac, fjac,
clamda, r, x, lwsav, iwsav, rwsav, 'm', m, 'n', n, 'nclin', nclin,
'ncnln', ncnln, 'user', user)
```

Before calling e04us, or the option setting function e04ur, e04wb **must** be called.

3 Description

e04us is designed to solve the nonlinear least-squares programming problem – the minimization of a smooth nonlinear sum of squares function subject to a set of constraints on the variables. The problem is assumed to be stated in the following form:

$$\underset{x \in R^n}{\text{Minimize}}\, F(x) = \tfrac{1}{2}\sum_{i=1}^{m}(y_i - f_i(x))^2 \qquad \text{subject to} \qquad l \leq \begin{Bmatrix} x \\ A_L x \\ c(x) \end{Bmatrix} \leq u, \tag{1}$$

where $F(x)$ (the *objective function*) is a nonlinear function which can be represented as the sum of squares of m subfunctions $(y_1 - f_1(x)), (y_2 - f_2(x)), \ldots, (y_m - f_m(x))$, the y_i are constant, 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 e04us 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**.)

You must supply an initial estimate of the solution to (1), together with (sub)programs that define $f(x) = (f_1(x), f_2(x), \ldots, f_m(x))^{\mathrm{T}}$, $c(x)$ and as many first partial derivatives as possible; unspecified derivatives are approximated by finite differences.

The subfunctions are defined by the array **y** and user-supplied (sub)program **objfun**, and the nonlinear constraints are defined by (sub)program **confun**. On every call, these (sub)programs must return appropriate values of $f(x)$ and $c(x)$. You should also provide the available partial derivatives. Any unspecified derivatives are approximated by finite differences for a discussion of the optional parameter **Derivative Level**. Just before either of the user-supplied (sub)programs **objfun** or **confun** is called, each element of the current gradient array **fjac** or **cjac** is initialized to a special value. On exit, any element that retains the value is estimated by finite differences. Note that if there *are* any nonlinear constraints, then the *first* call to **confun** will precede the *first* call to **objfun**.

For maximum reliability, it is preferable for you to provide all partial derivatives (see Chapter 8 of Gill *et al.* 1981 for a detailed discussion). If all gradients cannot be provided, it is similarly advisable to provide as many as possible. While developing the user-supplied (sub)programs **objfun** and **confun**, the optional parameter **Verify** should be used to check the calculation of any known gradients.

4 References

Gill P E, Murray W and Wright M H 1981 *Practical Optimization* Academic Press

Hock W and Schittkowski K 1981 *Test Examples for Nonlinear Programming Codes. Lecture Notes in Economics and Mathematical Systems* **187** Springer–Verlag

5 Parameters

5.1 Compulsory Input Parameters

1: **a(lda,$*$) – double array**

The first dimension of the array **a** must be at least $\max(1, \mathbf{nclin})$

The second dimension of the array must be at least **n** if $\mathbf{nclin} > 0$, and at least 1 otherwise

The ith row of **a** contains 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 $i = 1, 2, \ldots, \mathbf{nclin}$.

If $\mathbf{nclin} = 0$, the array **a** is not referenced.

2: **bl($\mathbf{n} + \mathbf{nclin} + \mathbf{ncnln}$) – double array**

3: **bu($\mathbf{n} + \mathbf{nclin} + \mathbf{ncnln}$) – double array**

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 nonexistent lower bound (i.e., $l_j = -\infty$), set $\mathbf{bl}(j) \leq -bigbnd$, and to specify a nonexistent upper bound (i.e., $u_j = +\infty$), set $\mathbf{bu}(j) \geq bigbnd$; the default value of $bigbnd$ is 10^{20}, but this may be changed by the optional parameter **Infinite Bound Size**. To specify the jth constraint as an *equality*, set $\mathbf{bl}(j) = \mathbf{bu}(j) = \beta$, say, where $|\beta| < bigbnd$.

Constraints:

$\mathbf{bl}(j) \leq \mathbf{bu}(j)$, for $j = 1, 2, \ldots, \mathbf{n} + \mathbf{nclin} + \mathbf{ncnln}$;
if $\mathbf{bl}(j) = \mathbf{bu}(j) = \beta$, $|\beta| < bigbnd$.

4: **y(m) – double array**

The coefficients of the constant vector y of the objective function.

5: **confun – string containing name of m-file**

confun must calculate the vector $c(x)$ of nonlinear constraint functions and (optionally) its Jacobian $\left(= \frac{\partial c}{\partial x}\right)$ for a specified n element vector x. If there are no nonlinear constraints (i.e., $\mathbf{ncnln} = 0$), **confun** will never be called by e04us and **confun** may be the string `'e04udm'`. **e04udm** is included in the NAG Fortran Library. If there are nonlinear constraints, the first call to **confun** will occur before the first call to user-supplied (sub)program **objfun**.

Its specification is:

```
[mode, c, cjac, user] = confun(mode, ncnln, n, ldcj, needc, x, cjac,
nstate, user)
```

Input Parameters

1: **mode – int32 scalar**

Indicates which values must be assigned during each call of **confun**. Only the following values need be assigned, for each value of i such that **needc**$(i) > 0$:

mode $= 0$

c(i).

mode $= 1$

All available elements in the ith row of **cjac**.

mode $= 2$

c(i) and all available elements in the ith row of **cjac**.

May be set to a negative value if you wish to terminate the solution to the current problem, and in this case e04us will terminate with **ifail** set to **mode**.

2: **ncnln – int32 scalar**

n_N, the number of nonlinear constraints.

3: **n – int32 scalar**

n, the number of variables.

