Updates to TOMS algorithm 630

A.G. Buckley

INRIA

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP105
78153 Le Chesnay Cedex
France
Tel.(1) 39 63 55 11

Rapports Techniques

N° 91

UPDATES TO TOMS ALGORITHM 630

Albert G. BUCKLEY

NOVEMBRE 1987

Updates to TOMS Algorithm 630
(for limited memory quasi-Newton function minimization)

*Albert G. Buckley**
INRIA, Rocquencourt
78153 Le Chesnay Cedex
(1) 39 63 55 11

Juin, 1987

* *au congé sabbatique de la*
Department of Mathematics,
Statistics and Computing Science
Dalhousie University
Halifax, N. S. B3H 3J5
C A N A D A

Updates to TOMS Algorithm 630
(for limited memory quasi-Newton function minimization)

Abstract

This paper describes a number of changes which have been made to the TOMS Algorithm published earlier, namely *BBVSCG—A Variable-Storage Algorithm for Function Minimization.* These changes correct some problems with the published code, and also incorporate a number of improvements.

MISES A JOUR POUR L'ALGORITHME TOMS 630

Cet article définit un certain nombre de modifications apportées à un algorithme TOMS publié précédemment, à savoir BBVSCG-Un algorithme d'optimisation à encombrement variable. Ces modifications éliminent quelques difficultés du code tel qu'il était publié, et ajoutent quelques améliorations. (n.b. BBVSCG est le code original qui a donné naissance à M1GC2).

Updates to TOMS Algorithm 630
(for limited memory quasi-Newton
function minimization)*

* Submitted to ACM Transactions on Mathematical Software. It also appears as Technical Report 1987CS–4 in the Department of Mathematics, Statistics and Computing Science, Dalhousie University, Halifax N.S. B3L 3C4, Canada.

REMARK ON ALGORITHM 630

1. Purpose

This paper will describe some recent changes to the minimization algorithm BBVSCG [bbl] published earlier. These changes are varied, and include, amongst others, improvements to certain internal parts of the algorithm, and modifications intended to make the code easier to use. Facilities have also been added for better handling of "real-life" problems; for those persons having the large kind of problems for which BBVSCG has been designed, the new version is more flexible. The changes are by no means trivial in some respects, but most users will be affected by few of them.

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

2. Overview of Changes

The minimization routine could 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 will 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 necessary, although three new (probably dummy) arguments are required in the calling sequence of the function evaluation routine. These points are discussed in Section 2.=vscg.

For tougher problems, more options are available which the sophisticated user can utilize to maximize the performance of the optimization algorithm on large problems. These include, amongst others, changing the termination test to be relative to the initial values, using a new form of update procedure for the limited storage algorithm (due to Nocedal [noc]), and redefining 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.

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. The first of these is the user's principal program, that is to say, the program segment, say USER, which calls BBVSCG with a request to minimize a function, say f. The second is the code FUNCTN which evaluates f and its gradient g at a specified point x. This code FUNCTN normally has a predetermined calling sequence and is called by BBVSCG. This structure should be remembered in the comments which follow.

2.=vscg Required Changes for the User of BBVSCG

There are just four, and all are minor. First, the call to BBVSCG has changed in that the argument DERV (*Section 3.4*) has disappeared. This makes the call slightly simpler, and will affect few users since the default now chosen is the most common case of analytic derivative evaluation. (See Section 2.=cip if analytic derivatives really are not available.)

Second, values are now returned for the variables ACC, PFREQ and MAX of *Section 3.6*. 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 the same. However, ACC becomes ACCT, which, on return, will contain the CPU time used in the minimization. PFREQ becomes ITER, which will return the number of iterations done. Finally, MAX becomes FNCCT, which returns the actual number of function values computed. These changes should make very little difference in using BBVSCG, except that the user must remember that the values set on entry will very likely have changed on return. Also, one new option is available for STATUS on entry, and values for STATUS for two more cases are included on return (Section 2.=stat).

Third, the function evaluation routine (*Section 3.6*) has a slightly different calling sequence. The order of the arguments has changed a bit, but more significantly, there must now be three new array arguments. If the user has no need of these, they may be declared as dummy arrays of length 1 and henceforth ignored. See Section 3.6.

Finally, the function evaluation routine *must* pass back a value for the argument called IFG (*Section 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 non-zero value should be passed back, as discussed in Section 2.=rtc.

