1 SUMMARY

To **minimize an objective function consisting of a sum of squares of 'element' functions, each of which involves only a few variables or whose second derivative matrix has a low rank for other reasons.** Bounds on the variables and known values may be specified. The subroutine is especially effective on problems involving a large number of variables.

The objective function has the form

$$F(\mathbf{x}) = \frac{1}{2}\sum_{k=1}^{n_s} f_k^2(\mathbf{x}_k) \qquad \mathbf{x} = (x_1, x_2, ..., x_n)$$

where the $\mathbf{x}_k$ are small subsets of $\mathbf{x}$. The user is required to write a subroutine that calculates each function f_k and (optionally) its gradient

$$g_k = \left(\frac{\partial f_k}{\partial x_i}\right).$$

When code for the gradient is supplied (and this usually results in faster and more reliable execution), there are facilities for automatically checking it during the first iteration.

There are flexible re-start facilities for problems that do not differ too substantially from a problem previously solved. Using these can sometimes lead to dramatically reduced execution times.

If there is a linear transformation of variables such that one or more of the element functions depend on fewer transformed variables than the number of original variables x_i they involve, this may be specified. This too can lead to substantially improved computing time.

There are facilities for the user: to specify the termination criterion, and to choose whether to calculate the approximations to the Hessian matrices

$$H_k = \left(\frac{\partial^2 f_k}{\partial x_i \partial x_j}\right).$$

by finite differences at the starting point.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** `VE08`. **Original date:** March 1987. **Origin:** Ph.Toint, FUNDP, Namur, Belgium. **Licence:** A third-party licence for this package is available without charge.

2 HOW TO USE THE PACKAGE

2.1 The Argument List

The single precision version

```
CALL VE10A (N,NS,INVAR,NVAR,ELFNCT,RANGE,XLOWER,XUPPER,
            X,FX,EPSIL,NGR,NIT,FKNOWN,FLOWBD,RELPR,
            DIFGRD,RESTRT,TESTGX,HESDIF,STMAX,STEPL,ISTATE,
            IPDEVC,IPFREQ,IPWHAT,LWK,WK,INFO,IFLAG)
```

The double precision version

```
CALL VE10AD(N,NS,INVAR,NVAR,ELFNCT,RANGE,XLOWER,XUPPER,
            X,FX,EPSIL,NGR,NIT,FKNOWN,FLOWBD,RELPR,
            DIFGRD,RESTRT,TESTGX,HESDIF,STMAX,STEPL,ISTATE,
```

IPDEVC,IPFREQ,IPWHAT,LWK,WK,INFO,IFLAG)

N is an INTEGER variable, that must be set by the user to *n* the number of variables. It is not altered by the subroutine. **Restriction:** N > 0.

NS is an INTEGER variable, that must be set by the user to n_s the number of element functions. It is not altered by the subroutine. **Restriction:** NS > 0.

INVAR is an INTEGER array containing the indices of the variables in the first element, followed by those in the second element, etc. See §5 for an example. It is not altered by the subroutine.

NVAR is an INTEGER array of length at least NS+1, whose *k*th value is the position of the first variable of element function *k*, in the list INVAR. In addition, NVAR(NS+1) should be equal to the total length of INVAR plus one. See §5 for an example. It is not altered by the subroutine.

ELFNCT is the name of a user supplied subroutine that computes the values of the element functions at a given point. See the complete description in §2.2. This name should be declared EXTERNAL in the calling program.

RANGE is the name of a user supplied subroutine that performs various operations related to the true range of the element function second derivative matrices. See the complete description in §2.3. This name should be declared EXTERNAL in the calling program.

XLOWER is the name of a user supplied REAL (DOUBLE PRECISION in the D version) function subroutine that returns the lower bounds on the bounded variables. See the complete description in §2.4. This name should be declared EXTERNAL in the calling program.

XUPPER is the name of a user supplied REAL (DOUBLE PRECISION in the D version) function subroutine that returns the upper bounds on the bounded variables. See the complete description in §2.4. This name should be declared EXTERNAL in the calling program.

X is a REAL (DOUBLE PRECISION in the D version) array of length N which must be set by the user to the value of the variables at the starting point. On exit, it contains the values of the variables at the best point found (usually the solution).

FX is a REAL (DOUBLE PRECISION in the D version) variable. If the user knows the value of the objective function at the starting point, he must set its value in FX and set FKNOWN to true. Otherwise, the user should set FKNOWN to false and need not set FX. On exit, FX contains the value of the objective function at the point X.

EPSIL is a REAL (DOUBLE PRECISION in the D version) variable, that must be set by the user to a measure of the accuracy (in the sense of the the Euclidean norm of the projected gradient, that is the orthogonal projection of the gradient of the objective function onto the feasible region defined by the bounds on the components of **x**) required to stop the minimization procedure. This value is passed to the termination subroutine VE08S/SD (see §2.6). It is not altered by the subroutine.

NGR is an INTEGER array of length 2. The first element NGR(1) must be set by the user to the maximum number of calls to ELFNCT allowed for the minimization and is not altered. A suitable value will depend on the problem, and may be chosen in the first instance between NIT(1) and 3*NIT(1). It may be necessary to increase it if the problem is highly nonlinear. NGR(2) need not be set by the user and will contain, on exit, the actual number of calls to the subroutine ELFNCT that were made during the subroutine's execution.

