# Remark on Algorithm 630

A. BUCKLEY
Dalhousie University

Categories and Subject Descriptors: G.1.6 [**Numerical Analysis**]: Optimization; G.1.6 [**Numerical Analysis**]: Optimization—*gradient methods*

General Terms: Algorithms

Additional Key Words and Phrases: Conjugate gradient, major revision, quasi-Newton method, unconstrained minimization

## 1. PURPOSE

This paper describes some recent changes to Algorithm 630 for minimization, namely BBVSCG [2]. These changes are varied and include improvements to certain internal parts of the algorithm and modifications intended to make the code easier to use. Facilities have been added for better handling of "real-life" problems; for readers with the kind of large problems for which BBVSCG was designed, the new version is more flexible, and one recommendation (see Sec. 2) regarding use of the algorithm should be noted. Most users need to be aware of only a few of the changes.

The revisions may be grouped according to their purpose. In Section 2 we describe the major changes to the code; each group of revisions is described in one subsection. In Section 3, we revisit the subsections of Section 3 of the original paper [2], looking specifically at the changes in each. It is assumed that the reader is familiar with the earlier paper; references in *italics* are to sections of that paper.

## 2. OVERVIEW OF CHANGES

The minimization routine can be called in one of two ways: either through the simple call to BBVSCG or with the more complex call directly to BBLNIR. (Recall that BBVSCG calls BBLNIR; we refer to the algorithm generically as BBVSCG, and only mention BBLNIR when we clearly want to distinguish between the two calls.) For users with a straightforward minimization problem suitable for using the simple call to BBVSCG, little has changed. One argument is no longer

Author's address: Department of Mathematics, Statistics, and Computing Science, Dalhousie University, Halifax, Nova Scotia, Canada B3H 3J5.

necessary, although three new (probably dummy) arguments are required in the calling sequence of the function evaluation routine (see Sec. 2.1).

For more complex problems, options are available which the sophisticated user can utilize to enhance the performance or flexibility of the optimization algorithm. For example, one may change the termination test to be relative to the initial values or use an alternate form of the update procedure for the limited storage algorithm (due to Nocedal [5]), or redefine the inner product when the nature of the problem dictates something other than $(u, v) = u^T v$. Communication with and through the minimization algorithm is also better.

For large problems, it is recommended that two parameters of the routine be reset. These are ALPIS1 and QUADIN, which should both be set to 4. The method for revising them is given in Section 2.11. If more than the minimal amount of meaning is available, better performance is likely.

The algorithm BBVSCG is structured, like many other such codes, so that, when used, it normally lies *between* two other codes, each of which must be provided by the user of BBVSCG. First is the user's program segment, say USER, which calls BBVSCG to minimize a function, say $f$. Second is the code FUNCTN which evaluates $f$ and its gradient $g$ at a point $x$. This code FUNCTN normally has a predetermined calling sequence and is called by BBVSCG. This structure should be remembered.

### 2.1 Required Changes for the User of BBVSCG

There are just four, and all are minor.

(1) The argument DERV (*Sec. 3.4*) to BBVSCG is gone. This affects few users since the default now chosen is the most common case of analytic derivative evaluation. (See Sec. 2.11 if analytic derivatives are not available.)

(2) Values are now *returned* for the variables ACC, PFREQ, and MAX of *Sec. 3.5*. To emphasize this, the names of these have been changed here and in Section 3.5. For the call to BBVSCG, the meaning of each is unchanged. However, ACC becomes ACCT, which, on return, contains the CPU time used in the minimization. PFREQ becomes ITER, which returns the number of iterations done. Finally, MAX becomes FNCCT, which returns the number of points at which a function value was computed. The user must remember that the values set on entry normally have changed on return. Also, two new options are available for STATUS on entry, and values for STATUS for two more cases are included on return (see Sec. 2.9).

(3) The function evaluation routine (*Sec. 3.6*) has a slightly different calling sequence. The order of the arguments has changed, and there must now be three new array arguments. In most cases, these may be declared as dummy arrays of length 1 and henceforth ignored (see Sec. 3.6).

