

Fortran routines for testing unconstrained optimization software with derivatives up to third-order*

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

April 20, 2018

Contents

1 Introduction

This document gives details about the implementation and usage of a Fortran package that implements the computation of objective function and its first-, second-, and third-order derivatives for the well-known 35 problems proposed by Moré, Garbow and Hillstom [?, ?].

Originally, Moré, Garbow, and Hillstom proposed 35 test problems for unconstrained optimization and code for computing the objective function and its first-derivative. The problems were divided into three categories: (a) 14 *systems of nonlinear equations*, cases where $m = n$ and one searches for x^* such that $f_i(x^*) = 0$, $i = 1, 2, \dots, m$; (b) 18 *nonlinear least-squares* problems, cases where $m \geq n$ and one is interested in solving the problem

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

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

In 1994, Averbukh, Figueroa, and Schlick [?] added code to compute the second-order derivative for the 18 unconstrained minimization problems.

We now propose a package that considers (??) for all the 35 test problems and implements its first-, second-, and third-order derivatives.

*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

2 Getting Started

When unzipping the code, the user must get the following directories and files:

```
$(MGH) ..... root directory.
├─ Makefile
├─ mgh_doc.pdf ..... documentation PDF file.
├─ driver1.f08 ..... driver with new routines.
├─ driver2.f08 ..... driver with alg 566 routines.
├─ mgh.f08 ..... all the new routines.
├─ mgh_wrapper.f08 ..... wrapper with alg 566 routines.
└─ set_precision.f08 ..... precision definitions file.
```

After compiling the code, the user must get the binaries `driver1` and `driver2` and the object files inside `$(MGH)`.

3 Compiling the code

To compile the main code,

1. The user must have a Fortran compiler installed and must configure in `$(MGH)/Makefile` the variables `FC` with the Fortran compiler command-line and `FFLAGS` with the desired flags for the chosen Fortran compiler. We tested `gfortran` and `nagfor` compilers. Other Fortran compilers were not tested, but they may work as well.
2. Using the terminal, type `make` in the root directory.

To clean the compiled code, use `make clean`.

4 Using the module and compiling code

To use the module, the user must make the following modifications in the code.

1. Choose the precision you want to use in module `$(MGH)/set_precision.f08`. For this, set the parameter `rk` as `kind_s` for single-precision, `kind_d` for double-precision, and `kind_q` for quad-precision.
2. Implement the desired routines (see Section ??).
3. Compile your code

```
$(FC) -I $(MGH) -o your_bin your_code.f08
                                $(MGH)/mgh.o
                                $(MGH)/set_precision.o
```

replacing `$(FC)` by the Fortran compiler you are using. You may need to adjust command-line option `-I`, that stands for the directory where `.mod` files are.

If the user prefer, it is possible to use the classic Algorithm 566 routines, to which we added a new one to compute third-order derivatives. In this case,

1. Choose the precision you want to use in module `$(MGH)/set_precision.f08`. For this, set the parameter `rk` as `kind_s` for single-precision, `kind_d` for double-precision, and `kind_q` for quad-precision.
2. Implement the desired routines in the code (see Section ??).
3. Compile your code

```
$(FC) -I $(MGH) -o your_bin you_code.f08
                                $(MGH)/mgh.o
                                $(MGH)/mgh_wrapper.o
                                $(MGH)/set_precision.o
```

replacing `$(FC)` by the Fortran compiler you are using. You may need to adjust command-line option `-I`, that stands for the directory where `.mod` files are.

5 Using the drivers

Two drivers are available to test and validate the code:

1. `$(MGH)/driver1` implements the new routines module. It runs all the 35 problems from the test set. The output is given in `driver1.out` file.
2. `$(MGH)/driver2` implements the algorithm 566 routines. It runs all the 18 unconstrained minimization problems (see [?]). The output is given in `driver2.out` file.

6 Routines description

6.1 New routines

In order to use the new routines, first of all the user must

1. include the module in the code using `use mgh`.

2. set the number of problem to work with, between 1 and 35, using `mg_h_set_problem`,
3. customize `m` and `n` using `mg_h_set_dims` or retrieve default values using `mg_h_get_dims`,

After that, the user is able to retrieve the initial point using `mg_h_get_x0` and compute the objective function and its first-, second-, and third-order derivatives using `mg_h_evalf`, `mg_h_evalg`, `mg_h_evalh`, and `mg_h_evalt`, respectively. A detailed description of each routine follows.

subroutine `mg_h_set_problem(user_problem, flag)`

Sets the problem number. When the user set the problem number, default dimensions (`n` and `m`) for it are automatically set. The subroutines arguments are

- `user_problem` is an input integer argument that should contain the problem number between 1 and 35.
- `flag` is an output integer argument that contains 0 if the problem number was successfully set or -1 if the `user_problem` is out of the range.

subroutine `mg_h_set_dims(n, m, flag)`

Sets the dimensions for the problem.

- `n` is an input optional integer argument, sets the number of variables for the problem set.
- `m` is an input optional integer argument, sets the number of equations for the problem set.
- `flag` is an output optional integer, in the return contains 0 if the dimensions were set successfully, -1 if `n` is not valid, -2 if `m` is not valid, or -3 if both are not valid.

subroutine `mg_h_get_dims(n, m)`

Gets the dimension for the problem.

- `n` is an output optional integer argument with the number of variables for the problem.
- `m` is an output optional integer argument with the number of equations for the problem.

subroutine `mg_h_get_x0(x0, factor)`

Gets the initial point for the problem.