NIT is an INTEGER array of length 2. The first element NIT(1) must be set by the user to the maximum number of iterations that is allowed for the minimization and is not altered. The second element NIT(2) need not be be set by the user, and will contain, on exit, the actual number of minimization iterations performed by the subroutine.

FKNOWN is a LOGICAL variable. If it is set to true, FX must contain the function value at the starting point X, the first NS positions of the working vector WK must contain the individual values of the element functions at X, and the NVAR(NS+1)–1 following positions in WK must contain the gradients of the element functions. This option must be used only when the gradients of all element functions are analytically available. It is not altered by the subroutine.

FLOWBD is a REAL (DOUBLE PRECISION in the D version) variable, that must be set by the user to a lower bound on the optimal value of $F(\mathbf{x})$. In particular, it is known that $F(\mathbf{x})$ is always positive, so this provides a first lower bound. If a larger lower bound on the value of $F(\mathbf{x})$ is known, FLOWBD should be set to this bound. If no such bound is known, then the value zero should be used. It is not altered by the subroutine, provided it is nonnegative. Otherwise, the value zero is returned. Using this feature can help accelerate the convergence of the algorithm.

RELPR is a REAL (DOUBLE PRECISION in the D version) variable, that should be set by the user to the machine rounding error unit, i.e. the smallest positive ε such that $1+\varepsilon$ is still distinguishable from 1. The user may set RELPR to a negative number, and the subroutine will compute a value for it. If it is positive on input, then it is not altered by the subroutine. Otherwise, it contains, on exit, the value computed by the subroutine.

DIFGRD is a REAL (DOUBLE PRECISION in the D version) variable, that should be set by the user to the relative step size that is to be used if the gradient at the starting point is estimated by differences. If the user does not know a suitable value, or does not require such differences he must set it to a negative number, and the subroutine will use the square root of RELPR. It is unaltered if positive, and set to the square root of RELPR otherwise.

RESTRT is a LOGICAL variable that must be set to true by the user if VE10 is to be restarted and to false otherwise. In the first case, approximations to the element Hessians must be stored in the vector WK from position NS+NVAR(NS+1) onwards (see the description of WK). This preservation of second-order information may be rather efficient. Examples for which it is useful are for multiple criteria optimization, when one wishes to reoptimize a weighted sum of objectives with new weights and for a discretized problem with varying levels of mesh coarseness. RESTRT is not altered by the subroutine.

TESTGX is a LOGICAL variable that must be set to true by the user if he provides analytical gradients in ELFNCT and wishes them to be tested for accuracy at the starting point by comparing their values to difference approximations, and to false otherwise. If RESTRT is true then TESTGX is reset to false and it is as if it were false on entry. Otherwise, it is not altered by the subroutine.

HESDIF is a LOGICAL variable that must be set to true by the user if the initial Hessian approximations are to be computed by differences in the element gradient values, and to false otherwise. This option is only available if gradients are calculated in ELFNCT. When this option is not used, the element Hessians are initialized to a correctly scaled multiple of the identity matrix. If RESTRT is true then HESDIF is reset to false and it is as if it were false on entry. Otherwise HESDIF is not altered by the subroutine.

STMAX is a REAL (DOUBLE PRECISION in the D version) variable, that should be set by the user to the maximum steplength that is allowed during the minimization process. If the user does not know a suitable value, STMAX may be set to any negative number; the subroutine will then use a very large value. It is unaltered by the subroutine.

STEPL is a REAL (DOUBLE PRECISION in the D version) array of length 2. The first element STEPL(1) should be set by the user to an upper bound on the length of the first step taken by the method. If such a bound is unknown, STEPL(1) may be set to any negative number; the subroutine will then use a large default value. STEPL(1) is not altered by the subroutine. The second element STEPL(2) need not be set by the user, and on exit will contain the length of the last step taken by the algorithm.

ISTATE is an INTEGER array of length at least N+NS. It must be set as follows. For I=1,N, ISTATE(I) must be set to –1 if the I-th variable is unconstrained, to 0 if it is fixed, and to 1 if it is bounded. The values of fixed variables are never changed during the minimization process. (They may be useful, for example, when dealing with discretized problems having boundary conditions; the variables on the boundary can then be fixed, but otherwise treated as ordinary variables, which avoids the need for special expressions in the elements touching the boundary.) For I=N+1, ..., N+NS, ISTATE(I) must be set to 1 if the gradient of the (I-N)-th element function is supplied by the user, or to –1 if it has to be estimated by differences. (The difference approximations are computed by a modification of the method proposed by (Stewart 1967). Although efficiently computed, the use of approximated gradients can sometimes deteriorate the overall performance of VE10; analytical gradients

are always preferable when available.) ISTATE is used as workspace, so must be reset on a second entry.

IPDEVC is an INTEGER variable, that must be set by the user to the output device unit number for printing of messages. It is not altered by the subroutine.

IPFREQ is an INTEGER variable, that must be set by the user to specify the frequency of output by the subroutine VE10:

IPFREQ < 0: no output is generated,
IPFREQ = 0: output only at first and last iteration,
IPFREQ > 0: output every IPFREQ iteration.