2.=bugs Bugs and Precautions

There were (not surprisingly of course) some bugs in the earlier version. These have been fixed. There is no point in describing these here; if the code was used with no problems, they do not matter, and if any problems were encountered, it would be simplest to try these problems with the new code.

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

2.=rtc Return Codes from FUNCTN

In the present version of BBVSCG, 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 it is appropriate for FUNCTN to pass further information 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 at the current point. Second, a situation can arise where it is 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 looks for a return code from the function evaluation routine. This code may specify that all is *OK*, and that BBVSCG should carry on; that BBVSCG should *ABORT* 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*, etc. 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 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

The return codes for indicating these situations will be discussed further in Section 2.=scp; the kind of action which is suitable if f and/or g could not be computed is discussed in Section 2.=ls.

2.=ec Entry Codes for FUNCTN

In the earlier version of BBVSCG, there were three possible values for the code IFG (*Section 3.6*) passed into FUNCTN. These were 1, 0, or -1 to request FUNCTN to evaluate just f, both f and g, or just g, respectively. A fourth code has been added. In certain problems, it is convenient to be able to call the FUNCTN routine without any 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. On entry, X will contain the current point, F and G the current function and gradient values, and IW, RW and DW defined in Section 2.=irwd will be available as usual. On return, no information will be expected from FUNCTN, and it will be assumed that FUNCTN has not changed N, X, F or G.

2.=ls New Line Search

The line search subroutine has been totally replaced. This is in fact nominally transparent to the user, but there are certain points about the new search which are worthy of note. The new code is based on a cubic interpolation strategy described by Lemaréchal [lem]. The logic is clearer, cleaner and simpler than the earlier code. It also uses a method of performing the cubic interpolation which was given by Lemaréchal and is likely to be more robust.

But a more significant feature of the new routine is that is it much more careful about remembering and using bounds on the values which the line search step length parameter α may take. 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 coding of the evaluation of f and/or g. It will generally now be able to identify this situation and report back to the routine USER through the parameter STATUS that an error has occurred, whereas there were cases with Algorithm 630 which could cause infinite looping. Of course this should not happen, but in practice mistakes do occur in the evaluation routine and it is important that the minimization routine handle such errors as well as possible.

The new line search is also able to handle many cases where the function and/or gradient evaluation fails, as mentioned earlier. Particularly on some machines with a limited exponent range, it can happen that a value of α is computed which is a bit too large, so that an exponent overflow may occur. This is particularly prone to happen with functions containing exponential or penalty terms.

The current line search will not fail in this situation, whereas the earlier one had no way of dealing with it. 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 (see Section 2.=rtc) may be returned to BBVSCG. It will then assume that α was simply

too large and that the current point $x+\alpha d$ was outside the domain of (machine-defined) definition of f. The line search will remember this value of α as an upper bound, and it will then reduce α. With these improvements, one of the test functions provided will be correctly minimized on certain hardware where it failed earlier due to the extremely limited exponent range on that machine. We should reiterate that it is up to the user to check for difficulties and to set the return flag from FUNCTN.

2.=irdw 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 schema is that, in "real" applications, there are often various quantities which must be available to both user routines, i.e. 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, and in particular, certain problems have been designed in another manner. Therefore an additional mechanism has been added to the BBVSCG package.

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 sophisticated users will need to use this facility. Each is a 1-dimensional array, of type INTEGER, REAL and DOUBLE PRECISION, respectively. These arrays are accepted by BBLNIR, but are passed without any change whatsoever to the function evaluation routine FUNCTN in its calling sequence. Therefore any values desired may be placed into these arrays by USER and they will be available to FUNCTN. The same applies upon return from FUNCTN through BBLNIR to USER.

Notice therefore that the function evaluation routine must have an extended calling sequence, namely with these three arrays added. It is the user's responsibility to ensure that the sizes and contents of these arrays are appropriate. Note that this system is used as a "standard" by the informal French optimization group "Modulopt" [mopt] 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 will not require these three arrays, but it is little effort to provide three dummy arrays.

2.=upd The New Update Formula

Algorithm 630 uses the mixed conjugate gradient quasi-Newton strategy described by Buckley and LeNir in [vscg]. The revised algorithm incorporates, as an option, the update strategy of Nocedal [noc]. Tests which have been performed on both small ($n < 100$) and fairly large ($n > 1000$) problems indicate that the Nocedal strategy is sometimes superior. However, the choice of update strategy is not clearcut, and therefore both are available in the revised algorithm. The strategy of [vscg] remains the default, while the Nocedal strategy may be optionally selected by setting the internal parameter UPDATT = 2, as discussed in Section 2.=cip.