(4) The function evaluation routine *must pass back* a value for the argument called IFG (*Sec. 3.6*). Normally, 0 should be passed back. However, if for any reason the evaluation routine is not able to evaluate $f$ and/or $g$ at the given point, then a nonzero value should be returned (see Sec. 2.3).

### 2.2 Errors and Precautions

There were some errors in Algorithm 630 that have been fixed. If the code was used with no difficulties, they likely do not matter; if any problems were encountered, it would be worth retrying these minimizations with the new code.

In addition to fixing these errors, some precautions have been added to the code. These concern situations that are transparent to users, such as the possibility of getting an abnormally small value of $\alpha$. The result is that the code is now more robust.

### 2.3 Return Codes from FUNCTN

In Algorithm 630, there is no provision for any information to be returned from the routine FUNCTN to BBVSCG (except of course for the $f$ and $g$ values). In practical problems, however, there are at least two situations which can occur where FUNCTN should pass further information back to BBVSCG.

First, it may not be possible to evaluate $f$ and/or $g$ at the specified point; a simple example occurs when an overflow would occur. Second, it may be clear to FUNCTN that the minimization should terminate *immediately* (although there would be no reason to expect a general purpose minimization routine such as BBVSCG to be able to recognize that fact).

Therefore, the new version of BBVSCG accepts a return code (in CASE) from the function evaluation routine. This code may specify that all is OK, and that BBVSCG should carry on, that BBVSCG should cease computations (ABORT or LIMIT) by doing an immediate return to USER, or, that it was impossible (NOF, NOG, NOFG) to evaluate $f$ and/or $g$ at the given point with BBVSCG being left to decide what action to take. Note that OK, and so forth, are just named parameters for integer codes, as follows:

| | |
|---|---|
| 0 OK | $f$ and/or $g$ done; continue |
| −1 ABORT | return to USER immediately |
| −2 LIMIT | quit; function evaluation limit reached |
| −3 NOF | $f$ could not be evaluated |
| −4 NOG | $g$ could not be evaluated |
| −5 NOFG | neither $f$ nor $g$ could be evaluated |

These return codes are discussed further in Section 2.13; the action suitable when $f$ and/or $g$ could not be computed is discussed in Section 2.5.

### 2.4 Entry Codes for FUNCTN

In Algorithm 630, there were three possible values for the code IFG (*Sec. 3.6*) passed into FUNCTN, namely 1, 0, or −1 (which may be changed; see Sec. 2.13) to request FUNCTN to evaluate just $f$, both $f$ and $g$, or just $g$, respectively. (Because a value is returned in IFG, we now refer to it with the new name CASE.) A fourth code has been added. In certain problems, it is convenient to call the FUNCTN routine with no requirement to evaluate either $f$ or $g$ but in order to allow the user of BBVSCG to take some appropriate action. This is done by specifying the code 2 on entry to FUNCTN: N, the dimension; X, the current point; F and G, the current function and gradient values; and IW, RW, and DW, defined in Section 2.6, are available as usual. On return, no information is expected from FUNCTN, and

it is assumed that FUNCTN has not changed N, X, F, or G. The response code (see Sec. 2.3) from FUNCTN should be either ABORT or OK. The new code (2) is used in conjunction with the new "pass-through" STATUS option (see Sec. 2.9).

### 2.5 New Line Search

The line search subroutine has been totally replaced. This is nominally transparent to the user, but there are certain aspects of the new search which are worthy of note. The new code is based on a cubic interpolation strategy described by Lemaréchal [3] and has clearer, cleaner, and simpler logic. Lemaréchal's interpolation approach is also likely to be more robust.

The new routine is more careful about bounding the acceptable values for the line search step length parameter $\alpha$. In normal situations, the distinction here between the new and old searches is not great. However, the new routine is better at handling the case where FUNCTN contains errors in the evaluation of $f$ and/or $g$. It can now better identify this situation and report back an error code (in STATUS) to the routine USER; there were cases with Algorithm 630 which could cause infinite looping. Of course, errors should not be present in FUNCTN, but in practice mistakes do occur and it is important that the minimization routine handle such errors robustly.

