



1 SUMMARY

To **minimize an objective function consisting of a sum of ‘finite 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 routine is especially effective on problems involving a large number of variables.

The objective function has the form

$$F(\mathbf{x}) = \sum_{k=1}^{n_s} f_k(\mathbf{x}_k) \quad \mathbf{x} = (x_1, x_2, \dots, x_n)$$

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, to influence the step size in the line-search procedure, and to choose whether to calculate the first approximations to the Hessian matrices

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

by finite differences.

The method is that of Griewank and Toint, Numer. Math. **39** (1982), 119-137.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Remark:** At least 8-byte arithmetic is recommended. **Calls:** FD15. **Original date:** December 1983. **Origin:** Ph. Toint, Univ. of Namur. **Licence:** A third-party licence for this package is available without charge.

2 HOW TO USE THE PACKAGE

Although there are both single and double precision versions of the routine available, the user is **strongly advised** to use the double precision version unless single precision on his or her machine actually means 8-byte arithmetic.

2.1 The Argument List

The single precision version

```
CALL VE08A (N,NS,INVAR,NVAR,ELFNCT,RANGE,XLOWER,XUPPER,
*          X,FX,EPSIL,EBOUND,NGR,NIT,FKNOWN,FLOWBD,RELPR,
*          DIFGRD,RESTR,TESTGX,HESDIF,STMAX,STEPL,ISTATE,
*          IPDEVC,IPFREQ,IPWHAT,LWK,WK,INFO,IFLAG)
```

The double precision version

```
CALL VE08AD(N,NS,INVAR,NVAR,ELFNCT,RANGE,XLOWER,XUPPER,
*          X,FX,EPSIL,EBOUND,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 routine. **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 routine. **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 routine.

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 routine.

ELFNCT is the name of a user supplied subroutine that computes the values of the element functions at a given point. See 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 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 routine that returns the lower bounds on the bounded variables. See 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 routine that returns the upper bounds on the bounded variables. See 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) required to stop the minimization procedure. This value is passed to the termination routine **VE08SD** (see §2.6). It is not altered by the routine.

EBOUND is a REAL (DOUBLE PRECISION in the D version) variable, that must be set by the user to a tolerance above which a bound is considered to be ‘nearly active’. If it is negative, the default value 10^{-5} is used. This tolerance is used in the anti-zigzagging device of (Bertsekas 1982), and has no influence on the precision with which the bound constraints will be satisfied at the solution. It is not altered by the routine.

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 $3*\mathbf{NIT}(1)$. 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 routine **ELFNCT** that were made during the routine’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 set

by the user, and will contain, on exit, the actual number of minimization iterations performed by the routine.

`FKNOWN` is a LOGICAL variable. If it is set to true, `FX` must contain the function value at the starting point \mathbf{x} , the first `NS` positions of the working vector `WK` must contain the individual values of the element functions at \mathbf{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 routine.

`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 $F(\mathbf{x})$. For instance, if it is known that $F(\mathbf{x})$ is always positive, it can help convergence to pass this information to the routine. If no bound is known, then a large negative value should be used. It is not altered by the routine.

`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 $1+\epsilon$ is still distinguishable from 1. The user may set `RELPR` to a negative number, and the routine will compute a value for it. If it is positive on input, then it is not altered by the routine. Otherwise, it contains, on exit, the value computed by the routine. **Restriction:** `RELPR` < 1.0.

`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 in the first gradient estimation by differences. If the user does not know a suitable value, he may set it to a negative number, and the routine 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 `VE08` 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 routine.

`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 routine.

`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 routine.

`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 routine will then use a very large value. It is unaltered by the routine.

`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 routine will then use a large default value. `STEPL(1)` is not altered by the routine. 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 $I=N+1, \dots, N+NS$, $ISTATE(I)$ 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 VE08; 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 routine.

$IPFREQ$ is an INTEGER variable, that must be set by the user to specify the frequency of output by the routine VE08:

$IPFREQ < 0$: 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 routine

$IPWHAT$ 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 active 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 routine

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 routine VE08 varies from problem to problem. It essentially depends on the amount of storage needed for the element Hessian approximations. Let

$$NV = NVAR(NS+1) - 1,$$

$$NE = \text{maximum of } NVAR(I+1) - NVAR(I), I=1, NS,$$

$$NQ = \text{maximum of } 2*N \text{ and } NV,$$

then the minimum required storage is

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

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

$I=1, \dots, NS$, the values of $f_i(\mathbf{x})$, $i = I$ and \mathbf{x} in X .

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

$I=NS+NVAR(NS+1), \dots, NS+NVAR(NS+1)+LWK(2)-1$, 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 routine. If this flag is greater or equal to 11, an error has been detected by the routine (see §2.5 for further details).

2.2 Subroutine ELFNCT

We now describe the calling sequence of the routine 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 routine 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 routine.

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 routine.

FX is a REAL (DOUBLE PRECISION in the D version) variable, that must be set by the routine 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 routine 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 routine.

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

JFLAG is an INTEGER variable, that contains a code to describe the information expected on return from the routine **ELFNCT**:

JFLAG = 1: function value is needed,

JFLAG = 2: gradient and function values are needed.

JFLAG < 0: if **ELFNCT** sets a negative value in **JFLAG**, **VE08** will terminate abnormally, with exit condition 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** and the steplength will be halved; 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 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 function value that will be accepted by the algorithm as satisfying the current linesearch criteria. It must be left unchanged by the routine.

FNOISE is a REAL (DOUBLE PRECISION in the D version) variable that the routine 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 less rows than columns, that maps the variables x_i , $i = \text{INVAR}(\text{NVAR}(k))$ up to $\text{INVAR}(\text{NVAR}(k+1) - 1)$, 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 routine called `RANGE`, that is called by `VE08`, 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 routine.

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

`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 .

`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 $\mathbf{U}_k^T \mathbf{w}_2 = \mathbf{w}_1$.

`MODE` must be left unchanged by the routine.

`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 routine.

`W2` is a REAL (DOUBLE PRECISION in the D version) array that must be set by the routine 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 routine.

`NSUBK` is an INTEGER variable that must be set by `RANGE` to the number of rows in \mathbf{U} .

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

If the user does not know such information it is always possible to use the following 'empty' routine `RANGE` (where REAL is replaced by DOUBLE PRECISION for the double precision version):