2.=term Termination Tests

There has been a change to the termination tests available. First, one new test has been added, making 4 in total. Second, it is now possible (but optional) 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 $|f(x)| \leq \epsilon \times |f(x_0)|$. Third, one may apply each of these tests selectively, that is, each test may be applied or ignored, independently of the others. A mechanism for selecting the desired tests is of course provided. To illustrate, following the description of Section 2.=cip, we can select the two tests $|f(x)| \leq \epsilon \times |f(x_0)|$ and $\|g\| \leq \epsilon \times \|g_0\|$ with the following code:

```
INTEGER  INTS(13)
LOGICAL  LOGS(33)
REAL     REALS(2)
CALL BBVALS ( INTS, LOGS, REALS)   get current values
LOGS(22) = .TRUE.          gradient test on
LOGS(23) = .FALSE.         step test off
LOGS(24) = .FALSE.         Shanno test off
LOGS(25) = .TRUE.          function test on
LOGS(26) = .TRUE.          relative test for f
LOGS(27) = .TRUE.          relative test for g
CALL BBSVAL (INTS, LOGS, REALS )   reset values
```

The set of four tests is the following, where the name is that used in BBSVAL to determine the index (in Section 2.=cip) used to control that particular test:

TGRAD	$\|g(x_i)\| \leq \epsilon \times \|g(x_0)\|$
TSTEP	$\|x_i - x_{i-1}\| \leq \epsilon \times \max\{1, \|x_i\|\}$
TSHXG	$\|g(x_i)\| \leq \epsilon \times \max\{1, \|x_i\|\}$
TFUNC	$\|f(x_i)\| \leq \epsilon \times \|f(x_0)\|$

If relative testing is not selected, then the terms $|f(x_0)|$ and $\|g_0\|$ are replaced by 1.

2.=stat New Status Codes

When BBVSCG returns to USER, it sets a parameter STATUS to indicate to USER whether the minimization was successful or not. Additional values of STATUS (see Section 3.7) are now used to indicate (a) that the function/gradient evaluation failed on the first attempt to evaluate f and/or g, or (b) that an ABORT was requested by the function evaluation routine, as discussed in Section 2.=rtc.

As well, a new value of STATUS may be used on entry to BBVSCG. For some applications, it is appropriate to evaluate the function at the initial point *before* calling the minimization routine BBVSCG. It is therefore redundant for BBVSCG to re-evaluate f at this point, especially if function evaluation is expensive. Thus, when STATUS = -1 on entry to BBVSCG, the routine will assume that values for f and g are already available in F and G. It should be pointed out that the evaluation of f before attempting to minimize f is generally an excellent idea; it will often reveal errors at an early stage which might otherwise have caused the minimization to fail.

In the case where reverse communication is being used, it may happen that BBVSCG returns to USER with a request to evaluate f and g at some point x where it is not

possible to do so (as discussed in Section 2.=rtc). In this case, BBVSCG may be called with STATUS = 3 to indicate that the reverse call failed; then the line search routine may take appropriate action, as discussed in Section 2.=ls.

2.=inn Inner Products

In the original version of BBVSCG, all inner products of the form (u, v) were computed internally as $(u, v) = u^T v$. However, some (particularly large) problems intrinsically use a different inner product. This may be because the minimization problem in imbedded in the solution of some other problem, say in partial differential equations. The current version 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 the norm of U, defined as $\| U \| \equiv (U,U)^{1/2}$. The arrays IW, RW and DW are precisely those described in Section 2.=irdw, which are thus available to the inner product routine.

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

2.=cip Changing Internal Parameters

As described in [bbl], there are a number of control variables in the BBVSCG package. All have default values, but they may be reset if desired (hopefully only by knowledgable users). The technique already described (in *Section 3.16*) for changing these may still be used; however an alternate is available which will generally be simpler. Note also that there are some new control parameters, including RELF, RELG and UPDATT. RELF and RELG pertain to Section 2.=term, while UPDATT is discussed in Section 2.=upd.

The user can obtain the current default setting of all appropriate values by calling a routine BBVALS(INTS, LOGS, REALS), where each of the three arguments is an array of the type clearly implied by the names (and of sizes 13, 33 and 2). One may then assign a new value to any of the elements of these arrays. Finally, one may change the setting of the defaults for those values by calling BBSVAL(INTS, LOGS, REALS); the defaults are then reset to the values currently defined by these arrays. The correspondence between the elements of these arrays and the control parameters described in [bbl] is given below.