The new line search also handles cases where the function and/or gradient evaluation fails. On some machines (particularly those with a small exponent range), a very moderate value of $\alpha$ may lead to an exponent overflow, particularly with functions containing exponential or penalty terms. The current line search does not fail in this situation. If the user either identifies the potential overflow in FUNCTN (or if it can be trapped on the system in use), the appropriate return code NOF, NOG, or NOFG of Section 2.3 may be returned to BBVSCG, which then assumes that the current point $x + \alpha d$ was outside the (machine-dependent) domain of definition of $f$. The line search uses this value of $\alpha$ as an upper bound, and it then reduces $\alpha$. With these improvements, one of the test functions provided is correctly minimized on certain hardware where it failed earlier because of the limited exponent range on that machine. We reiterate that the user must check for difficulties and set the return flag from FUNCTN.

### 2.6 The Function Evaluation Problem

We explained the structure of using BBVSCG earlier and noted that it lies between USER and FUNCTN. The difficulty with this scheme is that, in "real" applications, there are often various quantities which must be available to both user routines, that is, both to the routine USER which calls BBVSCG and to the code FUNCTN evaluating $f$. Since the calling sequences to both BBVSCG and the function evaluation routine are predetermined, it is generally not possible, or at least very inconvenient, to pass these quantities as arguments.

There are several possible solutions to this problem. Values shared by USER and FUNCTN may be placed in COMMON blocks constructed by the user; this may be done independently of BBVSCG. Alternately, reverse communication may be used; this feature is available with BBVSCG. There are, however, situations in which neither of these is convenient or acceptable, and, in particular, certain

problems [4] have been designed in another manner. Therefore, an additional mechanism has been added to BBVSCG.

Three new arguments are available in the calling sequence of BBLNIR, namely, IW, RW, and DW. These are not provided in the call to BBVSCG in order to keep that call as simple as possible; it is assumed that only experienced users want to use this facility. Each is a one-dimensional array, of type INTEGER, REAL, and DOUBLE PRECISION, respectively; each is passed, unchanged by BBLNIR, in the calling sequence to FUNCTN. Therefore, any values desired may be placed into these arrays by USER for use by FUNCTN. The same applies upon return from FUNCTN through BBLNIR to USER.

The function evaluation routine must therefore have a calling sequence with these three arrays added. It is the user's responsibility to ensure that the sizes and contents of these arrays are appropriate. This system is used as a "standard" by the informal French optimization group "Modulopt" [4] and has been found to be quite successful in handling very large problems in meteorology, seismic processing, and the aircraft industry. Of course many problems do not require these three arrays, but it is little effort to provide three dummy arrays.

## 2.7 The New Update Formula

Algorithm 630 uses the mixed conjugate gradient quasi-Newton strategy described by Buckley and LeNir [1]. The revised algorithm incorporates, as an option, the update strategy of Nocedal [5]. Tests on both small ($n < 100$) and fairly large ($n \gg 100$) problems indicate that the Nocedal strategy is sometimes superior. However, the choice of update strategy is not clearcut, so both are available. The strategy of [1] remains the default; the Nocedal strategy may be selected by setting the internal parameter UPDATT = 2 (see Sec. 2.11).

## 2.8 Termination Tests

There is a change to the termination tests available. One new test has been added, making a total of 4. It is now possible (and the default) to require the termination criteria to be applied *relative* to the values at the starting point. Thus, for example, the new test may be applied as $|f(x)| \leq \epsilon$ or as $|f(x)| \leq \epsilon \times |f(x_0)|$. One may apply each test independently; all tests selected must be passed before a point is accepted. A means of selecting the desired tests is provided. To illustrate (see Sec. 2.11), we can terminate when $|f(x)| \leq \epsilon \times |f(x_0)|$ and $\|g(x)\| \leq \epsilon$ with the following code:

```
INTEGER  INTS(14)
LOGICAL  LOGS(32)
REAL     REALS(2)   if appropriate, REAL should be DOUBLE PRECISION
CALL BBVALS(INTS, LOGS, REALS)   get current values
LOGS(XTGRAD) = .TRUE.            gradient test on
LOGS(XTSTEP) = .FALSE.           step test off
LOGS(XTSHXG) = .FALSE.           Shanno test off
LOGS(XTFUNC) = .TRUE.            function test on
LOGS(XRELF)  = .TRUE.            relative test for f
LOGS(XRELG)  = .FALSE.           absolute test for g
CALL BBSVAL(INTS, LOGS, REALS)   reset values
```