This argument is not altered by the subroutine

IPWHAT is an INTEGER variable, that must be set by the user to specify the amount of output generated when output occurs:

IPWHAT = 0: iteration count, function value, norm of the projected gradient and number of function calls,
IPWHAT = 1: + step selection iterations and linesearch parameter,
IPWHAT = 2: + vector of variables,
IPWHAT = 3: + gradient vector,
IPWHAT = 4: + last step of the algorithm,
IPWHAT = 5: + element Hessian approximations

This argument is not altered by the subroutine

LWK is an INTEGER array of length 2. The first element LWK(1) must be set by the user to the length of the working array WK and is not altered. The second element LWK(2) need not be set by the user and contains, on exit, the length of the array needed to store the approximating element matrices. The length of the workspace WK that is required by the subroutine VE10 varies from problem to problem. It essentially depends on the amount of storage needed for the element Hessian approximations. Let

NV = NVAR(NS+1)-1,
NE = maximum of NVAR(I+1)-NVAR(I), I=1,NS,
NQ = maximum of 2*N and NV,

then the minimum required storage is

```
LWK(1)=2*NS+2*NV+4*N+NQ+3*NE+LWK(2)
```

The quantity NE in the relation above is the maximum number of variables appearing in any element function.

WK is a REAL (DOUBLE PRECISION in the D version) working array of length LWK(1). On successful termination, it contains a list of the element function values, gradients and approximating Hessians and is organized as follows : WK(I) contains, for

I=1,...,NS, the values of $f_i(\mathbf{x})$ for i = I,

I=NS+1,...,NS+NVAR(NS+1)-1, the NS successive element gradient vectors, one after the other,

I=NS+NVAR(NS+1),...,NS+NVAR(NS+1)+LWK(2)-1, the lower triangular parts of the element Hessian approximations, stored one after the other in row-wise fashion.

INFO is an INTEGER variable. It need not be set by the user and contains, on exit under an error condition, information about the error (see §2.5 for further details).

IFLAG is an INTEGER variable, that need not be set by the user and contains, on exit, the exit condition of the subroutine. If this flag is greater or equal to 11, an error has been detected by the subroutine (see §2.5 for further details).

2.2 Subroutine ELFNCT

We now describe the calling sequence of the subroutine that the user must provide to evaluate the element functions (and possibly their gradients).

It is worth mentioning that the points at which element functions are computed are almost always feasible, that is they satisfy whatever bounds there are. The only cases when they may not is when gradients are to be estimated by differences; in this case, the function value may be required at some points that fail to satisfy one or more bounds by a small amount.

The subroutine has the following argument list:

```
SUBROUTINE ELFNCT(K,X,FX,GX,NDIMK,NS,JFLAG,FMAX,FNOISE)
```

`K` is an `INTEGER` variable, that contains the index of the element function to be computed. It must be left unchanged by the subroutine.

`X` is a `REAL` (`DOUBLE PRECISION` in the D version) array of length `NDIMK`, that contains the values of the variables of element `K`, in the order in which they appear in the vector `INVAR`. It must be left unchanged by the subroutine.

`FX` is a `REAL` (`DOUBLE PRECISION` in the D version) variable, that must be set by the subroutine to the value of the element function at the point `X`.

`GX` is a `REAL` (`DOUBLE PRECISION` in the D version) array of length `NDIMK`. If `JFLAG` has value 2, the subroutine must set `GX` to the components $\partial f_k/\partial x_i$ of the gradient of the element function at the point `X`. If `JFLAG` has other values, `GX` need not be set.

`NDIMK` is an `INTEGER` variable, that contains the number of variables in element `K`. It is equal to `NVAR(K+1)` – `NVAR(K)` and must be left unchanged by the subroutine.

`NS` is an `INTEGER` variable, that contains the total number of elements in the problem. It must be left unchanged by the subroutine.

`JFLAG` is an `INTEGER` variable, that contains, on input, a code to describe the information expected on return from the subroutine `ELFNCT`; possible values are:

`JFLAG` = 1: the function value of element `K` is needed,

`JFLAG` = 2: the gradient and function values of element `K` are needed.

In addition, the user may wish to pass information back to `VE10` by resetting `JFLAG` according to the following rules:

`JFLAG` < 0: if `ELFNCT` sets a negative value in `JFLAG`, `VE10` will terminate abnormally, with exit condition `IFLAG=25` and `INFO` parameter equal to `JFLAG`.

`JFLAG` = 0: if `ELFNCT` sets `JFLAG` to 0, no further calls of `ELFNCT` will be made to complete the current evaluation of F; this can be useful if F value is already too large compared to `FMAX`, and if the user does not want to estimate the remaining element functions because of the cost.

If none of these events is desired, `JFLAG` should be left unchanged. The possibility of tampering with the linesearch using a zero return value for `JFLAG` should be used with caution, especially when no bounds are imposed and analytical gradients available, as it may reduce the overall efficiency.

`FMAX` is a `REAL` (`DOUBLE PRECISION` in the D version) variable, that contains the maximum value of the function F that will be accepted by the algorithm as satisfying the current linesearch criteria. It must be left unchanged by the subroutine.