For example, the following code suffices to change the termination norm to $\| \cdot \|_\infty$ and the derivative evaluation to finite differencing:

```
INTEGER  INTS(13)
LOGICAL  LOGS(33)
REAL     REALS(2)
CALL BBVALS ( INTS, LOGS, REALS)   get current values
INTS(1) = 2        change the derivative mode
INTS(2) = 3        change the type of norm
CALL BBSVAL (INTS, LOGS, REALS )   reset values
```

Then call BBVSCG or BBLNIR as usual. Note that the declaration of REALS(2) should be DOUBLE PRECISION if appropriate.

In some cases, it is most convenient to be able to change these values interactively. For this reason, an alternate call to BBVALS is given. One may

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

A prompt will then appear on the unit WUNIT (typically 6) requesting which logical values are to be changed. A string of up to 33 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 will then be prompts requesting the integer and real values to change. This input is list-directed, i.e. free format, and is most easily terminated by a /.

The full list of values which can be changed either by calling BBVALS and BBSVAL, or through BBRVAL follows. Note that the "Use" column for the logical values specifies the action that takes place if the corresponding logical is set to .TRUE.. In the default column, T is used for .TRUE. and F is for .FALSE..

Array	*Index*	*Name*	*Default*	*Use*
INTS	1	DERVMD	1	Control of derivative mode; see Section 3.13.
INTS	2	NORM	2	Control of norm; see *Section 3.14*.
INTS	3	SCALE	0	Scaling to apply to f; see *Section 3.13*.
INTS	4	TRACUN	6	Unit for output of traces in BBLNIR (See LOGS(1-15)).
INTS	5	TRACEU	6	Unit for output of f and g from ZZEVAL (see LOGS(11,12)).
INTS	6	UNIT	6	Unit for output from ZZPRNT (See *Section 3.12*).
INTS	7	TTRACU	6	Unit for output of trace in ZZTERM (see LOGS(27)).
INTS	8	METH	0	See BBLSET in listing of BBLNIR.
INTS	9	QUADIN	1	See BBLSET in listing of BBLNIR.
INTS	10	ALPIS1	1	See BBLSET in listing of BBLNIR.
INTS	11	SCGAMM	1	See BBLSET in listing of BBLNIR.
INTS	12	HTEST	1	See BBLSET in listing of BBLNIR.
INTS	13	UPDATT	1	Flag to control Nocedal Updates; see Section 2.=upd.
LOGS	1–15	TRACES	F	The 15 trace flags for BBLNIR; see BBLSET in the listing.
LOGS	16	TRF	F	Trace the evaluation of f in ZZEVAL.
LOGS	17	TRG	F	Trace the evaluation of g in ZZEVAL.
LOGS	18	TTRACE	F	Trace the termination tests.
LOGS	19	TRTEST	F	Trace the computation of finite differences in test mode (See *Section 3.13*).
LOGS	20	GRAD	T	Include the gradient in output from ZZPRNT.
LOGS	21	POINT	T	Include the point x in output from ZZPRNT.
LOGS	22	TGRAD	F	Include the gradient test for termination.
LOGS	23	TSTEP	T	Include the step test for termination.
LOGS	24	TSHXG	T	Include Shanno's test for termination.
LOGS	25	TFUNC	F	Include the function test for termination.
LOGS	26	RELF	T	Make function tests relative to $f(x_0)$.
LOGS	27	RELG	T	Make gradient tests relative to $g(x_0)$.
LOGS	28	FQUAD	F	See BBLSET in listing of BBLNIR.
LOGS	29	DIAGNL	F	See BBLSET in listing of BBLNIR.
LOGS	30	SHANNO	F	See BBLSET in listing of BBLNIR.

LOGS	31	FROMRS	F	See BBLSET in listing of BBLNIR.
LOGS	32	FORCEF	T	See BBLSET in listing of BBLNIR.
LOGS	33	FLETSC	F	See BBLSET in listing of BBLNIR.
REALS	1	RO	0.2	See BBLSET in listing of BBLNIR.
REALS	2	BETA	1.0	See BBLSET in listing of BBLNIR.

2.=trc New Trace Control

The trace which has been built into the package has been improved. There are a couple of new trace control flags; these are documented internally in BBLNIR.