The integer parameters XTGRAD, and so forth, have values described in Section 2.11. The four tests are

| | |
|---|---|
| 1 *gradients* | $\| g(x_i) \| \leq \epsilon \times \| g(x_0) \|$ |
| 2 *step size* | $\| x_i - x_{i-1} \| \leq \epsilon \times \max\{1, \| x_i \|\}$ |
| 3 *Shanno's test* | $\| g(x_i) \| \leq \epsilon \times \max\{1, \| x_i \|\}$ |
| 4 *function value* | $\| f(x_i) \| \leq \epsilon \times \| f(x_0) \|$ |

If relative testing is not selected, then the terms $| f(x_0) |$ and $\| g(x_0) \|$ are replaced by 1.

## 2.9 New Status Codes

When BBVSCG returns to USER, it sets a parameter STATUS to indicate to USER if the minimization was successful. The error codes have changed in value (see BBLNIR), and additional values of STATUS (see Sec. 3.7) are now used to indicate (a) that the function/gradient evaluation failed on the *first* attempt to evaluate $f$ and/or $g$, (b) that an ABORT was requested by the function evaluation routine (see Sec. 2.3), or (c) that it is returning from a successful "pass-through" call.

New values of STATUS may be used on entry to BBVSCG as well. For some applications, it is appropriate to evaluate the function at the initial point *before* calling BBVSCG. It is redundant for BBVSCG to reevaluate $f$ at this point and undesirable if function evaluation is expensive. Thus, when STATUS = −1 (see Sec. 2.13) on entry to BBVSCG, it assumes that values for $f$ and $g$ at $x_0$ are available in F and G. The evaluation of $f$ before attempting to minimize $f$ is generally an excellent idea; it may reveal errors at an early stage which might otherwise have caused the minimization to fail.

A user may also wish to access the function evaluation routine, with no action being taken by the minimization routine; control should "pass-through" BBVSCG. If STATUS = −2 (see Sec. 2.13) on entry, FUNCTN is called and then control returns immediately to USER with STATUS = −8 to indicate that the access to FUNCTN was successful.

In the case where reverse communication is being used, it may happen that BBVSCG returns to USER with a request to evaluate $f$ and/or $g$ at some point $x$ where it is not possible to do so (see Sec. 2.3). In this case, BBVSCG may be called with STATUS = 3 (see Sec. 3.15) to indicate that the reverse call failed; then the line search routine may take appropriate action (see Sec. 2.5).

## 2.10 Inner Products

In Algorithm 630, all inner products $(u, v)$ were computed as $(u, v) = u^T v$. However, some problems intrinsically use a different inner product, possibly because the minimization problem is imbedded in the solution of some other problem, say in partial differential equations. The algorithm now allows the inner product to be changed, provided that BBLNIR is called directly. One argument of BBLNIR is INNER, which must be the name of a double precision function of the form

```
DOUBLE PRECISION FUNCTION INNER (N, U, V, NRMFLG, IW, RW, DW)
```

Here N is the dimension of the vectors U and V, which must be REAL or DOUBLE PRECISION, as appropriate. INNER must *always* return a DOUBLE PRECISION value. If NRMFLG is TRUE, INNER should return $\| U \| \equiv (U, U)^{1/2}$. The arrays IW, RW, and DW are precisely those described in Section 2.6, which are thus available to the inner product routine.

If BBVSCG is used, a routine ZZINNR is provided. This routine computes the normal Euclidean inner product and calls ZZNRM2 to compute the norm of U when NRMFLG is TRUE. Of course, ZZINNR may be used in a call to BBLNIR.