4: **ldcj – int32 scalar**

5: **needc(ncnln) – int32 array**

The indices of the elements of **c** and/or **cjac** that must be evaluated by **confun**. If **needc**$(i) > 0$, then the ith element of **c** and/or the available elements of the ith row of **cjac** (see parameter **mode**) must be evaluated at x.

6: **x(n) – double array**

x, the vector of variables at which the constraint functions and/or all available elements of the constraint Jacobian are to be evaluated.

7: **cjac(ldcj,n) – double array**

ldcj, the first dimension of the array, must be at least $\max(1,\mathbf{ncnln})\max(1,\mathbf{ncnln})$.

Is set to a special value.

If **needc**$(i) > 0$ and **mode** $= 1$ or 2, the ith row of **cjac** must contain the available elements of the vector ∇c_i given by

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

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. See also the parameter **nstate**. The remaining rows of **cjac**, corresponding to nonpositive elements of **needc**, are ignored.

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 in **confun** will retain its initial value throughout. Constant elements may be loaded into **cjac** either before the call to e04us or during the first call to **confun** (signalled by the value **nstate** $= 1$). The ability to preload constants is useful when many Jacobian elements are identically zero, in which case **cjac** may be initialized to zero and nonzero elements may be reset by **confun**.

Note that constant nonzero elements do affect the values of the constraints. Thus, if **cjac**(i,j) is set to a constant value, it need not be reset in subsequent calls to **confun**, but the value **cjac**$(i,j) \times$ **x**(j) must nonetheless be added to **c**(i). For example, if **cjac**$(1,1) = 2$ and **cjac**$(1,2) = -5$, then the term $2 \times$ **x**$(1) - 5 \times$ **x**(2) must be included in the definition of **c**(1).

It must be emphasised that, if **Derivative Level** $= 0$ or 1, unassigned elements of **cjac** are not treated as constant; they are estimated by finite differences, at nontrivial expense. If you do not supply a value for the optional parameter **Difference Interval**, 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.

8: **nstate – int32 scalar**

If **nstate** $= 1$ then e04us is calling **confun** for the first time. This parameter setting allows you to save computation time if certain data must be read or calculated only once.

9: **user – Any MATLAB object**

confun is called from e04us with **user** as supplied to e04us

Output Parameters

1: **mode – int32 scalar**

Indicates which values must be assigned during each call of **confun**. Only the following values need be assigned, for each value of i such that **needc**$(i) > 0$:

mode $= 0$

c(i).

mode $= 1$

All available elements in the ith row of **cjac**.

mode $= 2$

c(i) and all available elements in the ith row of **cjac**.

May be set to a negative value if you wish to terminate the solution to the current problem, and in this case e04us will terminate with **ifail** set to **mode**.

2: **c(ncnln) – double array**

If **needc**$(i) > 0$ and **mode** $= 0$ or 2, **c**(i) must contain the value of the ith constraint at x. The remaining elements of **c**, corresponding to the nonpositive elements of **needc**, are ignored.

3: **cjac(ldcj,n) – double array**

Is set to a special value.

If **needc**$(i) > 0$ and **mode** $= 1$ or 2, the ith row of **cjac** must contain the available elements of the vector ∇c_i given by

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

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. See also the parameter **nstate**. The remaining rows of **cjac**, corresponding to nonpositive elements of **needc**, are ignored.

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 in **confun** will retain its initial value throughout. Constant elements may be loaded into **cjac** either before the call to e04us or during the first call to **confun** (signalled by the value **nstate** $= 1$). The ability to preload constants is useful when many Jacobian elements are identically zero, in which case **cjac** may be initialized to zero and nonzero elements may be reset by **confun**.

Note that constant nonzero elements do affect the values of the constraints. Thus, if $\mathbf{cjac}(i,j)$ is set to a constant value, it need not be reset in subsequent calls to **confun**, but the value $\mathbf{cjac}(i,j) \times \mathbf{x}(j)$ must nonetheless be added to $\mathbf{c}(i)$. For example, if $\mathbf{cjac}(1,1) = 2$ and $\mathbf{cjac}(1,2) = -5$, then the term $2 \times \mathbf{x}(1) - 5 \times \mathbf{x}(2)$ must be included in the definition of $\mathbf{c}(1)$.

It must be emphasised that, if **Derivative Level** $= 0$ or 1, unassigned elements of **cjac** are not treated as constant; they are estimated by finite differences, at nontrivial expense. If you do not supply a value for the optional parameter **Difference Interval**, 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.

4: **user – Any MATLAB object**

confun is called from e04us with **user** as supplied to e04us

Note: confun should be tested separately before being used in conjunction with e04us. See also the description of the optional parameter **Verify**.

6: **objfun – string containing name of m-file**

objfun must calculate either the ith element of the vector $f(x) = (f_1(x), f_2(x), \ldots, f_m(x))^{\mathrm{T}}$ or all m elements of $f(x)$ and (optionally) its Jacobian ($= \dfrac{\partial f}{\partial x}$) for a specified n element vector x.

Its specification is:

```
[mode, f, fjac, user] = objfun(mode, m, n, ldfj, needfi, x, fjac,
nstate, user)
```

Input Parameters

1: **mode – int32 scalar**

Indicates which values must be assigned during each call of **objfun**. Only the following values need be assigned:

mode $= 0$ and **needfi** $= i > 0$

$\mathbf{f}(i)$.

mode $= 0$ and **needfi** < 0

f.

mode $= 1$ and **needfi** < 0

All available elements of **fjac**.

mode $= 2$ and **needfi** < 0

f and all available elements of **fjac**.

May be set to a negative value if you wish to terminate the solution to the current problem, and in this case e04us will terminate with **ifail** set to **mode**.