The main points are the following. One often wants a summary of the settings used in a particular run; turning on Trace 1 will now give a summary at the beginning of the run. For large problems, it is seldom desired to print the vectors defining the current point or the gradient. Thus the trace contains no vector output unless specifically requested by turning on Trace 9 or Trace 10. In general, the output has been improved to more clearly show entry and exit points for subroutines, and to more clearly explain the values being printed.

A trace has been added to the termination routine. If it is turned on, then the result of each of the tests (see Section 2.=term) applied to ascertain whether to terminate is printed. This is sometimes useful to see which criterion, if more than one is being applied, is critical to termination. Note that normally, once one test has failed, further tests are not evaluated; however, if the trace is on, each desired test is evaluated and its result printed.

It is also possible to trace the computations done when testing derivative calculations with DERV = 3 or 4 (*Section 3.14*). This can assist in isolating errors in function/derivative calculations. Finally, there is also a trace in the cubic interpolation part of the line search; this can reveal information about the function behaviour.

The traces available can be seen by looking at the entries in the array LOGS in Section 2.=cip.

2.=scp Settable Communication Parameters

As explained in Section 2.=rtc, it is possible to pass a return code from the routine FUNCTN back to BBVSCG. It is of course an integer value, and BBVSCG will interpret it according to certain assumptions. For example, if the value returned is 0 (in the default case), BBVSCG will assume that all is *OK* and that the minimization may proceed. Similarly, when BBVSCG calls FUNCTN, it will pass in an integer code indicating what values it wishes to be calculated. For example, if it passes in the integer 0, it means that both the function and gradient values are to be computed.

However, it may 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 redefine the integer codes which BBVSCG will use. This is done by setting new values of the codes by calling the entry points BBLFDF and BBLRDF. To be specific, one may change the codes to be *passed to* FUNCTN by calling

```
CALL BBLFDF ( JUSTF, JUSTG, BOTH, NOOP)
```

where the names JUSTF, etc. have been set to the desired integer codes for doing just f, just g, both or neither, respectively. Consider the use of BBVSCG with the Modulopt codes of the French group [mopt]. In this case, one would

```
CALL BBLFDF ( 2, 3, 4, 1 )
```

This defines 2 as the code to evaluate f, 3 to evaluate both and 4 as the code to evaluate g. Note that 1 is added as the "no-operation" code, as discussed in Section 2.ec.

For the code *returned by* FUNCTN, take an example, again using the Modulopt norm. Their standard is that FUNCTN should return a code of:

1	everything is fine; proceed
0	abort the minimization immediately
-1	error: the function evaluation limit was reached
-2	error: the function could not be evaluated
-3	error: the gradient could not be evaluated
-4	error: neither f nor g could be evaluated

To use these codes with BBVSCG, one makes the call

```
CALL BBLRDF ( 1, 0, -1, -2, -3, -4 )
```

3. The Sections of Algorithm 630

We would now like to consider the sections of Algorithm 630 [bbl] one by one, 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 now available from the author. It may be requested either on a magnetic tape, or via electronic mail (either from `dalcs!buckley` on the UUCP network, or as `buckley@cs.dal.cdn` on the CDN network.). Upon request, a choice of the following will be sent.

1 • *Machine dependent source tailored for a particular machine.* It should be possible to compile and run this source, as is. Such source is available for a variety of machines, but not for all.

2 • *Machine independent source.* This is the source from which the machine dependent code is derived. A few minor changes are required in order to produce code to run on a specific system. A program designed to produce machine dependent source code will be provided.

3 • *A set of "include" files* which are used to maintain the source. If this is requested, then the source mentioned above will be sent without replacing the include files which are used. Otherwise, the source will be shipped with all include files having already been incorporated into the source code.

4 • *A set of programs to test the code*, as discussed in Section 3.17.

5 • *A conversion routine*, as discussed in Section 3.2.

3.2 Single/Double Precision

The routine continues to be available in either single or double precision. In requesting the code above, one must indicate which version is desired. However, regardless of which version is selected, the code for the other version will be included, but it will be

marked as commentary. If desired, a Fortran '77 program to convert from one precision to the other is available as well.

3.3 Machine Dependence

The two machine dependent routines ZZMPAR and ZZSECS are still required. Notice that versions of these routines are now available for the Honeywell Multics System, and for the Apple Macintosh using Microsoft (Absoft) Fortran. Furthermore, the algorithm has been successfully tested on both of the machines, in addition to VAX/VMS, VAX/Berkeley Unix 4.3, and the CDC Cyber 170.

Notice that, as before, ZZSECS may be replaced by a dummy routine if no timing information is desired. However ZZMPAR is mandatory.