## 2.11 Changing Control Parameters

There are a number of control variables in the BBVSCG package, including some new ones such as RELF, RELG (see Sec. 2.8), and UPDATT (see Sec. 2.7). All have default values, but they may be reset if desired (hopefully only by knowledgable users). The technique described in *Section 3.16* for changing these may still be used; however, a simpler alternative is available.

The user can obtain the current settings of all values by calling a routine BBVALS(INTS, LOGS, REALS), where each of the three arguments is an array of the type clearly implied by its name (and of sizes 14, 32, and 2). One may then assign a new value to any element of these arrays. Finally, one may change the setting of those values by calling BBSVAL(INTS, LOGS, REALS); the defaults are then set to the values currently defined by these arrays.

An example of changing the termination criteria appears in Section 2.8. One can change the parameters ALPIS1 and QUADIN to 4, the termination norm to $\| \cdot \|_\infty$ and the derivative evaluation to finite differencing by calling BBVALS and setting

```
INTS(XDRVMD) = 2                    change the derivative mode
INTS(XNORM)  = 3                    change the type of norm
INTS         = (XALPS1) = 4
INTS         = (XQUADN) = 4
```

and then calling BBSVAL. The routine BBVSCG or BBLNIR is then called as usual.

In some cases, it is convenient to change these values interactively. One may

```
CALL BBRVAL(WUNIT, RUNIT)
```

A prompt then appears on the unit WUNIT (typically 6, see Sec. 3.3) requesting which logical values are to be changed. A string of up to 32 characters may then be input, T to set the value to true, F to make it false, and anything else to leave it unchanged. The $n$th character affects the $n$th value. There are then prompts requesting the integer and real values to change. This input is list directed, that is, free format and is easily terminated by a /.

The full list of values that can be changed either by calling BBVALS and BBSVAL or through BBRVAL, and the correspondence between the elements of arrays INTS, LOGS, and REALS and the control parameters described in [2] are given as part of the commentary for the routine BBVALS.

### 2.12 New Trace Control

The trace built into the package has been improved; the output shows entry and exit points for subroutines and explains the values printed more clearly. There are a couple of new trace control flags; all are documented in BBVALS.

The main points are the following. One often wants a summary of the settings used in a particular run; Trace 1 gives a summary at the beginning of the run. For large problems, printing the vectors defining the current point or the gradient is seldom desired. Thus, the trace contains no vector output unless specifically requested by Trace 9 or Trace 10. There is a trace of the termination criteria which shows the result of each of the tests (see Sec. 2.8) as it is applied in deciding whether to terminate. This can be useful to see which criterion, if more than one is being applied, is critical to termination. Normally, once one test has failed, further tests are not evaluated; if the trace is on, each desired test is evaluated and its result printed. One can trace the computations done when testing derivative calculations with DERV = 3 or 4 (*Sec. 3.13*). This can assist in isolating errors in function/derivative calculations. Finally, there is a trace in the cubic interpolation part of the line search; this can reveal information about the function behavior.

### 2.13 Settable Communication Parameters

It is possible to change the integer codes passed to and from routines in the algorithm. This applies to

—the status codes passed into BBVSCG and BBLNIR,
—the status codes returned by BBVSCG and BBLNIR,
—the function evaluation codes passed into FUNCTN,
—the return codes passed back by FUNCTN,
—the derivative evaluation codes used by FUNCTN.

For example, it is possible to pass a return code (see Sec. 2.3) from the routine FUNCTN back to BBVSCG. It is an integer value, and BBVSCG interprets it according to a set convention: If 0 is returned, BBVSCG assumes that the minimization may proceed. Similarly, when BBVSCG calls FUNCTN, it passes in an integer code indicating what values it wishes to be calculated; the value 0 means that both the function and gradient values are to be computed.

It can happen that the routine FUNCTN has already been written (and perhaps used with some other minimization routine) and, therefore, these codes (i.e., 0 for OK) passed by and back to BBVSCG may be inconvenient choices for FUNCTN. In this case, to avoid recoding in FUNCTN, it is possible to dynamically redefine these (and other) integer codes. This is documented and illustrated at the beginning (statement label 19) of the execution section of the test routine (see Sec. 3.17) distributed with the new algorithm.

