TRDF Tutorial
A trust-region derivative-free algorithm for constrained optimization

May 27, 2015

This text describes how to use the implemented algorithm TRDF proposed in [1]. The algorithm solves the problem

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & x \in \Omega \end{array} \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $\Omega = \{x \in \mathbb{R}^n \mid h(x) = 0, g(x) \leq 0\}$ where $h : \mathbb{R}^n \to \mathbb{R}^p$ and $g : \mathbb{R}^n \to \mathbb{R}^q$ are differentiable functions. Although the algorithm can be applied when the objective function is nonsmooth, it was designed for the class of problems in which f is smooth but its derivatives are not available.

The algorithm is implemented in `Fortran` and is available at this page[1].

In Section 1, we show how to compile and run a simple problem for testing TRDF. Section 2 is devoted to explaining how to reproduce the numerical experiments with the Hock-Schittkowski test problems. In sections 3 and 4 we explain how the user can write his/her own problems and how to call the TRDF subroutine using the correct parameters. Section 5 explains the inner parameters defined in `tr_params.par`. Finally, in Section 6 we describe how an advanced user can write an interface and change the inner solver used by TRDF.

1 Compiling and running

The present implementation of the TRDF algorithm contains an example problem (problem 37 from the Hock & Schittkowski [4] test set). The file `trdf_main.f` shows how to implement the necessary subroutines, define the problem and call TRDF.

[1] http://people.ufpr.br/~ewkaras/pesquisa/publicacoes/supplemental_conejo_karas_pedroso.html

By default, the TRDF implementation uses the nonlinear programming algorithm ALGENCAN [2, 3] for solving the constrained trust-region subproblems. Two versions of ALGENCAN are supported: 2.2.1 and 3.0.0 (default). In order to use such versions, the user has to download ALGENCAN and create the library `libalgencan.a`. Version 3.0.0 already has a natural way to create it, by simply typing `make` inside the top-level directory of ALGENCAN. The library will be created at subdirectory `lib`. See Section 6 for more information on how to write an interface to a different nonlinear programming solver.

After creating ALGENCAN's library, the user has to edit the `Makefile` and change the path to the solver' library in variable `SOLVERLIB`. Then, just type

```
make trdf
```

to build the executable.

If version 2.2.1 is used, the user must manually build the library, by entering in the (already downloaded) ALGENCAN directory and typing the following commands:

```
gfortran -xf77-cpp-input -c *.f
ar ruv libalgencan.a *.o
```

Then, the file `Makefile` should be edited to inform the path to the library in variable `SOLVERLIB and` variable `SOLVER` has to be changed to `algencan_old_solver`.

The user is able to suppress ALGENCAN's output by creating an empty file called `.silent` in the same directory where the executable will be run. This can be done by typing

```
touch .silent
```

in the terminal. To suppress TRDF's output, please see Section 5.

Additionally, the present implementation also contains an interface to the C language. Users familiar with the C language can write their problems and solve using the TRDF algorithm. An example of how to define a problem is implemented in file `trdf_main.c` (problem 37 of the Hock & Schittkowski [4] test set). The process of building the libraries is the same for the `Fortran` case, except that, the in the last step,

```
make c_trdf
```

should be typed.

Quick start

For quickly building the first example, under ALGENCAN 3.0.0, follow the steps below:

1. Download and unpack ALGENCAN 3.0.0 in [3];
2. In the root directory of ALGENCAN, build `libalgencan.a` by typing `make`. The file should be under `lib` subdirectory;
3. Download and unpack TRDF;
4. Under TRDF's directory, edit, in the file `Makefile`, the variable `SOLVERLIB` with the full path to the `lib` subdirectory of ALGENCAN;
5. Type `make trdf` ;
6. Suppress ALGENCAN's output by typing `touch .silent` ;
7. Run the TRDF example with `./trdf` .

2 Running the full Hock-Schittkowski test set

In order to reproduce the experiments of the supplemental material, the user should build the `hstests` executable. For this task, it is necessary first to build a library with the Hock-Schittkowski test problems. The updated (Oct, 2011) source file of the problems can be downloaded at

http://www.ai7.uni-bayreuth.de/tpnp08.htm

but we provide an older version with TRDF (with some personal corrections). The numerical experiments were performed with the provided old version.

To build this executable is sufficient to build ALGENCAN's library, as described in Section 1, and then follow the steps below:

1. Type `make hstests` ;
2. Run the executable by typing `./hstests` ;
3. Write the problem number and hit Enter.

An output file called `runhs.out` is created with the following values: the problem number, number of variables, number of inequality constraints, number of equality constraints, the best (known) value of the objective function, the objective function value found by TRDF, the sup-norm of the infeasibility (see Section 4), and the number of performed function evaluations.