If one wishes to run the test program of Section 3.17, there are two other machine dependent routines required, namely ZZDATE and ZZTIME. Both are very simple, and versions are available for the machines mentioned above. As with ZZSECS, they may be replaced by dummy routines, but then no date or time information will appear in the output.

Note also that the test program assumes that units 5 and 6 refer to the normal input and output streams (i.e. the terminal). If that is not the case, then the definition of TRMINP and TRMOUT, in a parameter statement near the beginning of the test program, should be modified.

Finally, generic versions of all machine-dependent routines are available, along with a program to produce machine specific code from the generic routines. If the machine independent code is requested, then this program will be provided as well.

3.4 Calling Sequence

The calling sequence to BBVSCG remains the same, except for the removal of *DERV*, as in Section 2.=vscg. The permissible values of *STATUS* on input now include -1 and 3, as in Section 2.=stat. 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, as in Section 2.=vscg. The effect of *ACCT* may be somewhat different in that termination may be relative to the initial values of f and/or g (see Section 2.=term). Note also the changes to the calling sequence of FUNCTN, as in Section 3.6.

For the calling sequence to BBLNIR, see Section 3.10.

3.5 Example

To keep the call to BBVSCG simple, there are no changes, except that the parameter DERV is no longer required. Thus the example remains unchanged, except that the declaration of DERV and the statement setting DERV = 1 may be removed, and the call may be changed to

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

One could also of course add statements to print the values of ACCT, ITER and FNCCT which are now returned.

3.6 Subroutine FUNCTN

There are some straightforward changes to the function evaluation routine. The motivation for these is to improve the ability of the minimization package to deal with

very large problems, as discussed in Section 2.=irdw. For many users, the changes can be implemented with almost no effort.

The form of the call to this routine is now

```
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 which is being used.

The order has changed a bit, and the work array WORK has disappeared. Note that CASE is now first. On input, CASE should be assumed to have a value of -1, 0 or 1, as before, although is may now have a value 2 as well, as in Section 2.=ec. The name has been changed to emphasize that it now may have a value on exit as well. On output, it *must* be set as described in Section 2.=rtc; normally returning the value 0 will suffice. (Also see Section 2.=scp.)

The arrays IW, RW and DW are included for communication in large problems, as in Section 2.=irdw. For simple problems, such as the ROSENB example mentioned, they can just be declared as dummy arrays, as in the example below. Note that the types must be as stated, regardless of whether one is using the single or double precision minimization routine. For example, the (single precision) code to evaluate Rosenbrock's function would now be as follows:

```
SUBROUTINE ROSENB ( CASE, N, X, F, G, IW, RW, DW )

INTEGER           CASE, N
REAL              X(N), G(N), F

INTEGER           IW(*)
REAL              RW(*)
DOUBLE PRECISION  DW(*)

LOGICAL  DOF, DOG
REAL     X1,  W1, W2

REAL        ONE,          TWO
PARAMETER ( ONE = 1E0,    TWO = 2E0 )
REAL        R100,         R200,         R400
PARAMETER ( R100 = 100E0, R200 = 200E0, R400 = 400E0 )