## 3. THE SECTIONS OF ALGORITHM 630

We would now like to consider each section of Algorithm 630 [2] and to examine the modifications to each which should be made as a result of the changes to the code.

### 3.1 Installation Guideline

The revised code is available from the ACM Distribution Service. Both double and single precision versions are available. The distribution contains all of the routines listed in Section 3.9, plus the test program and test functions from Section 3.17. After installing and testing the entire set of routines, those pertinent only to Section 3.17 may be deleted. Multiple versions (in both precisions) of the machine-dependent routines (discussed in Sec. 3.3) are part of the distribution. The one relevant copy should be selected and the others discarded, or one of them should be appropriately modified if no suitable version is present. The distribution also contains a copy of the output which should be expected from running the test program of Section 3.17 on a machine with 13 or more digits of precision.

### 3.2 Single/Double Precision

The routine continues to be available in either single or double precision. Regardless of which version is selected, the code for the other version is included, but it is marked as commentary.

### 3.3 Machine Dependence

The two short machine dependent routines ZZMPAR and ZZSECS are still required. Versions of these routines are distributed for a number of systems. The algorithm has been successfully tested on VAX/VMS, SUN3/SunOS3.2, SUN4/SunOS 4.0, Macintosh SE,II/Absoft Fortran 2.3, and Cyber 170/NOS systems. The distributed single precision version should run unchanged on a Cyber 170 using FTN5; the double precision version should run on a SUN/4.

As before, ZZSECS may be replaced by a dummy routine if no timing information is desired, but ZZMPAR is mandatory.

To run the test program of Section 3.17, three other simple machine dependent routines are required, namely ZZDATE, ZZLCUC, and ZZTIME. Versions are distributed for the machines mentioned above. They may also be replaced by dummy routines, but then no date or time information appears in the output and the input must be in upper case.

The test program assumes that units 5 and 6 refer to the normal input and output streams (i.e., a terminal). If not, then the definition of TRMINP and TRMOUT, in a parameter statement (label 11) near the beginning of the test program, should be modified. The same definition sets unit 8 to direct output from the test to a file RESULTS.

### 3.4 Calling Sequence

The calling sequence to BBVSCG remains the same, except for the removal of DERV (see Sec. 2.1). The permissible values of STATUS on entry now include −1, −2, and 3, and new values are possible on return (see Sec. 2.9). Correspondingly, F and G may have entry values as well. The parameters ACC, PFREQ, and MAX (now ACCT, ITER, and FNCCT) also have values on return (see Sec. 2.1). The effect of ACCT may be somewhat different in that termination may be relative to

the initial values of $f$ and/or $g$ (see Sec. 2.8). There are changes to the calling sequence of FUNCTN (see Sec. 3.6).

For the calling sequence to BBLNIR, see Section 3.10.

### 3.5 Example

The call to BBVSCG remains simple. The parameter DERV is not required, so the example remains unchanged, except that the declaration of DERV and the statement setting DERV = 1 may be removed:

```
CALL BBVSCG(ROSENB, N, X, F, G, ACCT, STATUS,
            ITER, FNCCT, WORK, LWORK)
```

As mentioned in Section 2.1, there are some name changes required in the declarations.

### 3.6 Subroutine FUNCTN

There are some straightforward changes to the function evaluation routine. The motivation is to improve the ability of the minimization package to deal with very large problems (see Sec. 2.6). For many users, the changes can be easily implemented.

The form of the call to this routine is now (with IFG renamed CASE)

```
CALL FUNCTN(CASE, N, X, F, G, IW, RW, DW)
```

The arguments should be declared as

```
INTEGER           IW(*), N, CASE
REAL              RW(*)
DOUBLE PRECISION DW(*)
```

with X, F, and G being single or double precision in accordance with the version of the code being used.

The order has changed, and the array WORK has disappeared. On input, CASE must have a value of −1, 0, or 1, as before, or the value −2 (see Sec. 2.4). The name IFG has been changed to CASE to emphasize that it may now have a value on exit as well. Before returning, it *must* be set as described in Section 2.3; normally the value 0 (see Sec. 2.13) suffices.