`FNOISE` is a `REAL` (`DOUBLE PRECISION` in the D version) variable that the subroutine must set to an estimate of the noise (roundoff) present in the computation of the element function and element gradient.

An example of the use of ELFNCT is shown in §5.

2.3 Subroutine RANGE

We now turn to describing the way in which the user can pass to the algorithm information about a linear transformation that reduces the number of independent variables upon which each element function depends. For example the function

$$x_1^2 + (x_2 - x_3)^2$$

depends on the two variables $x_1' = x_1$ and $x_2' = x_2 - x_3$. In such cases, the convergence of the algorithm may be substantially enhanced. For each element function *k*, the user must express the dependence in terms of a full-rank matrix $\mathbf{U}_k$, with fewer rows than columns, that maps the variables x_i, *i* = INVAR(NVAR(k)) up to INVAR(NVAR(k+1)-1), that are involved in element function *k* to the smaller set. In the example of this paragraph

$$\mathbf{U}_k = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \end{pmatrix}$$

In such cases, the Hessian matrix can be held as

$$\mathbf{B}_k = \mathbf{U}_k^T \mathbf{C}_k \mathbf{U}_k$$

where $\mathbf{C}_k$ is a square symmetric matrix of order the number of rows in $\mathbf{U}_k$.

The user should therefore provide a subroutine called RANGE, that is called by VE10, and has the following argument list:

```
SUBROUTINE RANGE(K,MODE,W1,W2,NDIMK,NSUBK,NS)
```

K is an INTEGER variable, that contains the index of the element function. It must be left unchanged by the subroutine.

MODE is an INTEGER variable, that contains a code for the work to be accomplished by the subroutine :

MODE=1: the user should provide the vector $\mathbf{w}_2$ such that $\mathbf{U}_k \mathbf{w}_1 = \mathbf{w}_2$.

MODE=2: the user should provide the vector $\mathbf{w}_2$ with the smallest norm such that $\mathbf{U}_k \mathbf{w}_2 = \mathbf{w}_1$. Equivalently, $\mathbf{w}_2$ is the result of the application of the Moore-Penrose generalized inverse of $\mathbf{U}_k$ to $\mathbf{w}_1$, i.e.,

$$\mathbf{w}_2 = \mathbf{U}_k^T (\mathbf{U}_k \mathbf{U}_k^T)^{-1} \mathbf{w}_1 .$$

This computation may be performed, for example, by the HSL subroutine MB11. It is worth noticing that the vector $\mathbf{w}_1$ always belongs to the range of $\mathbf{U}_k$.

MODE=3: the user should provide the vector $\mathbf{w}_2$ such that $\mathbf{U}_k^T \mathbf{w}_1 = \mathbf{w}_2$.

MODE=4: the user should provide the vector $\mathbf{w}_2$ such that $\mathbf{U}_k^T \mathbf{w}_2 = \mathbf{w}_1$. This may be found from the formula

$$\mathbf{w}_2 = (\mathbf{U}_k \mathbf{U}_k^T)^{-1} \mathbf{U}_k \mathbf{w}_1 .$$

However, because $\mathbf{w}_1$ belongs to the range of $\mathbf{U}_k$, it is worth noting that $\mathbf{w}_2$ may be found more easily by selecting any NSUBK rows of $\mathbf{U}_k^T$, in a matrix $\hat{\mathbf{U}}_k^T$ say, and setting

$$\mathbf{w}_2 = (\hat{\mathbf{U}}_k^T)^{-1} \mathbf{w}_1 .$$

MODE must be left unchanged by the subroutine.

W1 is a REAL (DOUBLE PRECISION in the D version) array containing the input vector $\mathbf{w}_1$. It must be left unchanged by the subroutine.

W2 is a REAL (DOUBLE PRECISION in the D version) array that must be set by the subroutine to the result vector $\mathbf{w}_2$, related to $\mathbf{U}_k$ and $\mathbf{w}_1$ as required by the argument MODE. When NSUBK=NDIMK, W2 must be set equal to W1.

NDIMK is an INTEGER variable, containing the number of variables in the element. It must be left unchanged by the subroutine.

NSUBK is an INTEGER variable that must be set by RANGE to the number of rows in **U**.

NS is an INTEGER variable, containing the number of elements of the problem. It must be left unchanged by the subroutine.

If the user does not know such information it is always possible to use the following 'empty' subroutine RANGE:

```
      SUBROUTINE RANGE(K,MODE,W1,W2,NDIMK,NSUBK,NS)
      DOUBLE PRECISION W1(1),W2(1)
      NSUBK=NDIMK
      DO 1 I=1,NDIMK
      W2(I)=W1(I)
    1 CONTINUE
      RETURN
      END
```

The use of this 'empty' subroutine is nevertheless not recommended if a reduction in the number of variables internal to the element is possible, especially when the gradients are not available analytically, or when the option HESDIF=.FALSE. is used in VE10. It may, in these cases, result in much slower convergence, or possibly in no convergence at all.

2.4 Functions XLOWER and XUPPER

The fact that the I-th variable of the problem is bounded is signalled to the subroutine VE10 by ISTATE(I) being equal to 1. VE10 will then need to known the actual values of the bounds on this variable. This information is provided by two user-supplied real/double precision functions XLOWER(I) and XUPPER(I).