2: **m – int32 scalar**

m, the number of subfunctions.

3: **n – int32 scalar**

n, the number of variables.

4: **ldfj – int32 scalar**

5: **needfi – int32 scalar**

If **needfi** $= i > 0$, only the ith element of $f(x)$ needs to be evaluated at x; the remaining elements need not be set. This can result in significant computational savings when $m \gg n$.

6: **x(n) – double array**

x, the vector of variables at which $f(x)$ and/or all available elements of its Jacobian are to be evaluated.

7: **fjac(ldfj,n) – double array**

ldfj, the first dimension of the array, must be at least **m**.

Is set to a special value.

If **mode** $= 1$ or 2 and **needfi** < 0, the ith row of **fjac** must contain the available elements of the vector ∇f_i given by

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

evaluated at the point x. See also the parameter **nstate**.

8: **nstate – int32 scalar**

If **nstate** $= 1$ then e04us is calling **objfun** for the first time. This parameter setting allows you to save computation time if certain data must be read or calculated only once.

9: **user – Any MATLAB object**

objfun is called from e04us with **user** as supplied to e04us

Output Parameters

1: **mode – int32 scalar**

Indicates which values must be assigned during each call of **objfun**. Only the following values need be assigned:

mode $= 0$ and **needfi** $= i > 0$

f(i).

mode $= 0$ and **needfi** < 0

f.

mode $= 1$ and **needfi** < 0

All available elements of **fjac**.

mode $= 2$ and **needfi** < 0

f and all available elements of **fjac**.

May be set to a negative value if you wish to terminate the solution to the current problem, and in this case e04us will terminate with **ifail** set to **mode**.

2: **f(m) – double array**

If **mode** $= 0$ and **needfi** $= i > 0$, $\mathbf{f}(i)$ must contain the value of f_i at x.

If **mode** $= 0$ or 2 and **needfi** < 0, $\mathbf{f}(i)$ must contain the value of f_i at x, for $i = 1, 2, \ldots, m$.

3: **fjac(ldfj,n) – double array**

Is set to a special value.

If **mode** $= 1$ or 2 and **needfi** < 0, the ith row of **fjac** must contain the available elements of the vector ∇f_i given by

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

evaluated at the point x. See also the parameter **nstate**.

4: **user – Any MATLAB object**

objfun is called from e04us with **user** as supplied to e04us

Note: objfun should be tested separately before being used in conjunction with e04us. See also the description of the optional parameter **Verify**.

7: **istate(n + nclin + ncnln) – int32 array**

Need not be set if the (default) optional parameter **Cold Start** is used.

If the optional parameter **Warm Start** has been chosen, 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 $\mathbf{bl}(j) = \mathbf{bu}(j)$.

The values -2, -1 and 4 are also acceptable but will be modified by the function. If e04us 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**.) The function also adjusts (if necessary) the values supplied in **x** to be consistent with **istate**.

Constraint: $-2 \leq \mathbf{istate}(j) \leq 4$, for $j = 1, 2, \ldots, \mathbf{n} + \mathbf{nclin} + \mathbf{ncnln}$.

8: **cjac(ldcj,*) – double array**

The first dimension of the array **cjac** must be at least $\max(1, \mathbf{ncnln})$

The second dimension of the array must be at least **n** if **ncnln** > 0, and at least 1 otherwise

In general, **cjac** need not be initialized before the call to e04us. However, if **Derivative Level** $= 3$, you may optionally set the constant elements of **cjac** (see parameter **nstate** in the description of (sub)program **confun**). Such constant elements need not be re-assigned on subsequent calls to **confun**.

9: **fjac**(**ldfj**,**n**) – **double array**

ldfj, the first dimension of the array, must be at least **m**.

In general, **fjac** need not be initialized before the call to e04us. However, if **Derivative Level** $= 3$, you may optionally set the constant elements of **fjac** (see parameter **nstate** in the description of user-supplied (sub)program **objfun**). Such constant elements need not be re-assigned on subsequent calls to **objfun**.

10: **clamda**($\mathbf{n} + \mathbf{nclin} + \mathbf{ncnln}$) – **double array**

Need not be set if the (default) optional parameter **Cold Start** is used.

If the optional parameter **Warm Start** has been chosen, **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, for $j = \mathbf{n} + \mathbf{nclin} + 1, \mathbf{n} + \mathbf{nclin} + 2, \ldots, \mathbf{n} + \mathbf{nclin} + \mathbf{ncnln}$. 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 nonnegative; if the jth constraint is an inequality active at its upper bound (i.e., **istate**(j) $= 2$, **clamda**(j) should be nonpositive. If necessary, the function will modify **clamda** to match these rules.

11: **r**(**ldr**,**n**) – **double array**

ldr, the first dimension of the array, must be at least **n**.

Need not be initialized if the (default) optional parameter **Cold Start** is used.

If the optional parameter **Warm Start** has been chosen, **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.

12: **x**(**n**) – **double array**

An initial estimate of the solution.

13: **lwsav**(**120**) – **logical array**

14: **iwsav**(**610**) – **int32 array**

15: **rwsav**(**475**) – **double array**

The arrays **lwsav**, **iwsav** and **rwsav** **must not** be altered between calls to any of the functions e04wb, e04us, e04ur.

5.2 Optional Input Parameters

1: **m** – **int32 scalar**

Default: The dimension of the arrays **fjac**, **r**, **x**. (An error is raised if these dimensions are not equal.)

m, the number of subfunctions associated with $F(x)$.

Constraint: $\mathbf{m} > 0$.

2: **n** – **int32 scalar**

n, the number of variables.