The arrays IW, RW, and DW are included for communication in complex problems (see Sec. 2.6). For simple problems, for example, ROSENB (see Sec. 3.5), they may be dummy arrays. The types must be as stated for either the single or double precision minimization routine. The code for Rosenbrock's function is again given with the test functions of Section 3.17.

### 3.7 Return Codes

The return codes in STATUS have changed (see BBLNIR), and there are three new values. A value of −3 indicates that the minimization failed because it was not possible to evaluate the function and/or gradient at the initial point; a value of −8, that a successful "pass-through" call was completed; and a value of −9, that an abort of the minimization process is requested (see Sec. 2.3).

It should also be noted that a return code of STATUS = −5 is likely to eventually result from errors in the coding of the evaluation of $f$ and/or $g$ (see Sec. 2.5). The codes are changeable (see Sec. 2.13).

### 3.8 Size of Work

The use of the EXTRA storage in the array WORK is essentially as before. This assumes that the simple call to BBVSCG is being used. For the single precision version of BBVSCG, FUNCTN is called with the EXTRA storage passed as the single precision array RW; for the double precision version, the EXTRA storage is in the double precision array DW.

For complex problems, it may be more appropriate to explicitly use the arrays IW, RW, and DW (see Sec. 2.6). Then one must call BBLNIR directly.

There is a new option (see Sec. 2.7) for the minimization algorithm: Nocedal's update formula [5] for the quasi-Newton matrices, for which each update requires only $2n + 1$ storage locations (not $2n + 2$). This has a small effect on the computation of $m = (\text{LWORK} - 3n)/(2n + 1)$.

### 3.9 Subroutines Used

There are some additional routines to implement some new features and improve organization. The calls to several internal routines have changed. By group, the new routines are

(a) BBCUBC [BBSCUB]: This implements the cubic interpolation in Lemaréchal [3].
BBNOCE [BBSNOC]: This implements the updating strategy of Nocedal [5].
BBVALS [BBSVAL, BBRVAL]: This is used to maintain default values (see Sec. 2.11).

(b) ZZINNR: This computes the default Euclidean inner product of vectors, with a call to ZZNRM2 when needed to compute the norm of a vector (see Sec. 2.10).
ZZSCAL: This is used by ZZEVAL for possible scaling of test functions.
(ZZIRD): This has been removed in favor of the Fortran 77 intrinsic NINT.

(c) ZZDATE, ZZLCUC, ZZTIME: These machine-dependent routines are necessary only for use with the test routine TESTBB of Section 3.17.

(d) ZZDTTM, ZZFNS, ZZLENG, ZZSHFT: These are also necessary only for use with the tests of Section 3.17.

### 3.10 Internal Structure

The structure has changed only a little. The call to BBVSCG has been simplified a bit (see Sec. 2.1). The method for changing internal control parameters has been simplified (see Sec. 2.11). A means of redefining integer values passed between routines has been provided (see Sec. 2.13). The call to the inner routine BBLNIR is a bit longer.

The test routine of Section 3.17 illustrates how all of these points affect use of the routine. Readers wishing to directly call BBLNIR in order to use more features of the minimization algorithm should examine the test routine,

particularly regarding the call to BBLNIR and the change of control parameters (see Sec. 2.11).

### 3.11 Changing Defaults

This is described in Section 2.11, at least with respect to changing values dynamically. It requires simply calling BBVALS and BBSVAL or BBRVAL. To make permanent changes, one should now change the parameter statements in the routine BBVALS, rather than in BBDFLT.

### 3.12 Print Control

There are some minor changes here. The output is a bit more compact. One can retain printing of the gradient without printing the point by setting the new argument POINT to .FALSE. (see Sec. 2.11). This is implemented by calling the entry point ZZP1ST(UNIT, GRAD, POINT, PFREQ). The last argument of ZZPRNT is now integer rather than logical and is documented internally. It is also possible to produce output on a second unit, independently of the first, by calling an identical entry point ZZP2ST(UNIT2, GRAD2, POINT2, PFREQ2). Each call to ZZPRNT generates output on either or both units as set.