```

SUBROUTINE RANGE(K,MODE,W1,W2,NDIMK,NSUBK,NS)
REAL W1(1),W2(1)
NSUBK=NDIMK
DO 1 I=1,NDIMK
W2(I)=W1(I)
1 CONTINUE
```

```

RETURN
END

```

The use of this ‘empty’ routine is nevertheless not recommended, especially when the gradients are not available analytically, or when the option `HESDIF=.FALSE.` is used in VE08. 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 routine VE08 by `ISTATE(I)` being equal to 1. VE08 will then need to know the actual values of the bounds on this variable. This information is provided by two user-supplied REAL (DOUBLE PRECISION in the D version) 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

```

REAL FUNCTION XLOWER(I)
REAL 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 small or a very large constant.
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

When the exit condition of VE08, i.e. `IFLAG`, is greater than 10, this means that it has detected something going wrong in the computation. The routine 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 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=26 $F(\mathbf{x})$ is unbounded because the I -th variable appears linearly and is unbounded. $INFO = I$.
- 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 VE08). 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 this 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 this 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 routine VE08S/VE08SD (see §2.6) returned a value for IFLAG outside the range $0 \leq IFLAG \leq 10$. $INFO =$ 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 $0 \leq IPWHAT \leq 5$). $INFO = IPWHAT$.
- IFLAG=35 The routine 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 routine 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 routine VE08S/VE08SD in §2.6).

2.6 Termination Criterion

One of the features of VE08 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 routine VE08S/VE08SD by another routine VE08S/VE08SD incorporating the user's stopping criterion.