DOF = CASE .GE. 0
DOG = CASE .LE. 0

X1 = X(1)
W1 = ONE - X1
W2 = X(2) - X1*X1

IF ( DOF ) THEN
   F = R100*W2**2 + W1**2
ENDIF
```

```
IF ( DOG ) THEN
   G(1) = -R400*W2*X1 - TW0*W1
   G(2) =  R200*W2
ENDIF
CASE = 0
RETURN
END
```

3.7 Return Codes

There are two new values which may be returned in STATUS. A value of 7 indicates that the minimization failed because it was not possible to evaluate the function and/or gradient at the initial point. A value of 6 is returned when the function evaluation routine has requested an abort of the minimization process, as in Section 2.=rtc.

It should also be noted that a return code of STATUS = 4 is likely to eventually result from errors in the coding of the evaluation of f and/or g, as described in Section 2.=ls.

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. In this case, note the following. If the single precision version of BBVSCG is being called, then FUNCTN will be called with the EXTRA storage passed as the single precision array RW, whereas, when the double precision version of BBVSCG is used, the EXTRA storage will be in the double precision array DW.

For really complex problems, it will likely be more appropriate to explicitly use the arrays IW, RW and DW discussed in Section 2.=irdw. In this case, one must call BBLNIR directly.

Note that there is a new option for the minimization algorithm, namely to use Nocedal's update formula [noc] for the quasi-Newton matrices, as in Section 2.=upd. In this case, each update requires only $2n+1$ storage locations (rather that $2n+2$). This will have a small effect the computation of m, which now becomes $m = (\mathtt{LWORK}-3n)/(2n+1)$.

3.9 Subroutines Used

There are some additional routines, partly to implement some new features, and partly just for better organization. Note that the calling sequences of several internal routines have changed. By section, the new routines are:

(a) BBVALS [BBSVAL,BBRVAL] This is used to maintain the default values used by the package. They may be changed, as in Section 2.=cip.
BBNOCE [BBSNOC] This implements the updating strategy of Nocedal [noc].
BBCUBC [BBSCUB] This implements the cubic interpolation in Lemarechal [lem].

(b) ZZINNR This routine computes the default Euclidean inner product of vectors, with a call to ZZNRM2 when needed to compute the norm of a vector. This routine may be replaced, as in Section 2.=inn.
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, ZZTIME These machine dependent routines are necessary only for use with the test routine TESTBB of Section 3.17.

(d) ZZLENG, ZZSHFT, ZZDTTM These are also necessary only for use with the test routine TESTBB of Section 3.17.

3.10 Internal Structure

The structure has in fact only changed a little. The call to the routine BBVSCG has been simplied a bit, as in Section 2.=vscg. The method for changing internal parameters has been simplified, as in Section 2.=cip. However, the call to the inner routine BBLNIR is a bit longer.

To illustrate, consider the optimization of a function NTROPY for maximizing a certain entropy. It will look something like this:

```
INTEGER      N                          Problem dimension
PARAMETER ( N = 10000 )

INTEGER      WORKSZ                     Amount of working storage
PARAMETER ( WORKSZ = 37000 )

EXTERNAL           ZZINNR, NTROPY       External functions

INTEGER            INTS(13)             Arrays for resetting parameters
LOGICAL            LOGS(33)
DOUBLE PRECISION REALS(2), ZZINNR

DOUBLE PRECISION  F, X(N), G(N)         Function, point, gradient
DOUBLE PRECISION  WORK(WORKSZ)          Working storage

INTEGER  IW(10)              The problem specific communication arrays
REAL     RW(1)
DOUBLE PRECISION DW(4*N)

INTEGER    ID, IX, IG, IH, HDIM      Miscellaneous variables
INTEGER    FREQ, MAXF, CASE, STATUS
DOUBLE PRECISION ACCT, DF1

CALL BBVALS ( INTS, LOGS, REALS )   Get current values
LOGS(26) = .FALSE.                  Change to absolute tests
LOGS(27) = .FALSE.
CALL BBSVAL ( INTS, LOGS, REALS )   Reset values

FREQ = -10                          Every 10th point; no vectors
MAXF = 100                          Limit on function evaluations
CALL BBDFLT ( FREQ, MAXF )          Initialize; set all default values

CALL BBLRDF ( 1, 0, -1, -2, -3, -4 )  Set to use the Modulopt
CALL BBLFDF ( 2, 3, 4, 1 )            communication parameters.

ID = 1                              Indices to break up work array
IX = ID + N
IG = IX + N
IH = IG + N

HDIM = WORKSZ - 3*N                 Remaining working storage

DF1    =    Something appropriate; estimate of expected function reduction
```

```
... Code to set up  IW,  RW and  DW according to the problem
ACCT  = 1.E-5                          Desired accuracy
... Code to initialize  X

CASE = 2
CALL NTROPY ( CASE, N, X, F, G, IW, RW, DW )  Do initial evaluation
STATUS = -1                            Set to indicate f(x0), g(x0) done