### 3.13 Function/Gradient Evaluation

The call to the evaluation routine has changed (see Sec. 3.6). The most common case of analytic derivative calculation is now automatic, so DERV is no longer in the calling sequence of BBVSCG or BBLNIR (but it may be changed; see Sec. 2.11). There is a fourth option: If DERV = 4, then the option DERV = 3 is implemented, but only on the first call to ZZEVAL.

The count of function evaluations done is now returned as an argument in BBVSCG. One can trace the computations done when testing derivative calculations (see Sec. 2.11); it is accomplished by calling the revised entry point ZZESET(TRF, TRG, TRTEST, UNIT) in ZZEVAL.

### 3.14 Termination Control

There are changes to the method of terminating a minimization (see Sec. 2.8). Four termination criteria are now available, each of which may be applied independently. The call to the ZZTERM entry point has changed to ZZTSET(NORM, TESTS, TRACE, TRU). Here TESTS is a string of 4 characters, each of which is T or F to control one of the 4 traces. The default of 'FTTF' effectively provides the same termination as Algorithm 630.

It is now possible (and the default) to make termination relative to the initial values of $f$ and $g$, which is often desirable since it removes the effect of scaling $f$ and/or $g$ by a scalar. Application of the termination criteria may be traced (see Sec. 2.12).

### 3.15 Reverse Communication

This feature is still available, although it may not perhaps be used as often, given the new features described in Section 2.6. However, its use is the same, with one addition. If it is not possible to evaluate $f$ and/or $g$ at the specified point, BBVSCG may be recalled with STATUS = 3 (see Sec. 2.9).

### 3.16 Control Parameters for BBLNIR

These are still available and may be changed if desired. There are further controls available, and the method for changing them is simpler (see Sec. 2.11).

### 3.17 Test Program and Storage Requirements

A sample program is distributed with BBVSCG; as in [2], this may be used to verify that BBVSCG is installed and running correctly, and it also demonstrates some of the points raised elsewhere in this paper. Some changes have been made to this program; it now requires some further subroutines (see Sec. 3.9), all of which are provided.

The program requires input. For this, it writes prompts on unit 6, assuming that it is the terminal, and it reads a response from unit 5, which is again normally the terminal. To conduct the normal test of all cases with all functions, the only input required is to provide an end-of-file (or an empty file) on unit 5. All output generated by the test goes to unit 8. If the assignment of the units is inconvenient, they may easily be changed (see Sec. 3.3).

The changes to TESTBB also allow selective testing of problems and they permit a change of settings described in Section 2.11. This is documented internally in TESTBB.

#### ACKNOWLEDGMENTS

The author would like to thank INRIA (l'Institut National de Recherche en Informatique et en Automatique) in Rocquencourt, France for providing the facilities where the revisions to this code were developed during the author's sabbatical leave. Particular thanks are also due to M. Claude Lemaréchal, who collaborated on many of these changes and who provided several rather large "real-life" test problems. The author would also like to acknowledge the support of the Natural Sciences and Engineering Research Council of Canada under operating grant A2694.

#### REFERENCES

1. BUCKLEY, A., AND LENIR, A. QN-like variable-storage conjugate gradients. *Math. Program. 27* (1983), 155–175.
2. BUCKLEY, A., AND LENIR, A. ALGORITHM 630: BBVSCG—A variable-storage algorithm for function minimization. *ACM Trans. Math. Softw. 11*, 2 (June 1985), 103–119.
3. LEMARÉCHAL, C. A view of line searches. In *Optimization and Optimal Control*, A. Auslender, W. Oettli, and J. Stoer, Eds. Springer-Verlag, Heidelberg, 1981.
4. LEMARÉCHAL, C. Using a Modulopt optimization code. I.N.R.I.A., Rocquencourt, 78153 Le Chesnay Cedex, France.
5. NOCEDAL, J. Updating quasi-Newton matrices with limited storage. *Math. Comput. 35*, 151 (1980), 773–782.

Received December 1988; accepted December 1988