The routine has the following argument list:

The single precision version

```
SUBROUTINE VE08S (EPSIL,NIT,NGR,DNGR,ITMAX,NFMAX,GXN,NEGCUR,
*                DIFEST,X,FX,GX,DX,DF,N,INFO,IFLAG)
```

The double precision version

```
SUBROUTINE VE08SD(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 VE08 by the user (see above). It must not be altered by VE08S/VE08SD.

NIT is an INTEGER variable, that contains the number of iterations already performed by VE08. It must not be altered by VE08S/VE08SD.

NGR is an INTEGER variable, that contains the number of calls already made to the routine ELFNCT. It must not be altered by VE08S/VE08SD.

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 VE08S/VE08SD.

ITMAX is an INTEGER variable, containing the maximum number of iterations of VE08 allowed. It must not be altered by VE08S/VE08SD.

NFMAX is an INTEGER variable, containing the maximum number of calls to the complete objective function allowed. It must not be altered by VE08S/VE08SD.

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 VE08S/VE08SD.

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 VE08S/VE08SD.

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 VE08S/VE08SD.

X is a REAL (DOUBLE PRECISION in the D version) array of length N, containing the current best point found by the routine VE08. It must not be altered by VE08SD.

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 VE08S/VE08SD.

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 VE08S/VE08SD.

DX is a REAL (DOUBLE PRECISION in the D version) variable, containing the Euclidean length of the last step taken by the routine VE08. It must not be altered by VE08S/VE08SD.

DF is a REAL (DOUBLE PRECISION in the D version) variable, containing the last improvement in objective

function values realized by VE08. It must not be altered by VE08S/VE08SD.

N is an INTEGER variable, that contains the number of variables of the problem. It must not be altered by VE08S/VE08SD.

INFO is an INTEGER variable, that is meaningless on input. The output value is only relevant if VE08 is terminated by a nonzero output value of IFLAG. In this case, the argument INFO of VE08 will return this value to the user.

IFLAG is an INTEGER variable, that is meaningless on input. If it 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 $1 \leq \text{IFLAG} \leq 10$ should be returned. A return value outside the range $0 \leq \text{IFLAG} \leq 10$ will cause abnormal termination of VE08 with error message 31.

The default routine supplied with VE08 tests

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

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

2.7 Linesearch Step

Another interesting feature of VE08 is to allow the user with some knowledge of his problem to verify if the proposed value of the step along the current search direction is plausible, and to modify it, if it is not the case. This verification can be based on some relevant quantities that are passed to the user by VE08.

This verification takes the form of a routine called VE08T/VE08TD, whose argument list is as follows:

The single precision version

```
SUBROUTINE VE08T (A,X,FX,GXS,SNORM,S,N)
```

The double precision version

```
SUBROUTINE VE08TD(A,X,FX,GXS,SNORM,S,N)
```

- A is a REAL (DOUBLE PRECISION in the D version) variable, containing the steplength proposed by VE08. On output (i.e. when control is returned to VE08), it may be set to a new and more realistic stepsize for minimizing the objective function starting from \mathbf{x} in the direction \mathbf{s} , i.e. A should be such that $F(\mathbf{x}+A\mathbf{s})$ is close to its minimum with respect to A.
- X is a REAL (DOUBLE PRECISION in the D version) array of length N containing the base point for the current linesearch. It must not be altered by VE08T/VE08TD.
- 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 VE08T/VE08TD.
- GXS is a REAL (DOUBLE PRECISION in the D version) variable, containing the directional derivative at the base point in X along the direction \mathbf{s} . It must not be altered by VE08T/VE08TD.
- SNORM is a REAL (DOUBLE PRECISION in the D version) variable, containing the Euclidean length of the search direction \mathbf{s} . It must not be altered by VE08T/VE08TD.
- S is a REAL (DOUBLE PRECISION in the D version) array of length N containing the current search direction. It must not be altered by VE08T/VE08TD.
- N is an INTEGER variable, containing the number of variables of the problem. It must not be altered by

VE08T/VE08TD.

The default routine VE08T/VE08TD supplied with VE08 is just the empty routine:

The single precision version