Constraint: $\mathbf{n} > 0$.

3: **nclin – int32 scalar**

n_L, the number of general linear constraints.

Constraint: **nclin** ≥ 0.

4: **ncnln – int32 scalar**

n_N, the number of nonlinear constraints.

Constraint: **ncnln** ≥ 0.

5: **user – Any MATLAB object**

user is not used by e04us, but is passed to **confun** and **objfun**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

5.3 Input Parameters Omitted from the MATLAB Interface

lda, ldcj, ldfj, ldr, iwork, liwork, work, lwork

5.4 Output Parameters

1: **iter – int32 scalar**

The number of major iterations performed.

2: **istate(n + nclin + ncnln) – int32 array**

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**). 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**). 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).

3: **c(∗) – double array**

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

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** $= 0$, the array **c** is not referenced.

4: **cjac(ldcj,∗) – double array**

The first dimension of the array **cjac** must be at least $\max(1,$ **ncnln**$)$

The second dimension of the array must be at least **n** if **ncnln** > 0, and at least 1 otherwise

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** and $j = 1, 2, \ldots,$ **n**. (See the discussion of parameter **cjac** under (sub)program **confun**.)

If **ncnln** $= 0$, the array **cjac** is not referenced.

5: **f**(**m**) – **double array**

f(i) contains the value of the ith function f_i at the final iterate, for $i = 1, 2, \ldots,$ **m**.

6: **fjac**(**ldfj,n**) – **double array**

The Jacobian matrix of the functions $f_1, f_2, \ldots, f_m$ at the final iterate, i.e., **fjac**(i,j) contains the partial derivative of the ith function with respect to the jth variable, for $i = 1, 2 \ldots,$ **m** and $j = 1, 2, \ldots,$ **n**. (See also the discussion of parameter **fjac** under user-supplied (sub)program **objfun**.)

7: **clamda**(**n** + **nclin** + **ncnln**) – **double array**

The values of the QP multipliers from the last QP subproblem. **clamda**(j) should be nonnegative if **istate**$(j) = 1$ and nonpositive if **istate**$(j) = 2$.

8: **objf** – **double scalar**

The value of the objective function at the final iterate.

9: **r**(**ldr,n**) – **double array**

If **Hessian** = No, **r** contains the upper triangular Cholesky factor R of $Q^{\mathrm{T}}\tilde{H}Q$, an estimate of the transformed and reordered Hessian of the Lagrangian at x (see (6) of the document for e04uf). 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.

10: **x**(**n**) – **double array**

The final estimate of the solution.

11: **user** – **Any MATLAB object**

user is not used by e04us, but is passed to **confun** and **objfun**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

12: **lwsav**(**120**) – **logical array**

13: **iwsav**(**610**) – **int32 array**

14: **rwsav**(**475**) – **double array**

The arrays **lwsav**, **iwsav** and **rwsav** **must not** be altered between calls to any of the functions e04wb, e04us, e04ur.

15: **ifail** – **int32 scalar**

0 unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

Note: e04us may return useful information for one or more of the following detected errors or warnings.

ifail < 0

A negative value of **ifail** indicates an exit from e04us because you set **mode** < 0 in the user-supplied (sub)programs **objfun** or **confun**. The value of **ifail** will be the same as your setting of **mode**.

ifail $= 1$

The final iterate x satisfies the first-order Kuhn–Tucker conditions (see Section 10.1 of the document for e04uf) to the accuracy requested, but the sequence of iterates has not yet converged. e04us 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 you ask 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***)). 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 $= 2$

e04us 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***), or no feasible point could be found in the number of iterations specified by the optional parameter **Minor Iteration Limit** (default value $= \max(50, 3(n + n_L + n_N))$). You should check that there are no constraint redundancies. If the data for the constraints are accurate only to an absolute precision σ, you 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. You should check the validity of constraints with negative values of **istate**. If you are convinced that a feasible point does exist, e04us 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 $= \max(50, 3(n + n_L) + 10n_N)$) 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 e04us or, alternatively, rerun e04us using the optional parameter **Warm Start**. If the algorithm seems to be making little or no progress however, then you should check for incorrect gradients or ill-conditioning as described 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 function.

ifail $= 6$

x does not satisfy the first-order Kuhn–Tucker conditions (see Section 10.1 of the document for e04uf), and no improved point for the merit function (see Section 8.1) could be found during the final linesearch.

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***)) is too small. In this case you should apply the four tests described under **ifail** $= 0$ to determine whether or not the final solution is acceptable (see Gill *et al.* 1981, for a discussion of the attainable accuracy).

If many iterations have occurred in which essentially no progress has been made and e04us has failed completely to move from the initial point then user-supplied (sub)programs **objfun** and/or **confun** may be incorrect. You should refer to comments under **ifail** $= 7$ and check the gradients using the optional parameter **Verify** (default value $= 0$). 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 12) is extremely large, it may be worthwhile rerunning e04us from the final point with the optional parameter **Warm Start**. 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 e04us with a relaxed value of the optional parameter **Feasibility Tolerance** (default value $= \sqrt{\epsilon}$, where ϵ is the ***machine precision***). (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).

ifail $= 7$

The user-supplied derivatives of the subfunctions and/or nonlinear constraints appear to be incorrect.

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

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

Special care should be used in this test if computation of the subfunctions involves subsidiary data communicated in global storage. Although the first evaluation of the subfunctions may be correct, subsequent calculations may be in error because some of the subsidiary data has accidentally been overwritten.

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

Errors in programming the subfunctions may be quite subtle in that the subfunction values are 'almost' correct. For example, a subfunction 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 subfunction 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 subfunction; 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 function.

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** and/or the optional parameter **Nonlinear Feasibility Tolerance** and rerunning the program. If the message recurs even after this change then the offending linearly dependent constraint (with index 'j') must be removed from the problem. If overflow occurs in one of the user-supplied (sub)programs (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 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***).

8 Further Comments

8.1 Description of the Printed Output

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 of the document for e04uf).

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

`Step` is the step α_k taken along the computed search direction. On reasonably well-behaved problems, the unit step (i.e., $\alpha_k = 1$) will be taken as the solution is approached.

`Merit Function` is the value of the augmented Lagrangian merit function (see (12) of the document for e04uf) at the current iterate. This function will decrease at each iteration unless it was necessary to increase the penalty parameters (see Section 10.3 of the document for e04uf). 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) then 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 e04us terminates with **ifail** $= 3$ (no feasible point could be found for the nonlinear constraints).

If there are no nonlinear constraints present (i.e., **ncnln** $= 0$) then 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 $\|Z^{\mathrm{T}} g_{\mathrm{FR}}\|$, the Euclidean norm of the projected gradient (see Section 10.2 of the document for e04uf). `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 ($H_Z = Z^{\mathrm{T}} H_{\mathrm{FR}} Z = R_Z^{\mathrm{T}} R_Z$; see (6) and (11) of the document for e04uf). 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 of the document for e04uf).

`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 then the switch to central differences was made because no lower point could be found in the linesearch. (In this case, the QP subproblem is resolved with the central difference gradient and Jacobian.) If the value of `Step` is nonzero then central differences were computed because `Norm Gz` and `Violtn` imply that x is close to a Kuhn–Tucker point (see Section 10.1 of the document for e04uf).

`L` is printed if the linesearch 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, optional parameter **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 then 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 working set, `EQ` if a fixed variable, `LL` if on its lower bound, `UL` if on its upper bound, `TF` if temporarily fixed at its current value). If `Value` lies outside the upper or lower bounds by more than the **Feasibility Tolerance**, `State` will be `++` or `--` respectively.

A key is sometimes printed before `State`.

`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 then 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 iteration.

Lower Bound is the lower bound specified for the variable. None indicates that **bl**$(j) \leq -bigbnd$.

Upper Bound is the upper bound specified for the variable. None indicates that **bu**$(j) \geq bigbnd$.

Lagr Mult is the Lagrange multiplier for the associated bound. This will be zero if State is FR unless **bl**$(j) \leq -bigbnd$ and **bu**$(j) \geq bigbnd$, in which case the entry will be blank. If x is optimal, the multiplier should be nonnegative if State is LL and nonpositive if State is UL.

Slack is the difference between the variable Value and the nearer of its (finite) bounds **bl**(j) and **bu**(j). A blank entry indicates that the associated variable is not bounded (i.e., **bl**$(j) \leq -bigbnd$ and **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', **bl**(j) and **bu**(j) are replaced by **bl**$(n+j)$ and **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

e04us_confun.m

```
function [mode, c, cjac, user] = ...
    confun(mode, ncnln, n, ldcj, needc, x, cjac, nstate, user)
  c = zeros(ncnln, 1);

  if (nstate == 1)
%   first call to confun. set all jacobian elements to zero.
%   note that this will only work when 'derivative level = 3'
%   (the default; see section 11.2).
    cjac = zeros(ncnln, n);
  end
  if (needc(1) > 0)
    if (mode == 0 || mode == 2)
      c(1) = -0.09 - x(1)*x(2) + 0.49*x(2);
    end
    if (mode == 1 || mode == 2)
      cjac(1,1) = -x(2);
      cjac(1,2) = -x(1) + 0.49;
    end
  end
```

```
e04us_objfun.m
```

```
function [mode, f, fjac, user] = objfun(mode, m, n, ldfj, needfi, x,
fjac, nstate, user)
  f = zeros(m, 1);

   a = [8, 8, 10, 10, 10, 10, 12, 12, 12, 12, 14, 14, 14, 16, 16, 16,
18, 18, ...
       20, 20, 20, 22, 22, 22, 24, 24, 24, 26, 26, 26, 28, 28, 30, 30,
30, ...
       32, 32, 34, 36, 36, 38, 38, 40, 42];

  for i = 1:m
    temp = exp(-x(2)*(a(i)-8));
    if (mode  ==  0 || mode  ==  2)
      f(i) = x(1) + (0.49-x(1))*temp;
    end
    if (mode  ==  1 || mode  ==  2)
      fjac(i,1) = 1 - temp;
      fjac(i,2) = -(0.49-x(1))*(a(i)-8)*temp;
    end
  end
```

```
a = [1, 1];
bl = [0.4;
     -4;
     1;
     0];
bu = [9.999999999999999e+24;
     9.999999999999999e+24;
     9.999999999999999e+24;
     9.999999999999999e+24];
y = [0.49;
     0.49;
     0.48;
     0.47;
     0.48;
     0.47;
     0.46;
     0.46;
     0.45;
     0.43;
     0.45;
     0.43;
     0.43;
     0.44;
     0.43;
     0.43;
     0.46;
     0.45;
     0.42;
     0.42;
     0.43;
     0.41;
     0.41;
     0.4;
     0.42;
     0.4;
     0.4;
     0.41;
     0.4;
     0.41;
     0.41;
     0.4;
     0.4;
     0.4;
     0.38;
```

```
      0.41;
      0.4;
      0.4;
      0.41;
      0.38;
      0.4;
      0.4;
      0.39;
      0.39];
istate = zeros(4, 1, 'int32');
cjac = [0, 0];
fjac = [0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0;
      0, 0];
clamda = zeros(4, 1);
r = [0, 0;
      0, 0];
x = [0.4;
      0];
[cwsav,lwsav,iwsav,rwsav,ifail] = e04wb('e04us');
[iter, istateOut, c, cjacOut, f, fjacOut, clamdaOut, objf, rOut, xOut,
user, ...