CALL BBLNIR (NTROPY, N, X, F, DF1, G, ACCT, STATUS,  ZZINNR,
-          WORK(ID), WORK(IX), WORK(IG), WORK(IH), HDIM, IW, RW, DW)
... and the remainder of the code ...
```

3.11 Changing Defaults

This has been described in Section 2.=cip, at least with respect to changing values dynamically. It requires simply calling BBVALS and BBSVAL. 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 format of the output has been changed to be a bit more compact. One can also retain printing of the gradient while turning off printing of the point by setting the new argument POINT to .FALSE. (see Section 2.=cip). This is implemented by calling the entry point ZZP1ST(UNIT,GRAD,POINT,PFREQ). Note that the call to ZZPRNT has changed in that the last argument is now an integer rather than a logical. Finally, it is also possible to produce output on a second unit, independently of the first. This is done by calling an identical entry point ZZP2ST(UNIT2, GRAD2,POINT2,PFREQ2). Each call to ZZPRNT will generate output on either or both units, as required.

3.13 Function/Gradient Evaluation

First, the call to the evaluation routine has changed, as described in Section 3.6. The default and most common case of analytic derivative calculation is now obtained automatically, and DERV is no longer in the calling sequence of BBVSCG or BBLNIR. It may be changed as illustrated in Section 2.=cip. There is a fourth option: if DERV = 4, then the option DERV = 3 is implemented, but only on the first call to ZZEVAL.

Note that the count of function evaluations done is now returned as an argument in BBVSCG. Finally, it is possible now to trace the computations done when testing derivative calculations. This trace may be turned on as explained in Section 2.=cip; it is accomplished by calling the revised entry point ZZESET(TRF,TRG,TRTEST,UNIT) in ZZEVAL.

3.14 Termination Control

Section 2.=term outlines the changes to the method of terminating a minimization. Note that 4 termination criteria are now available, each of which may be applied independently. The call to the entry point has changed accordingly, and is now given as ZZTSET(NORM,TESTS,TRACE,TRU). Here TESTS is now a string of 4 characters, each

of which is T or F to control one of the 4 traces TGRAD, TSTEP, TSHXG, or TFUNC. The default of 'FTTF' is effectively the same as for Algorithm 630.

As well, it is now possible 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. This is in fact now the default.

It is also possible to trace the application of the termination criteria, as in Section 2.=trc. This is accomplished by setting TRACE to .TRUE., with TRU the unit where the trace output should appear.

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.=irdw. 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 re-called with STATUS = 3, as explained in Section 2.=stat.

3.16 Control Parameters for BBLNIR

These are still available, and may be changed if desired. Note that there are some further controls available, and the method for changing them is simpler, as in Section 2.=cip, where there is a complete list.

3.17 Test Program and Storage Requirements

A sample program is distributed with BBVSCG; as described in [bbl], this may be used to verify that BBVSCG is installed and running correctly on a particular system. Some changes have been made to this program; it now requires some further subroutines, as in Section 3.9, but all are of course provided. The changes allow selective testing of problems and they permit you to change the settings described in Section 2.=cip.

It now asks first if you wish to change any settings, and then which problems you wish to test. For this, it writes prompts on unit 6, assuming that that is the terminal, and it reads a response from unit 5, again normally the terminal. All output generated by the test goes to unit 6.

To conduct the normal test of all cases with all functions, the input required is

(a) a blank line, when asked if a trace is desired;
(b) a line with just a slash, followed by another exactly the same;
(c) -3, when asked what functions to test.

Thus the basic test will look like:

```
ENTER STRING OF T, F OR BLANK CHARACTERS TO DEFINE UP TO 33 LOGICAL VALUES:
                                  Reply with blank line
ENTER FREE FORMAT LIST OF UP TO 13 INTEGER VALUES:
/                                 ... the reply.
ENTER FREE FORMAT LIST OF UP TO 2 REAL VALUES:
/                                 ... the reply.
BEGINNING RUN #1:  CALL BBVSCG, ANALYTIC MODE, FORWARD CALLS.
  CONTROL: 0 QUIT, -1 SKIP TO NEXT SET, -2 FINISH THIS SET
           -3 FINISH FULL RUN, >0 DO PROBLEM #
-3                                ... the reply.
```

If the assignment of the units is inconvenient, it may easily be changed; see Section 3.3. The unit where all the test output goes is defined in BBVALS; this may be changed as described in Section 2.=cip.

Acknowledgements

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.

Bibliography

[bbl] BUCKLEY, A. ALGORITHM 630: BBVSCG—A variable-storage algorithm for function minimization. *ACM Trans. on Math. Soft. 11, 2* (1985), 103-119.

[lem] LEMARÉCHAL, C. A view of line searches. In *Optimization and Optimal Control.* A. Auslender, W. Oettli and J. Stoer, eds. Springer Verlag, Heidelberg, 1981.

[mopt] LEMARÉCHAL, C. Using a Modulopt optimization code. I.N.R.I.A., Rocquencourt, 78153 Le Chesnay Cedex, France

[noc] NOCEDAL, J. Updating quasi-Newton matrices with limited storage. *Maths of Comp. 35, 151* (1980), 773–782.

[vscg] BUCKLEY, A. AND LENIR, A. QN-like variable-storage conjugate gradients. *Math Prog. 27* (1983), 155-175.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique