Third-order derivatives of the Moré, Garbow, and Hillstrom test set problems*

E. G. Birgin† J. L. Gardenghi† J. M. Martínez‡ S. A. Santos‡

April 1, 2018.

Abstract

The development of Fortran routines for computing third-order derivatives of the problems proposed by J. J. Moré, B. S. Garbow, and K. E. Hillstrom (*ACM Trans. Math. Softw.* 7, 14–41, 136–140, 1981) is reported in this work. For this, we have prepared a Fortran module for computing the function values, besides first-, second- and third-order derivatives for all the 35 problems presented by the authors of the original paper. We also provide the routines for unconstrained optimization including third-order derivatives. This will allow novel analysis on optimization for algorithms that possibly use third-order derivatives of the objective function.

Key words: Optimization test problems, third-order derivatives subroutines.

1 Introduction

The interest in high-order optimization methods re-emerged in the last few years. In [5] the authors proposed a high-order regularization method for unconstrained minimization that uses derivatives up to order $p \geq 1$ of the objective function (see also [7]). A trust-region algorithm that applies to convexly-constrained problems and uses derivatives up to order $p \geq 1$ was introduced in [6]. A method that uses high-order derivatives to escape from saddle point in non-convex optimization was considered in [1]. This state of facts encouraged us to implement third-order derivatives of the classical Moré, Garbow, and Hillstrom test set [8]. In particular, we were interested in implementing and testing a practical version [4] of the method proposed in [5] that makes use of third-order derivatives. Making third-order derivatives available also opens doors to implement and test other methods that may appear in literature.

In 1981, Moré, Garbow, and Hillstrom [8] proposed a well-known test set with 35 problems. Each problem is defined by $m > 0$ functions $f_1, f_2, \ldots, f_m$ such that $f_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, m$.

*This work has been partially supported by FAPESP (grants 2013/03447-6, 2013/05475-7, 2013/07375-0, 2013/23494-9, 2016/01860-1, and 2017/03504-0) and CNPq (grants 309517/2014-1, 303750/2014-6, and 302915/2016-8).

†Department of Computer Science, Institute of Mathematics and Statistics, University of São Paulo, Rua do Matão, 1010, Cidade Universitária, 05508-090, São Paulo, SP, Brazil. e-mail: {egbirgin | john}@ime.usp.br

‡Department of Applied Mathematics, Institute of Mathematics, Statistics, and Scientific Computing, University of Campinas, Campinas, SP, Brazil. e-mail: {martinez | sandra}@ime.unicamp.br

Problems are divided into three categories: (a) 14 *systems of nonlinear equations*, cases where $m = n$ and one searches for x^* such that $f_i(x^*) = 0$, $i = 1, 2, \ldots, m$; (b) 18 *nonlinear least-squares* problems, cases where $m \geq n$ and one is interested in solving the problem

$$\underset{x \in \mathbb{R}^n}{\text{Minimize}} \, f(x) = \sum_{i=1}^{m} f_i^2(x) \tag{1}$$

by exploring its particular structure; and (c) 18 *unconstrained minimization* problems, where one is interested in solving (1) just by applying a general unconstrained optimization solver.

For each problem, the original package [9] provided Fortran 77 code to compute the objective function and its first-order derivatives. In 1994, Averbukh, Figueroa, and Schlick [2, 3] made available Fortran 77 code for computing second-order derivatives of the 18 problems in the *unconstrained minimization* category. Although the problems are divided into three different categories, one can view the whole set of 35 problems as unconstrained minimization problems by considering the form (1). For this reason, in this work, we provide (a) second-order derivatives for the 17 problems that were not considered in [2, 3] and (b) third-order derivatives for all the 35 problems from the test set. The code for computing the objective functions, as well as their first-, second-, and third-order derivatives, was implemented using modern Fortran. Details about the implementation and the package usage are given in the next section.

2 Package subroutines

The present section briefly describes a modern Fortran package with routines that compute the objective function, first-, second-, and third-order derivatives of the 35 problems firstly coded in [9]. Table 1 displays, for each problem, its name, the default values for n and m adopted in this work, the restrictions on n and m for problems with variable dimensions, and, for compatibility with the subroutines for the *unconstrained minimization* category provided in [9], the number of the problem within this category.

The package includes three modules `mgh_sp`, `mgh_dp`, and `mgh_qp` that contain the subroutines described below in single, double, and quad precisions, respectively. Those should be the adopted precisions for the `real` parameters of the subroutines. The subroutines included in the package have the following prototypes:

```
subroutine mgh_set_problem(nprob,flag),
subroutine mgh_set_dims(n,m,flag),
subroutine mgh_get_dims(n,m),
subroutine mgh_get_x0(x0,factor),
subroutine mgh_get_name(name),
subroutine mgh_evalf(x,f,flag),
subroutine mgh_evalg(x,g,flag),
subroutine mgh_evalh(x,h,flag),
subroutine mgh_evalt(x,t,flag).
```

Table 1: Description of the 35 problems from [8].

Problem number and name	Default dimensions		Dimensions' restrictions		Number of the problem within the unconstrained minimization category
	n	m	n	m	
1 Rosenbrock	2	2	2	2	–
2 Freudstein and Roth	2	2	2	2	–
3 Powell badly scaled	2	2	2	2	4
4 Brown badly scaled	2	3	2	3	10
5 Beale	2	3	2	3	16
6 Jennrich and Sampson	2	10	2	$\geq n$	–
7 Helical valley	3	3	3	3	1
8 Bard	3	15	3	15	–
9 Gaussian	3	15	3	15	3
10 Meyer	3	16	3	16	–
11 Gulf research and development	3	99	3	$[n, 100]$	12
12 Box three-dimensional	3	10	3	$\geq n$	5
13 Powell singular	4	4	4	4	–
14 Wood	4	6	4	6	17
15 Kowalik and Osborne	4	11	4	11	–
16 Brown and Dennis	4	20	4	$\geq n$	11
17 Osborne 1	5	33	5	33	–
18 Biggs EXP6	6	13	6	$\geq n$	2
19 Osborne 2	11	65	11	65	–
20 Watson	6	31	$[2, 31]$	31	7
21 Extended Rosenbrock	10	10	even	n	14
22 Extended Powell singular	12	12	multiple of 4	n	15
23 Penalty I	4	5	variable	$n + 1$	8
24 Penalty II	4	8	variable	$2n$	9
25 Variably dimensioned	10	12	variable	$n + 2$	6
26 Trigonometric	10	10	variable	n	13
27 Brown almost-linear	40	40	variable	n	–
28 Discrete boundary value	10	10	variable	n	–
29 Discrete integral equation	10	10	variable	n	–
30 Broyden tridiagonal	10	10	variable	n	–
31 Broyden banded	10	10	variable	n	–
32 Linear - full rank	10	10	variable	$\geq n$	–
33 Linear - rank 1	10	10	variable	$\geq n$	–
34 Linear – rank 1 with zero columns and rows	10	10	variable	$\geq n$	–
35 Chebyquad	8	8	variable	$\geq n$	18

As its name indicates, subroutine `mgh_set_problem` must be called in first place to set the problem to be solved. In this subroutine, `nprob` is an input integer parameter that corresponds to the problem number; while `flag` is an output integer parameter that is set to a non-null value if `nprob` is not a valid problem number (between 1 and 35). When `mgh_set_problem` is called, dimensions n and m are set with their default values (listed in Table 1.) Subroutine `mgh_get_dims`, that has `n` and `m` as integer output parameters, can be used to obtain the values of n and m. Both parameters are optional. Subroutine `mgh_set_dims` can be used to modify the values of n and/or m. In this subroutine, `n` and `m` are optional integer input parameters; while `flag` is an integer output parameter that is set to a non-null value if the values of `n` and/or `m` do not obey the restrictions listed in Table 1. If, on return, `flag` is non-null then the values of n and m remain unaltered. Subroutine `mgh_get_name` has `name` as a character(len=60) output parameter and, as its name indicates, returns the problem name as listed in Table 1. Subroutine `mgh_get_x0` returns the initial point, scaled by `factor`. In this subroutine, `x0` is an output real n-dimensional array parameter; while `factor` is an optional real input parameter. Subroutines `mgh_evalf`, `mgh_evalg`, `mgh_evalh`, and `mgh_evalt` compute the objective function and its first-, second-, and third-order derivatives, respectively. In the four subroutines, `x` is an input real n-dimensional array and `flag` is an integer output parameter that is set to a non-null value if some failure occurred with the computation. Moreover, `f`, `g`, `h`, and `t` are all real output parameters such that `f` is a scalar, `g` is an n-dimensional array, `h` is an $(n \times n)$-dimensional array, and `t` is an $(n \times n \times n)$-dimensional array. On output, `f` contains $f(x)$, `g` contains $\nabla f(x)$, `h` contains the upper triangle of the Hessian $H = \nabla^2 f(x)$ (i.e. elements $H(i,j)$ with $i \leq j$), and `t` contains the upper portion of the third-order derivative $T = \nabla^3 f(x)$ (i.e. elements $T(i,j,k)$ such that $i \leq j \leq k$), respectively. The remaining entrances of `h` and `t` remain unaltered.

In addition to the subroutines described above, aiming to preserve compatibility with the traditional routines of Algorithm 566, we provide a wrapper that implements the routines

```
subroutine initpt(n,x,nprob,factor),
subroutine objfcn(n,x,f,nprob),
subroutine grdfcn(n,x,g,nprob),
subroutine hesfcn(n,x,hesd,hesl,nprob),
subroutine trdfcn(n,x,td,tl,nprob).
```

Subroutines `initpt`, `objfcn`, and `grdfcn` were first implemented in [9]. For a given problem `nprob` in the *unconstrained minimization* category (with `nprob` between 1 and 18 and according to the last column of Table 1), they return the initial point scaled by `factor`, the objective function, and its gradient, respectively. Subroutine `hesfcn`, first implemented in [2, 3], computes the Hessian matrix of the objective function. In the present work, we introduce the subroutine `trdfcn` that computes the third-order derivative tensor. When using these routines, the value of n may be the default value or any value satisfying the constraints, as displayed in Table 1. If n takes its default value, m takes its default value as well; otherwise, m is set satisfying the constraints in the table.

In subroutine `initpt`, `n` and `nprob` are input integer parameters; while `factor` is an input real scalar and `x` is an output real n-dimensional array. On output, `x` contains the initial point scaled by `factor`. In subroutines `objfcn`, `gradfcn`, `hesfcn`, and `trdfcn`, `n`, `x`, and `nprob` are input parameters, `n` and `nprob` being integers, and `x` being a real n-dimensional array. On the

other hand, `f`, `g`, `hesd`, `hesl`, `td`, and `tl` are all real output parameters. `objfcn` returns $f(x)$ in the scalar `f`. `gradfcn` returns $\nabla f(x)$ in the n-dimensional array `g`. In `hesfcn`, `hesd` is an n-dimensional array; while `hesl` is an n_h-dimensional array with $n_h = n(n-1)/2$. On output, `hesd` contains the diagonal of the Hessian H; while `hesl` contains the upper triangle of H in column-major order. This means that for any $i < j$, element $H(i,j)$ is located at position $(j-1)(j-2)/2+i$ of `hesl`. In `trdfcn`, `td` is an n-dimensional array and `tl` is an n_t-dimensional array with

$$n_t = \frac{(n-1)}{6}\Big[(n-2)(n-3) + 9(n-2) + 12\Big].$$

On output, `td` contains the diagonal of the third-order derivative T, i.e. elements $T(i,j,k)$ with $i = j = k$; while `tl` containts the upper triangle of T (i.e. elements $T(i,j,k)$ with $i \leq j \leq k$ and not all identical) in column-major order. This means that for any triplet (i,j,k) with $i \leq j \leq k$, not all identical, element $T(i,j,k)$ is located at position p of `td` with

$$p = \frac{k-2}{6}\Big[(k-3)(k-4) + 9(k-3) + 12\Big] + \frac{j(j-1)}{2} + i.$$

3 Driver

We follow Averbukh, Figueroa, and Schlick [3] and provide a driver program in which we test the derivatives up to third order of the 35 problems using a Taylor expansion of f around a point $x_c \in \mathbb{R}^n$. For this, given $\epsilon > 0$ and y a random vector in $\mathbb{R}^n$, we consider the Taylor expansion

$$f(x_c + \epsilon y) = f(x_c) + \sum_{j=1}^{3} \frac{1}{j!} P_j(x_c, \epsilon y) + R(x_c + \epsilon y),$$

where, for $x, s \in \mathbb{R}^n$, $P_j(x,s)$ is the homogeneous polynomial of degree j defined by

$$P_j(x,s) = \left(s_1 \frac{\partial}{\partial x_1} + \ldots + s_n \frac{\partial}{\partial x_n}\right)^j f(x)$$

and the remainder term $R(x_c + \epsilon y)$ is $O(\epsilon^4)$. The test consists in computing the Taylor approximation

$$f(x_c + \epsilon_k y) = f(x_c) + \sum_{j=1}^{3} \frac{1}{j!} P_j(x_c, \epsilon_k y)$$

for $\epsilon_0 = 1/2$ and $\epsilon_{k+1} = \epsilon_k/2$ for $k = 1, 2, \ldots$. If all the derivatives are correct, the ratio

$$\frac{R(x_c + \epsilon_k y)}{R(x_c + \epsilon_{k+1} y)} = \frac{O(\epsilon_k^4)}{\frac{1}{16} O(\epsilon_k^4)} = 16 \tag{2}$$

should be observable. In practice, if $R(x_c + \epsilon_k y)$ is relatively small, when k goes to infinity, the ratio (2) (a) tends to 2 if no derivatives are correct, (b) tends to 4 if only first-order derivatives are correct, (c) tends to 8 if only first- and second-order derivatives are correct, and (d) tends to 16 if all derivatives are correct. Other cases may include convergence to unity when the error is too large when compared to y.

References

[1] A. Anandkumar and R. Ge. Efficient approaches for escaping high-order saddle points in nonconvex optimization. 2016. Avaliable online at arXiv:1602.05908.

[2] V. Z. Averbukh, S. Figueroa, and T. Schlick. HESFCN - A FORTRAN package of Hessian subroutines for testing unconstrained optimization software. Technical report, Courant Institute of Mathematical Sciences, New York University, 1992.

[3] V. Z. Averbukh, S. Figueroa, and T. Schlick. Remark on algorithm 566. *ACM Transactions on Mathematical Software*, 20(3):282–285, 1994.

[4] E. G. Bigin, J. L. Gardenghi, J. M. Martínez, and S. A. Santos. On the use of third-order models with fourth-order regularization for unconstrained optimization. Submitted.

[5] E. G. Birgin, J. L. Gardenghi, J. M. Martínez, S. A. Santos, and Ph. L. Toint. Worst-case evaluation complexity for unconstrained nonlinear optimization using high-order regularized models. *Mathematical Programming*, 163(1):359–368, 2017.

[6] C. Cartis, N. I. M. Gould, and Ph. L. Toint. Second-order optimality and beyond: Characterization and evaluation complexity in convexly constrained nonlinear optimization. *Foundations of Computational Mathematics*. To appear (DOI: 10.1007/s10208-017-9363-y).

[7] C. Cartis, N. I. M. Gould, and Ph. L. Toint. Improved second-order evaluation complexity for unconstrained nonlinear optimization using high-order regularized models. 2017. Avaliable online at arXiv:1708.04044.

[8] J. J. Moré, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, 1981.

[9] J. J. Moré, Burton S. Garbow, and Kenneth E. Hillstrom. Algorithm 566: FORTRAN Subroutines for Testing Unconstrained Optimization Software. *ACM Transactions on Mathematical Software*, 7(1):136–140, 1981.