XLOWER(I) returns the value of the lower bound on the I-th variable, while XUPPER(I) returns the value of the upper bound on this variable. Their specification is as follows:

the single precision version

```
FUNCTION XLOWER(I)
FUNCTION XUPPER(I)
```

the double precision version

```
DOUBLE PRECISION FUNCTION XLOWER(I)
DOUBLE PRECISION FUNCTION XUPPER(I)
```

Remarks:

1. The functions XLOWER and XUPPER are only called for arguments I such that ISTATE(I)=1, i.e. they are only called for bounded variables.
2. When the I-th variable is bounded, both XLOWER and XUPPER are called for this variable : each bounded variable must be bounded below and above. If the problem only incorporates one of these bounds, the other should be supplied using a very constant positive or negative.
3. It is more efficient to declare a variable fixed (ISTATE(I)=0) than to constrain it by equal lower and upper bounds.

2.5 Error Messages

The exit condition parameter IFLAG is used to inform the user of the progress of the minimization performed within VE10. A value of IFLAG, larger than 10 indicates an abnormal termination of the subroutine. The subroutine is terminated with IFLAG set to an appropriate error index and INFO to a (sometimes) meaningful value. A complete list of these errors, together with the associated value of IFLAG and INFO is given below.

IFLAG=11 The value of variable N is not positive. INFO=N.

IFLAG=12 The machine precision constant RELPR is out of range. INFO is meaningless.

IFLAG=13 The last call to ELFNCT returned a negative value for FNOISE. INFO is meaningless.

IFLAG=14 The number of element functions NS is not positive. INFO=NS.

IFLAG=15 The starting position for the internal variables of the *k*-th element NVAR(*k*) is not positive. INFO=*k*.

IFLAG=16 The number of variables internal to the *k*-th element is not positive. INFO=*k*.

IFLAG=17 NSUBK > NDIMK on return from RANGE for element *k*. INFO=*k*.

IFLAG=18 The maximum number of iterations has been reached. INFO= maximum number of iterations.

IFLAG=19 The initial status of the *k*-th element function is incorrect: the only allowed values for ISTATE(N+*k*) are 1 (derivatives available) or –1 (derivatives not available). INFO=*k*.

IFLAG=20 The initial status of the *i*-th variable is incorrect because the only allowed values for ISTATE(*i*) are 1 (bounded), 0 (fixed), or –1 (free). INFO=*i*.

IFLAG=21 The accumulated dimension of the elements, NVAR(NS+1)–1, is not positive. INFO=NVAR(NS+1)–1.

IFLAG=22 The index of the *i*-th variable that appears in the vector INVAR is not positive, or is greater than N. INFO=*i*.

IFLAG=23 The total available work space provided in the vector WK, of length LWK(1), is too small. INFO = minimum necessary length of the vector WK.

IFLAG=24 The bounds on the *i*-th variable are inconsistent. INFO=*i*.

IFLAG=25 The user has stopped the minimization procedure by setting the element function flag to a negative value. INFO= value of the element function flag.

IFLAG=27 The directional derivative at the beginning of a linesearch is non-negative. This can be caused by an incorrect analytical gradient. INFO is meaningless.

IFLAG=28 The linesearch step became too small after *i* trials. This can be caused by an incorrect analytical gradient or a too noisy function or an exceptionally non-linear function. (This error is controlled by the variable MAXLS, representing the maximum number of linesearch trials, which is set at the beginning of the code of VE10). This can also be caused near the solution, when the accuracy requirement for termination is too high. In this case, further progress of the algorithm seems unlikely on the user's machine. INFO=*i*.

IFLAG=29 The algorithm was stopped because the difference between two successive function values along the current search direction is insignificant compared to the noise on these function values. This type of exit usually occurs quite close to the solution, and is mainly due to a too high accuracy requirement for termination. In this case, further progress of the algorithm on the user's machine is unlikely. INFO is meaningless.

IFLAG=30 An error was detected when scaling the *i*-th initial element Hessian matrix at the first step. This is usually caused by an incorrect gradient, or by incorrect problem structure specifications. INFO=*i*.

IFLAG=31 The last call to the subroutine VE10S/SD (see §2.6) returned a value for IFLAG outside the range $0 \leq$ IFLAG ≤ 10. INFO= returned value of IFLAG.

IFLAG=32 The check on the correctness of the analytical gradient at the starting point, has discovered a possible error in this computation. INFO is meaningless.

IFLAG=33 The lower bound on the objective function is not consistent with the function value at the starting point. INFO is meaningless.

IFLAG=34 The parameter IPWHAT, which controls the amount of output required by the user, is incorrect (outside the range $0 \leq$ IPWHAT ≤ 5). INFO=IPWHAT.

IFLAG=35 The subroutine stopped because the norm of the computed search direction is smaller than the given machine precision times the size of the current approximate solution. This error usually occurs when too much accuracy is required by the user for termination. INFO= iteration number when VE08 was stopped.

IFLAG=36 The subroutine is stopped because 5 successive very large steps ($> 0.999 \times$ STMAX) were taken. This is interpreted as a sign of divergence of the algorithm. This usually happens when the problem's minimum is at infinity. INFO= iteration number when VE08 was stopped.