- `x0` is an output array of length `n` that contains the initial point.
- `factor` is an optional real scalar that scales the initial point returned at `x0`.

subroutine mgh_evalf(x, f, flag)

Computes the objective function evaluated at **x**.

- x** is an input real array of length **n**, contains the point in which the objective function must be evaluated.
- f** is an output real that contains the objective function evaluated at **x**.
- flag** is an output integer that contains 0 is the computation was made successfully, -1 if problem is not between 1 and 35, or -3 if a division by zero was made.

subroutine mgh_evalg(x, g, flag)

Computes the gradient of the objective function evaluated at **x**.

- x** is an input real array of length **n**, contains the point in which the gradient must be evaluated.
- g** is an output real array of length **n** that contains the gradient evaluated at **x**.
- flag** is an output integer that contains 0 is the computation was made successfully, -1 if problem is not between 1 and 35, or -3 if a division by zero was made.

subroutine mgh_evalh(x, h, flag)

Computes the Hessian of the objective function evaluated at **x**.

- x** is an input real array of length **n**, contains the point in which the Hessian must be evaluated.
- h** is an output real array of length $n \times n$ that contains the upper triangle of the Hessian evaluated at **x**.
- flag** is an output integer that contains 0 is the computation was made successfully, -1 if problem is not between 1 and 35, or -3 if a division by zero was made.

subroutine mgh_evalt(x, t, flag)

Computes the third-order derivative tensor of the objective function evaluated at **x**.

- x** is an input real array of length **n**, contains the point in which the third-order derivative must be evaluated.
- t** is an output real array of length $n \times n \times n$ that contains the upper part of the third-derivative evaluated at **x**.
- flag** is an output integer that contains 0 is the computation was made successfully, -1 if problem is not between 1 and 35, or -3 if a division by zero was made.

subroutine mgh_get_name(name)

Returns the problem name

name is a `character(len=60)` output parameter that contains the problem name.

6.2 Algorithm 566 Routines + Third derivative computation

subroutine `initpt(n, x, nprob, factor)`

Returns the initial point for a given problem.

n is an integer input argument, should contain the dimension of the problem.

x is a real output array of length **n**, contains the initial point.

nprob is an integer input, contains the number of the problem between 1 and 18.

factor is a real input, contains the factor by which the initial point will be scaled.

subroutine `objfcn(n, x, f, nprob)`

Compute the objective function value for a given problem at **x**.

n is an integer input argument, should contain the dimension of the problem.

x is a real input array of length **n**, contains the point in which the objective function will be evaluated.

f is a real output argument that contains the objective function value.

nprob is an integer input, contains the number of the problem between 1 and 18.

subroutine `grdfcn(n, x, g, nprob)`

Compute the gradient of the objective function, for a given problem, evaluated at **x**.

n is an integer input argument, should contain the dimension of the problem.

x is a real input array of length **n**, contains the point in which the objective function will be evaluated.

g is a real output array of length **n**, contains the gradient of the objective function value evaluated at **x**.

nprob is an integer input, contains the number of the problem between 1 and 18.

subroutine `hesfcn(n, x, hesd, hesu, nprob)`

Compute the Hessian of the objective function, for a given problem, evaluated at **x**.

n is an integer input argument, should contain the dimension of the problem.

x is a real input array of length **n**, contains the point in which the objective function will be evaluated.

hesd is a real output array of length **n**, contains the diagonal of the Hessian.

hesu is a real output array of length

$$\frac{n(n-1)}{2},$$

contains the strict upper triangle of the Hessian stored by columns. The i, j term of the Hessian, $i < j$, is located at the position

$$\frac{(j-1)(j-2)}{2} + i$$

at **hesu**.

nprob is an integer input, contains the number of the problem between 1 and 18.

subroutine trdfcn(n, x, td, tu, nprob)

Compute the third-order derivative tensor of the objective function, for a given problem, evaluated at **x**.

n is an integer input argument, should contain the dimension of the problem.

x is a real input array of length **n**, contains the point in which the objective function will be evaluated.

td is a real output array of length **n**, contains the diagonal of the tensor.

tu is a real output array of length

$$\frac{n-1}{6}((n-2)(n-3) + 9(n-2) + 12),$$

contains the strict upper part of the tensor stored by columns. The i, j, k term of the tensor, $i \leq j \leq k$ but not $i = j = k$, is located at the position

$$\frac{k-2}{6}((k-3)(k-4) + 9(k-3) + 12) + \frac{j(j-1)}{2} + i$$

at **tu**.

nprob is an integer input, contains the number of the problem between 1 and 18.

References

- [1] V. Z. Averbukh, S. Figueroa, and T. Schlick, Remark on Algorithm 566, *ACM Transactions on Mathematical Software*, 20(3):282–285, 1994. DOI 10.1145/192115.192128.
- [2] J. J. Moré, B. S. Garbow, and K. E. Hillstom, Algorithm 566: FORTRAN Subroutines for Testing Unconstrained Optimization Software, *ACM Transactions on Mathematical Software*, 7(1):136–140, 1981. DOI 10.1145/355934.355943.

- [3] J. J. Moré, B. S. Garbow, and K. E. Hillstom, Testing Unconstrained Optimization Software, *ACM Transactions on Mathematical Software*, 7(1):17–41, 1981. DOI 10.1145/355934.355936.