 lwsavOut, iwsavOut, rwsavOut, ifail] = ...
       e04us(a, bl, bu, y, 'e04us_confun', 'e04us_objfun', istate, cjac,
fjac, clamda, r, x, lwsav, iwsav, rwsav)
```

```
iter =
           6
istateOut =
```

```
           0
           0
           0
           1
c =
  -9.7677e-13
cjacOut =
   -1.2848    0.0700
f =
    0.4900
    0.4900
    0.4253
    0.4253
    0.4253
    0.4253
    0.4204
    0.4204
    0.4204
    0.4204
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
    0.4200
fjacOut =
         0         0
         0         0
    0.9234   -0.0107
    0.9234   -0.0107
    0.9234   -0.0107
    0.9234   -0.0107
    0.9941   -0.0016
    0.9941   -0.0016
    0.9941   -0.0016
    0.9941   -0.0016
    0.9996   -0.0002
    0.9996   -0.0002
    0.9996   -0.0002
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
```

```
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
    1.0000   -0.0000
clamdaOut =
         0
         0
         0
    0.0334
objf =
    0.0142
rOut =
   -0.3365   -6.4556
   -0.7623   -1.3572
xOut =
    0.4200
    1.2848
user =
     0
lwsavOut =
     array elided
iwsavOut =
     array elided
rwsavOut =
     array elided
ifail =
           0
```

Note: *the remainder of this document is intended for more advanced users. Section 11 describes the optional parameters which may be set by calls to e04ur. Section 12 describes the quantities which can be requested to monitor the course of the computation.*

10 Algorithmic Details

e04us implements a sequential quadratic programming (SQP) method incorporating an augmented Lagrangian merit function and a BFGS (Broyden–Fletcher–Goldfarb–Shanno) quasi-Newton approximation to the Hessian of the Lagrangian, and is based on e04wd. The documents for e04nc, e04wd and e04uf should be consulted for details of the method.

11 Optional Parameters

Several optional parameters in e04us define choices in the problem specification or the algorithm logic. In order to reduce the number of formal parameters of e04us these optional parameters have associated *default values* that are appropriate for most problems. Therefore you need only specify those optional parameters whose values are to be different from their default values.

The remainder of this section can be skipped if you 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 e04ur before a call to e04us.

e04ur can be called to supply options directly, one call being necessary for each optional parameter. For example,

```
[lwsav, iwsav, rwsav, inform] = e04ur('Print Level = 1', lwsav, iwsav,
rwsav);
```

e04ur should be consulted for a full description of this method of supplying optional parameters.

All optional parameters not specified by you are set to their default values. Optional parameters specified by you are unaltered by e04us (unless they define invalid values) and so remain in effect for subsequent calls to e04us, unless altered by you.

11.1 Optional Parameter Checklist and Default Values

The following list gives the valid options. For each option, we give the keyword, any essential optional qualifiers and the default value. A definition for each option can be found in Section 11.2. 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 double values required with certain options. The number ϵ is a generic notation for ***machine precision*** (see x02aj), and ϵ_R denotes the relative precision of the objective function (the **Function Precision**). Further details of other quantities not explicitly defined in this section may be found by consulting the document for e04uf.

Optional Parameters	**Default Values**
Central Difference Interval	Default values are computed
Cold Start	Default
Crash Tolerance	Default $= 0.01$
Defaults	
Derivative Level	Default $= 3$
Difference Interval	Default values are computed
Feasibility Tolerance	Default $= \sqrt{\epsilon}$
Function Precision	Default $= \epsilon^{0.9}$
Hessian	Default = No
Infinite Bound Size	Default $= 10^{20}$
Infinite Step Size	Default $= \max\left(bigbnd, 10^{20}\right)$
Iteration Limit	See **Major Iteration Limit**
Iters	See **Major Iteration Limit**
Itns	See **Major Iteration Limit**
JTJ Initial Hessian	Default
Line Search Tolerance	Default $= 0.9$
Linear Feasibility Tolerance	Default $= \sqrt{\epsilon}$
List	Default for $e04us$ = **List**
Major Iteration Limit	Default $= \max(50, 3(n + n_L) + 10n_N)$
Major Print Level	Default for e04us $= 10$
Minor Iteration Limit	Default $= \max(50, 3(n + n_L + n_N))$
Minor Print Level	Default $= 0$
Monitoring File	Default $= -1$
Nolist	Default for $e04us$ = **Nolist**. See **List**.
Nonlinear Feasibility Tolerance	Default $= \epsilon^{0.33}$ or $\sqrt{\epsilon}$. See **Linear Feasibility Tolerance**.

Optimality Tolerance	Default $= \epsilon_R^{0.8}$
Print Level	Default for e04us $= 0$. See **Major Print Level**.
Reset Frequency	Default $= 2$
Start Constraint Check At Variable	Default $= 1$. See **Start Objective Check At Variable**.
Start Objective Check At Variable	Default $= 1$
Step Limit	Default $= 2.0$
Stop Constraint Check At Variable	Default $= n$. See **Start Objective Check At Variable**.
Stop Objective Check At Variable	Default $= n$. See **Start Objective Check At Variable**.
Unit Initial Hessian	See **JTJ Initial Hessian**
Verify	See **Verify Level**
Verify Constraint Gradients	See **Verify Level**
Verify Gradients	See **Verify Level**
Verify Level	Default $= 0$
Verify Objective Gradients	See **Verify Level**
Warm Start	See **Cold Start**

11.2 Description of the Optional Parameters

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 under the optional parameter **Difference Interval**.

Cold Start Default
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 e04us 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**).

With a **Warm Start**, you 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. e04us will override your 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 e04us 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 e04us 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 $a_j^{\mathrm{T}} x \geq l$ will be included in the initial working set if $\left|a_j^{\mathrm{T}} x - l\right| \leq r(1 + |l|)$. 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 in user-supplied (sub)programs **objfun** and **confun**. The possible choices for i are the following.

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

The value $i = 3$ should be used whenever possible, since e04us is more reliable (and will usually be more efficient) when all derivatives are exact.

If $i = 0$ or 2, e04us will approximate unspecified elements of the objective Jacobian, using finite differences. The computation of finite difference approximations usually increases the total run-time, since a call to user-supplied (sub)program **objfun** is required for each unspecified element. Furthermore, less accuracy can be attained in the solution (see Chapter 8 of Gill *et al.* 1981, for a discussion of limiting accuracy).

If $i = 0$ or 1, e04us will approximate unspecified elements of the constraint Jacobian. One call to (sub)program **confun** is needed for each variable for which partial derivatives are not available. For example, if the constraint Jacobian has the form

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

where '$*$' indicates an element provided by you and '?' indicates an unspecified element, e04us will call **confun** 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 calls to **confun**.)

At times, central differences are used rather than forward differences, in which case twice as many calls to user-supplied (sub)program **objfun** and (sub)program **confun** are needed. (The switch to central differences is not under your 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 the optional parameter **Verify**).

(b) For estimating unspecified elements of the objective and/or constraint Jacobian matrix.

In general, a derivative with respect to the jth variable is approximated using the interval δ_j, where $\delta_j = r\left(1 + \left|\hat{x}_j\right|\right)$, 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.* 1981 for a discussion of the accuracy in finite difference approximations.

If a difference interval is not specified by you, a finite difference interval will be computed automatically for each variable by a procedure that requires up to six calls of (sub)program **confun** and user-supplied (sub)program **objfun** for each element. This option is recommended if the function is badly scaled or you wish to have e04us determine constant elements in the objective and constraint gradients (see the descriptions of **confun** and **objfun** in Section 5).

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 under the descriptions of these optional 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.* 1981 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 e04us will not attempt to distinguish between function values that differ by less than the error inherent in the calculation.

Hessian Default $=$ No

This option controls the contents of the upper triangular matrix R (see Section 5). e04us works exclusively with the *transformed and reordered* Hessian H_Q, and hence extra computation is required to form the Hessian itself. If **Hessian** $=$ No, **r** contains the Cholesky factor of the transformed and reordered Hessian. If **Hessian** $=$ Yes, the Cholesky factor of the approximate Hessian itself is formed and stored in **r**. You should select **Hessian** $=$ Yes if a **Warm Start** will be used for the next call to e04us.

Infinite Bound Size r Default $= 10^{20}$

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

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

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.

JTJ Initial Hessian Default
Unit Initial Hessian

This option controls the initial value of the upper triangular matrix R. If J denotes the objective Jacobian matrix $\nabla f(x)$, then $J^{\mathrm{T}}J$ is often a good approximation to the objective Hessian matrix $\nabla^2 F(x)$ (see also optional parameter **Reset Frequency**).

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 linesearch). 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 $\sqrt{\epsilon}$

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 e04us, 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 optional parameter **Nonlinear Feasibility Tolerance** acts as a partial termination criterion for the iterative sequence generated by e04us (see also optional parameter **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 for $e04us =$ **List**
Nolist Default for $e04us =$ **Nolist**

Normally each optional parameter specification is printed as it is supplied. Optional parameter **Nolist** may be used to suppress the printing and optional parameter **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 for e04us $= 10$
Print Level Default for e04us $= 0$

The value of i controls the amount of printout produced by the major iterations of e04us, 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 the description of the optional parameter **Minor Print Level**.)

The following printout is sent to the current advisory message unit (as defined by x04ab):

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**:

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 (see (5) of the document for e04uf) of the QP working set, and the diagonal elements of R, the triangular factor of the transformed and reordered Hessian (see (6) of the document for e04uf).

If **Major Print Level** ≥ 5 and the unit number defined by the optional parameter **Monitoring File** is the same as that defined by x04ab, then the summary output for each major iteration is suppressed.

Minor Iteration Limit $\quad i \quad$ 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 $\quad i \quad$ Default $= 0$

The value of i controls the amount of printout produced by the minor iterations of e04us (i.e., the iterations of the quadratic programming algorithm), as indicated below. A detailed description of the printed output is given in Section 8.1 (summary output at each minor iteration and the final QP solution) and Section 12 (monitoring information at each minor iteration). (See also the description of the optional parameter **Major Print Level**.)

The following printout is sent to the current advisory message unit (as defined by x04ab):

i	**Output**
0	No output.
1	The final QP solution only.
5	One line of summary output ($<$ 80 characters; see Section 8.1) 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**:

i	**Output**
< 5	No output.
≥ 5	One long line of output ($>$ 80 characters; see Section 12) 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 (see (5) of the document for e04uf) of the QP working set, and the diagonal elements of the Cholesky factor R of the transformed Hessian (see (6) of the document for e04uf).

If **Minor Print Level** ≥ 5 and the unit number defined by the optional parameter **Monitoring File** is the same as that defined by x04ab, then the summary output for each minor iteration is suppressed.

Monitoring File i Default $= -1$

If $i \geq 0$ and **Major Print Level** ≥ 5 or $i \geq 0$ and **Minor Print Level** ≥ 5, monitoring information produced by e04us 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.

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

The parameter r ($\epsilon_R \leq r < 1$) specifies the accuracy to which you wish 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 e04us 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.

e04us 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 of the document for e04uf). The sequence of iterates is considered to have converged at x if

$$\alpha\|p\| \leq \sqrt{r}(1 + \|x\|), \tag{2}$$

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

$$\left\|Z^{\mathrm{T}} g_{\mathrm{FR}}\right\| \leq \sqrt{r}(1 + \max(1 + |F(x)|, \|g_{\mathrm{FR}}\|)) \tag{3}$$

and

$$\left|res_j\right| \leq ftol \qquad \text{for all} \qquad j, \tag{4}$$

where $Z^{\mathrm{T}} g_{\mathrm{FR}}$ is the projected gradient, 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**.

Reset Frequency i Default $= 2$

If $i > 0$, this parameter allows you to reset the approximate Hessian matrix to $J^{\mathrm{T}}J$ every i iterations, where J is the objective Jacobian matrix $\nabla f(x)$ (see also the description of the optional parameter **JTJ Initial Hessian**).

At any point where there are no nonlinear constraints active and the values of f are small in magnitude compared to the norm of J, $J^{\mathrm{T}}J$ will be a good approximation to the objective Hessian $\nabla^2 F(x)$. Under these circumstances, frequent resetting can significantly improve the convergence rate of e04us.

Resetting is suppressed at any iteration during which there are nonlinear constraints active.

If $i \leq 0$, the default value is used.

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 Default $= n$

These keywords take effect only if **Verify Level** > 0. They may be used to control the verification of Jacobian elements computed by user-supplied (sub)programs **objfun** and **confun**. For example, if the first 30 columns of the objective Jacobian appeared to be correct in an earlier run, so that only column 31 remains questionable, it is reasonable to specify **Start Objective Check At Variable** $= 31$. If the first 30 variables appear linearly in the subfunctions, so that the corresponding Jacobian 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 $m = 1$ or 2. If $i_{2m} \leq 0$ or $i_{2m} > n$, the default value is used, for $m = 1$ or 2.

Step Limit r Default $= 2.0$

If $r > 0, r$ specifies the maximum change in variables at the first step of the linesearch. 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 linesearch is restricted so that

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

The linesearch 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.

Verify Level i Default $= 0$
Verify
Verify Constraint Gradients
Verify Gradients
Verify Objective Gradients

These keywords refer to finite difference checks on the gradient elements computed by the user-supplied (sub)programs **objfun** and **confun**. (Unspecified gradient elements are not checked.) The possible choices for i are the following:

i	Meaning
-1	No checks are performed.
0	Only a 'cheap' test will be performed, requiring one call to user-supplied (sub)program **objfun**.
1	Individual gradient elements will also be checked using a reliable (but more expensive) test.

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 $i = -1$, 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 call to user-supplied (sub)program **objfun** and (if appropriate) one call to (sub)program **confun**. 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. A result of the form `OK` or `BAD?` is printed by e04us 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 function function is being developed.

12 Description of Monitoring Information

This section describes the long line of output (> 80 characters) which forms part of the monitoring information produced by e04us. (See also the description of the optional parameters **Major Print Level**, **Minor Print Level** and **Monitoring File**.) You can control the level of printed output.

When **Major Print Level** ≥ 5 and **Monitoring File** ≥ 0, the following line of output is produced at every major iteration of e04us on the unit number specified by optional parameter **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 of the document for e04uf).

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

`Step` is the step α_k taken along the computed search direction. On reasonably well-behaved problems, the unit step (i.e., $\alpha_k = 1$) will be taken as the solution is approached.

`Nfun` is the cumulative number of evaluations of the objective function needed for the linesearch. 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 linesearch.

`Merit Function` is the value of the augmented Lagrangian merit function (see (12) of the document for e04uf) at the current iterate. This function will decrease at each iteration unless it was necessary to increase the penalty parameters (see Section 10.3 of the document for e04uf). 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) then 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 e04us terminates with **ifail** $= 3$ (no feasible point could be found for the nonlinear constraints).

If there are no nonlinear constraints present (i.e., **ncnln** $= 0$) then 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 $\|Z^{\mathrm{T}} g_{\mathrm{FR}}\|$, the Euclidean norm of the projected gradient (see Section 10.2 of the document for e04uf). `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 of the document for e04uf). The value of `Nz` is the number of variables minus the number of constraints in the predicted active set; i.e., $\mathtt{Nz} = n - (\mathtt{Bnd} + \mathtt{Lin} + \mathtt{Nln})$.

`Bnd` is the number of simple bound constraints in the current working set.

`Lin` is the number of general linear constraints in the current working 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 ($H_Z = Z^{\mathrm{T}} H_{\mathrm{FR}} Z = R_Z^{\mathrm{T}} R_Z$; see (6) and (11) of the document for e04uf). 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 (2)–(4) defined in the description of the optional parameter **Optimality Tolerance**. Each letter is `T` if the test is satisfied and `F` otherwise. The three tests indicate whether:

(i) the sequence of iterates has converged;

(ii) the projected gradient (`Norm Gz`) is sufficiently small; and

(iii) the norm of the residuals of constraints in the predicted active set (`Violtn`) is small enough.

If any of these indicators is `F` when e04us terminates with **ifail** $= 0$, you 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 of the document for e04uf).

`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 then the switch to central differences was made because no lower point could be found in the linesearch. (In this case, the QP subproblem is resolved with the central difference gradient and Jacobian.) If the value of `Step` is nonzero then central differences were computed because `Norm Gz` and `Violtn` imply that x is close to a Kuhn–Tucker point (see Section 10.1 of the document for e04uf).

`L` is printed if the linesearch 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, optional parameter **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 then the approximate Hessian is refactorized using column interchanges. If necessary, R is modified so that its diagonal condition estimator is bounded.