The values of IFLAG between 1 and 10 are left free for description of normal termination (see the description of the subroutine VE10S/SD in §2.6).

2.6 Termination Criterion

One of the features of VE10A/AD is to allow the interested user to provide a problem-suited stopping criterion. There may be cases where the classical tests on the Euclidean length of the gradient are rather inadequate. One may wish to use another norm, or another stopping rule altogether. This can be done by replacing the subroutine VE10S/SD by another subroutine VE10S/SD incorporating the user's stopping criterion.

The subroutine VE10S/SD has the following argument list:

The single precision version

```
SUBROUTINE VE10S (EPSIL,NIT,NGR,DNGR,ITMAX,NFMAX,GXN,NEGCUR,
                  DIFEST,X,FX,GX,DX,DF,N,INFO,IFLAG)
```

The double precision version

```
SUBROUTINE VE10SD(EPSIL,NIT,NGR,DNGR,ITMAX,NFMAX,GXN,NEGCUR,
                  DIFEST,X,FX,GX,DX,DF,N,INFO,IFLAG)
```

EPSIL is a REAL (DOUBLE PRECISION in the D version) variable, containing a measure of the accuracy that is required by the user to terminate the minimization method successfully. It is the same value as that passed to VE10 by the user (see above). It must not be altered by VE10S/SD.

NIT is an INTEGER variable, that contains the number of iterations already performed by VE10. It must not be altered by VE10S/SD.

NGR is an INTEGER variable, that contains the number of calls already made to the subroutine ELFNCT. It must not be altered by VE10S/SD.

DNGR is a REAL (DOUBLE PRECISION in the D version) variable, containing the value of NGR divided by NS, i.e. a number representing the total number of calls to the complete objective function. It must not be altered by VE10S/SD.

ITMAX is an INTEGER variable, containing the maximum number of iterations allowed by VE10. It must not be altered by VE10S/SD.

NFMAX is an INTEGER variable, containing the maximum number of calls to the complete objective function allowed. It must not be altered by VE10S/SD.

GXN is a REAL (DOUBLE PRECISION in the D version) variable, containing the Euclidean norm of the orthonormal projection of the gradient onto the feasible region. It must not be altered by VE10S/SD.

NEGCUR is a LOGICAL variable, whose value is true if and only if some direction of negative curvature was detected on the current overall quadratic approximation to the objective function. It must not be altered by VE10S/SD.

DIFEST is a LOGICAL variable, whose value is true if and only if some of the element gradients are computed by differences. It must not be altered by VE10S/SD.

X is a REAL (DOUBLE PRECISION in the D version) array of length N, containing the current best point found by the subroutine VE10. It must not be altered by VE10S/SD.

FX is a REAL (DOUBLE PRECISION in the D version) variable, containing the value of the objective function at X. It must not be altered by VE10S/SD.

GX is a REAL (DOUBLE PRECISION in the D version) array of length N, containing the overall gradient of the

overall objective function at X. It must not be altered by VE10S/SD.

DX is a REAL (DOUBLE PRECISION in the D version) variable, containing the Euclidean length of the last step taken by the subroutine VE10. It must not be altered by VE10S/SD.

DF is a REAL (DOUBLE PRECISION in the D version) variable, containing the last improvement in objective function value realized by VE10. It must not be altered by VE10S/SD.

N is an INTEGER variable, that contains the number of variables of the problem. It must not be altered by VE10S/SD.

INFO is an INTEGER variable, that is meaningless on input. The output value is only relevant if VE10 is terminated by a nonzero output value of IFLAG. In this case, the argument INFO of VE10 will return this value to the user.

IFLAG is an INTEGER variable, that is meaningless on input. If if is decided to continue the minimization process, the value 0 should be returned in IFLAG. If it is decided that the minimization is complete, a value in the range $1 \leq$ IFLAG ≤ 10 should be returned. A return value outside the range $0 \leq$ IFLAG ≤ 10 will cause abnormal termination of VE10 with error message 31.

The default subroutine supplied with VE10 tests

1. if the Euclidean length of the projected gradient is smaller than EPSIL when DIFEST is false, or is the maximum of 10^{-4} and EPSIL when DIFEST is true, and if NEGCUR is false, or if the value of the objective function is less than or equal to EPSIL**2 (this is the normal successful exit and IFLAG is set to 1),
2. or if the maximum number of function calls has been exceeded (IFLAG is set to 2),
3. or if both the length of the last step and the last improvement in the function values are insignificant (IFLAG is set to 3).

If none of these occur, then IFLAG is set to 0.

3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided by the user (see argument WK).

Other routines called directly: VE08B/BD, VE08C/CD, VE08D/DD, VE08E/ED, VE08M/MD, VE08N/ND, VE08O/OD, VE08P/PD, VE08Q/QD, VE08R/RD; VE10F/FD, VE10G/GD, VE10I/ID, VE10J/JD, VE10K/KD, VE10L/LD, VE10S/SD, VE10U/UD, VE10V/VD, ELFNCT, RANGE, XUPPER, XLOWER.

Input/output: No input; output on device number IPDEVC.

Restrictions: $n>0$, $n_s>0$.

4 METHOD

The method that is implemented by subroutine VE10 is a partitioned quasi-Newton nonlinear least squares algorithm, as described under the name of algorithm BBO (Best Biased Overall) in (Toint 1986).

The main idea is to maintain two different models of the objective function, the first being the classical Gauss-Newton model, and the second a complete quasi-Newton one. To define the second model, a collection of small matrices approximating the Hessian matrices of each f_k is used and updated at every iteration using the BFGS or rank 1 update, depending on the curvature properties of f_k. The choice between the two models is made adaptively and aims at optimal performance for the method.

The step is determined by a trust-region approach that uses a truncated conjugate-gradient algorithm, followed by a very weak linesearch. The trust-region size is then augmented if the reduction obtained in the objective function is reasonable when compared with the predicted reduction, and reduced otherwise.

The strategy for treating bound constraints is based on the usual projection device. For a more detailed description, see (Bertsekas 1982).

References

Bertsekas,D.P. 1982. Projected Newton Methods for Optimization Problems with Simple Constraints. SIAM Journal of Control and Optimization 20(2):221-246.

Stewart, G.W. 1967. A Modification of Davidon's Minimization Method to Accept Difference Approximations of Derivatives. Journal of the ACM 14.

Toint, Ph.L. 1987. On large scale nonlinear least squares calculations. SIAM Journal on Statistical and Scientific Computing.

5 EXAMPLE OF USE

We now consider the small example problem,

$$\text{minimize } \tfrac{1}{2}\log^2(10+x_1^2+(x_2-x_3)^2)+\tfrac{1}{2}\log^2(10+x_2^2+(x_3-x_4)^2)$$

subject to the bound $x_2 \leq -1$.

The solution to this problem is at the point (0,–1,–1,–1) and the optimal function value is

$$\tfrac{1}{2}(\log^2(10)+\log^2(11)).$$

This function has four variables and two elements, namely

$$\log(10+x_1^2+(x_2-x_3)^2)$$

and

$$\log(10+x_2^3+(x_3-x_4)^2)$$

We can then set `N=4` and `NS=2.` The first element involves variables 1, 2 and 3 while the second element involves variables 2, 3 and 4; we now build the vector `INVAR` corresponding to the problem :

`I`	1	2	3	4	5	6
`INVAR(I)`	1	2	3	2	3	4
	←	el. 1	→	←	el. 2	→

In this vector, we now locate the position of the first variable of each element, and build the vector `NVAR` as follows:

`I`	1	2	3
`NVAR(I)`	1	4	7

Observe that `NVAR(NS+1)` is set to the total length of `INVAR` plus 1, as required.

The second variable is bounded, so we set `ISTATE(2)` to 1, while the other variables are unbounded, which imposes `ISTATE(I)=-1` for `I=1,3,4.` Moreover, the analytical gradients of both element functions are available, so that the last 2 components of `ISTATE` are set to 1. We therefore obtain

`I`	1	2	3	4	5	6
`ISTATE(I)`	–1	1	–1	–1	1	1

The minimum length required for the workspace `WK,` i.e. `LWK` is given by

```
LWK = 2*2+2*6+4*4+8+3*3+6 = 55
```

This count is obviously much more favourable when the dimension increases and when the element Hessians can be stored in compact form.

We now give the subroutine `ELFNCT` that would describe our simple example problem:

```
      SUBROUTINE ELFNCT(K,X,FX,GX,NDIMK,NS,JFLAG,FMAX,FNOISE)
      INTEGER K,NDIMK,NS,JFLAG
      DOUBLE PRECISION X(NDIMK),FX,GX(NDIMK),FMAX,FNOISE,TEMP
      DOUBLE PRECISION TEMP1,GMAX
      INTRINSIC ABS,MAX,LOG
      TEMP=X(2)-X(3)
      TEMP1=10.0D+0+X(1)**2+TEMP**2
      FX=LOG(TEMP1)
      FNOISE=1.0D-25*FX
      IF(JFLAG.LT.2)RETURN
      GX(1)=2.0D0*X(1)/TEMP1
      GX(2)=2.0D0*TEMP/TEMP1
      GX(3)=-GX(2)
      GMAX=MAX(ABS(GX(1)),ABS(GX(2)))
      FNOISE=MAX(FNOISE,1.0D-25*GMAX)
      RETURN
      END
```

We now wish to write down the subroutine RANGE associated with our example. First, we observe that both element functions may be reduced to dependence on two variables with the matrix

$$\mathbf{U}_k = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \end{pmatrix}.$$

As a consequence, it is easy to verify that the following subroutine RANGE satisfies all the above requirements

```
      SUBROUTINE RANGE(K,MODE,W1,W2,NDIMK,NSUBK,NS)
      INTEGER K,MODE,NDIMK,NSUBK,NS
      DOUBLE PRECISION W1(*),W2(*)
      NSUBK=2
      GO TO (1,2,3,4),MODE
C
C      1) U*w1=w2
    1 W2(1)=W1(1)
      W2(2)=W1(2)-W1(3)
      RETURN
C
C      2) U*w2=w1
    2 W2(1)=W1(1)
      W2(2)=0.5*W1(2)
      W2(3)=-W2(2)
      RETURN
C
C      3) U'*w1=w2
    3 W2(1)=W1(1)
      W2(2)=W1(2)
      W2(3)=-W1(2)
      RETURN
C
C      4) U'*w2=w1
    4 W2(1)=W1(1)
      W2(2)=W1(2)
      RETURN
      END
```

As one can see, writing the subroutine RANGE is not difficult. We notice that the simplified method for calculating $\mathbf{w}_2$ in the case where MODE=4 has been used (see §2.3).

The last thing we have to set up for our example is the bound specification. Only the second variable is bounded, so we do not have to distinguish between them for finding the bounds. Moreover, it is only bounded above, so we have to build an artificial lower bound. This gives the following functions :

```
      DOUBLE PRECISION FUNCTION XLOWER(I)
      INTEGER I
      XLOWER=-1.0D35
      RETURN
      END
```

and

```
      DOUBLE PRECISION FUNCTION XUPPER(I)
      INTEGER I
      XUPPER=-1.0D0
      RETURN
      END
```

This completes the supply of information to VE10. The problem may be solved using the following main program.

```
      INTEGER          N,NS,IPFREQ,IPWHAT,IPDEVC,INFO,IFLAG
      INTEGER          INVAR(6),NVAR(3),INTVAR(1000)
      INTEGER          ISTATE(6),NIT(2),NGR(2),LWK(2)
      LOGICAL          FKNOWN,RESTRT,TESTGX,HESDIF
      DOUBLE PRECISION FX,EPSIL
      DOUBLE PRECISION FLOWBD,DIFGRD,STMAX
      DOUBLE PRECISION X(4),WK(55),STEPL(2)
      DOUBLE PRECISION XUPPER,XLOWER
      EXTERNAL         XUPPER,XLOWER,ELFNCT,RANGE
      INTRINSIC SQRT
      DOUBLE PRECISION EPSMCH,RELPR,FD15AD
      DATA INVAR / 1, 2, 3, 2, 3, 4 /
      DATA NVAR / 1, 4, 7 /
      DATA ISTATE / - 1, 1, - 1, - 1, 1, 1 /
      DATA X / 3.0D+0, 3.0D+0, 3.0D+0, 3.0D+0 /
      N        = 4
      NS       = 2
      EPSMCH   = FD15AD('E')
      EPSIL    = 1.0D-7
      NGR(1)   = 500
      NIT(1)   = 1000
      FKNOWN   = .FALSE.
      FLOWBD   = 0.0D+0
      RELPR    = EPSMCH
      DIFGRD   = SQRT(EPSMCH)
      RESTRT   = .FALSE.
      TESTGX   = .TRUE.
      HESDIF   = .FALSE.
      STMAX    = - 1.0D+0
      STEPL(1) = - 1.0D+0
      IPDEVC   = 6
      IPFREQ   = 0
      IPWHAT   = 2
      LWK(1)   = 55
      CALL VE10AD(N,NS,INVAR,NVAR,ELFNCT,RANGE,XLOWER,XUPPER,
     *              X,FX,EPSIL,NGR,NIT,FKNOWN,FLOWBD,
     *              RELPR,DIFGRD,RESTRT,TESTGX,HESDIF,STMAX,
     *              STEPL,ISTATE,IPDEVC,IPFREQ,IPWHAT,
     *              LWK,WK,INFO,IFLAG)
      STOP
      END
```

This produces the following output.

```
 **** ITERATION    0   ( AFTER      4.00 (      8) FUNCTION CALLS)

  FUNCTION VALUE =  0.9195201D+01

  NORM OF THE PROJECTED GRADIENT =  0.102D+01

  STEP STRATEGY ITERATIONS       0     LINESEARCH PARAMETER  0.100D+01

  APPROXIMATE SOLUTION X

   0.30000D+01-0.10000D+01 0.30000D+01 0.30000D+01
```

```
**** ITERATION    9   ( AFTER     14.00 (     28) FUNCTION CALLS)

 FUNCTION VALUE =  0.5525900D+01

 NORM OF THE PROJECTED GRADIENT =  0.162D-09

 STEP STRATEGY ITERATIONS      25     LINESEARCH PARAMETER  0.100D+01

 APPROXIMATE SOLUTION X

 -0.29226D-09-0.10000D+01-0.10000D+01-0.10000D+01

 ************* EXIT OF ROUTINE VE10AD *************

 TERMINATION CRITERION OF USER SATISFIED (FLAG= 1)
```

We now consider a restart of VE10, after some previous computation. Assume that after solving the problem we wish to solve another which is the same except that the first element function is scaled by 0.9. This will require the stored Hessian approximations to be rescaled. The components of WK are

WK(1)	f_1	
WK(2)	f_2	
WK(3) to WK(5)	g_1	
WK(6) to WK(8)	g_2	
WK(9) to WK(11)	$\mathbf{C}_1$	(The Hessians are approximated
WK(12) to WK(14)	$\mathbf{C}_2$	by $\mathbf{U}_k^T\mathbf{C}_k\mathbf{U}_k$, see §2.3)

The sequence of instructions preceding the restart call to VE10 is now as follows :

```
C   RESCALE THE APPROXIMATE HESSIANS
      DO 10 I=9,11
      WK(I)=WK(I)*0.9
 10   CONTINUE
C
C   RESET THE PARAMETERS OF VE10AD
      RESTRT=.TRUE.
C
C   RESTART CALL TO VE10AD
       ...
```