```
SUBROUTINE VE08T (A,X,FX,GSX,SNORM,S,N)
  REAL A,X(1),FX,GSX,SNORM,S(1)
  RETURN
END
```

The double precision version

```
SUBROUTINE VE08TD(A,X,FX,GSX,SNORM,S,N)
  DOUBLE PRECISION A,X(1),FX,GSX,SNORM,S(1)
  RETURN
END
```

It should be replaced by a more complex routine VE08TD only if the user feels his knowledge of the problem is sufficient to predict rather accurately the minimum of the objective function along any search direction.

3 GENERAL INFORMATION

Use of common: None.

Other routines called directly: VE08B/BD, VE08C/CD, VE08D/DD, VE08E/ED, VE08F/FD, VE08G/GD, VE08H/HD, VE08I/ID, VE08J/JD, VE08K/KD, VE08L/LD, VE08M/MD, VE08N/ND, VE08O/OD, VE08P/PD, VE08Q/QD, VE08R/RD, VE08S/SD, VE08T/TD; FD15A/AD, ELFNCT, RANGE, XUPPER, XLOWER.

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

Restrictions: $N > 0$, $NS > 0$, $RELPR < 1.0$.

4 METHOD

The method that is implemented by routine VE08 is a partitioned quasi-Newton algorithm, as described in (Griewank & Toint 1982a), and whose convergence analysis for the convex case can be found in (Griewank & Toint 1982b). The theory for the non-convex decomposition is still an open question.

The main idea is that 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 updates, depending on the curvature properties of f_k .

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).

Acknowledgements

The author of the routine, Philippe Toint, wishes to acknowledge the useful discussions with A. Griewank, and the most kind help of J.K. Reid and I.S. Duff in setting up this specification sheet and modifying the routines to the HSL format.

References

- Bertsekas, D.P. 1982. Projected Newton Methods for Optimization Problems with Simple Constraints. SIAM Journal of Control and Optimization 20(2):221-246.
- Griewank, A. and Ph.L. Toint. 1982a. Partitioned Variable Metric Updates for Large Structured Optimization

Problems. Numerische Mathematik (39):119-137.

Griewank, A. and Ph.L. Toint. 1982b. Local Convergence Analysis for Partitioned Quasi-Newton Updates . Numerische Mathematik (39):429-448.

Griewank, A. and Ph.L. Toint. 1982c. On the Unconstrained Optimization of Partially Separable Functions. In M.J.D. Powell (editor), Nonlinear Optimization 1981. Academic Press, New-York.

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

5 EXAMPLE OF USE

We now consider the small example problem,

$$\text{minimize } \left(\sqrt{1+x_1^2+(x_2-x_3)^2} + \sqrt{1+x_2^2+(x_3-x_4)^2} \right)$$

subject to the bound $x_1 \leq -1$.

The solution to this problem is at the point $(-1,0,0,0)$ and the optimal function value is $1+\sqrt{2}$.

This function has four variables and two elements, namely

$$\sqrt{1+x_1^2+(x_2-x_3)^2}$$

and

$$\sqrt{1+x_2^2+(x_3-x_4)^2}$$

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 :

<code>I</code>	1	2	3	4	5	6
<code>INVAR(I)</code>	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:

<code>I</code>	1	2	3
<code>NVAR(I)</code>	1	4	7

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

The first variable is bounded, so we set `ISTATE(1)` to 1, while the other variables are unbounded, which imposes `ISTATE(I)=-1` for $I=2, 4$. 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

<code>I</code>	1	2	3	4	5	6
<code>ISTATE(I)</code>	1	-1	-1	-1	1	1

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

$$\text{LWK} = 2+6+4*4+8+3*3+6 = 47$$

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 routine `ELFNCT` that would describe our simple example problem:

```

SUBROUTINE ELFNCT(K,X,FX,GX,NDIMK,NS,JFLAG,FMAX,FNOISE)
DOUBLE PRECISION X(*),FX,GX(*),FMAX,FNOISE,TEMP,GMAX
INTEGER K,NDIMK,NS,JFLAG
TEMP=X(2)-X(3)
FX=SQRT(1.0D0+X(1)**2+TEMP**2)
FNOISE=1.0D-25*FX

```

```

IF(JFLAG.LT.2)RETURN
GX(1)=X(1)/FX
GX(2)=TEMP/FX
GX(3)=-GX(2)
GMAX=DMAX1(DABS(GX(1)),DABS(GX(2)))
FNOISE=DMAX1(FNOISE,1.0D-25*GMAX)
RETURN
END

```

We now wish to write down the routine 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 routine RANGE satisfies all the above requirements

```

SUBROUTINE RANGE( IELF,MODE,W1,W2,NDIMI,NSUBI,NS)
INTEGER IELF,MODE,NDIMI,NSUBI,NS
DOUBLE PRECISION W1(*),W2(*)
NSUBI=2
GO TO (481,482,483,484),MODE
C
C 1) U*W1=W2
C
481 CONTINUE
W2(1)=W1(1)
W2(2)=W1(2)-W1(3)
RETURN
C
C 2) U*W2=W1
C
482 CONTINUE
W2(1)=W1(1)
W2(2)=0.5D+0*W1(2)
W2(3)=-W2(2)
RETURN
C
C 3) U' *W1=W2
C
483 CONTINUE
W2(1)=W1(1)
W2(2)=W1(2)
W2(3)=-W1(2)
RETURN
C
C 4) U' *W2=W1
C
484 CONTINUE
W2(1)=W1(1)
W2(2)=W1(2)
RETURN
END

```

As one can see, writing routine RANGE is not difficult.

The last thing we have to set up for our example is the bound specification. Only the first 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(IVAR)
INTEGER IVAR
XLOWER=-1.0D+20
RETURN
END

```

and

```

DOUBLE PRECISION FUNCTION XUPPER(IVAR)
INTEGER IVAR
XUPPER=-1.0D+0
RETURN
END

```

This completes the supply of information to VE08. 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,EBOUND
DOUBLE PRECISION FLOWBD,DIFGRD,STMAX
DOUBLE PRECISION X(4),WK(47),STEPL(2)
DOUBLE PRECISION XUPPER,XLOWER
EXTERNAL     XUPPER,XLOWER,ELFNCT,RANGE
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
EBOUND     = 1.0D-5
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)     = 47
CALL VE08AD(N,NS,INVAR,NVAR,ELFNCT,RANGE,XLOWER,XUPPER,
*          X,FX,EPSIL,EBOUND,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   -1   ( AFTER      4.00 (      8) FUNCTION CALLS)

FUNCTION VALUE =  0.4576491D+01

NORM OF THE PROJECTED GRADIENT =  0.949D+00

STEP STRATEGY ITERATIONS      0      LINESEARCH PARAMETER  0.100D+0

APPROXIMATE SOLUTION X

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

**** ITERATION    5   ( AFTER      11.00 (     22) FUNCTION CALLS)

```

```

FUNCTION VALUE = 0.2414214D+01
NORM OF THE PROJECTED GRADIENT = 0.158D-10
STEP STRATEGY ITERATIONS      15      LINESEARCH PARAMETER 0.100D+0
APPROXIMATE SOLUTION X
-0.10000D+01 0.15816D-10 0.15816D-10 0.15816D-10
***** EXIT OF ROUTINE VE08AD *****

TERMINATION CRITERION OF USER SATISFIED (FLAG= 1)

```

We now consider a restart of VE08, 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)  $C_1$ 
WK(12) to WK(14)  $C_2$ 

```

(The Hessians are approximated by $U_k^T C_k U_k$, see §2.3)

The sequence of instructions preceding the restart call to VE08 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 VE08AD
   RESTRT=.TRUE.
C
C  RESTART CALL TO VE08AD
   ...

```