We also provide a shell script `runhstests.sh` which runs all the test problems considered in the supplemental material. In order to use this script, build the `hstests` executable then type

```
./runhstests.sh hstests
```

For each test problem, at most 30 minutes of CPU time will be allowed. A file called `analysis-all` will be created at the end of the tests, where each column was explained in the former paragraph.

3 The user-defined problem

The user should define his/her own problem in a separate file. We provide an possible example of implementation in file `trdf_main.f`. This file contains the main program where all the input parameters are defined and the main TRDF subroutine is called. Each TRDF parameter is fully detailed in Section 4. In addition, 4 user subroutines can be used by the algorithm:

- `calobjf`: calculates the objective function
- `calcon`: calculates the i-th constraint
- `caljac`: calculates the sparse Jacobian of the i-th constraint
- `calhc`: calculates the sparse Hessian of the i-th constraint

We suppose that at least two subroutines **have** to be coded by the user: `calobjf` and `calcon` (if the problem has constraints). If the problem does not have constraints, the user should set $m = 0$ and create an empty subroutine.

Depending on the solver used for the subproblems (see Section 6) the user can provide two more subroutines, related with the first and second derivatives of the constraints. ALGENCAN [3], the solver supported by default, is able to use first and second order information. If subroutine `caljac` is provided, the the user should set `CCODED(1) = .true.` in the main program. Likewise, if subroutine `calhc` is provided, then `CCODED(2) = .true.` should be set in the main program.

If the user does not want to use first and second order information to solve his/her problem, both subroutines should still be present, should be empty, return flag -1 and their respective `CCODED` values should be set to `.false.`.

4 The `TRDF` subroutine

A full call to the TRDF subroutine is as follows:

```
 CALL TRDF(N,NPT,X,XL,XU,M,EQUATN,LINEAR,CCODED,
+          MAXFCNT,RBEG,REND,XEPS,
+          F,FEAS,FCNT)
```

Basically, the first line contains problem specific input parameters, the second line contains configuration parameters (which can be omitted by a call to the `EASYTRDF` subroutine) and the third line contains the output parameters. Variable `X` is an exception, since it is used as both input, for the starting point, and output, for the solution found by the algorithm. The `EASYTRDF` subroutine is an alias to the `TRDF` subroutine, where the parameters `NPT`, `MAXFCNT`, `RBEG`, `REND` and `XEPS` are set to their default values.

Each one of the arguments is described below.

- `N` (integer, input): the dimension of the problem
- `NPT` (integer, input): the number of points used for building the quadratic model of the objective function. Must be an integer in the range $[2N+1, (N+1)(N+2)/2]$
- `X(N)` (double precision, input/output): as input, contains the initial point used for the subroutine. If the provided initial point is infeasible, the phase 0 is applied, an attempt to project it onto the feasible set Ω by calling `SOLVER` with `PHASE0 = .true.`. As output, it is the solution found by the algorithm
- `XL(N)` and `XU(N)` (double precision, input): the lower and upper bounds on variable `X`, respectively
- `M` (integer, input): the number of constraints of the problem
- `EQUATN(M)` (logical, input): if `EQUATN(i)` is `.true.`, then constraint i is an equality constraint

- `LINEAR(M)` (logical, input): if `LINEAR(i)` is `.true.`, then constraint i is a linear constraint
- `CCODED(2)` (logical, input): indicates if the Jacobian (`CCODED(1) = .true.`) and the Hessian (`CCODED(2) = .true.`) of the constraints will be provided (see Section 3 for a description of the subroutines that should be provided by the user)
- `MAXFCNT` (integer, input): the maximum number of allowed function evaluations
- `RBEG` (double precision, input): the initial value of the trust-region radius
- `REND` (double precision, input): the final value of the trust-region radius, used for declaring convergence of the method
- `XEPS` (double precision, input): the tolerance used by the method to decide if a point is feasible or not
- `F` (double precision, output): the value of the objective function at the solution
- `FEAS` (double precision, output): the sup-norm of the infeasibility at the solution ($\max\{\|g(x^*)\|_\infty, \|h(x^*)\|_\infty\}$)
- `FCNT` (integer, output): the number of function evaluations used to find the solution

5 The file `tr_params.par`

The file `tr_params.par` contains internal parameters used by TRDF. They define the size of the arrays and also reduce the number of parameters passed to the TRDF subroutine. Below we describe each parameter in detail.

- `NMAX` is the maximum number of variables that are allowed. When solving problem with more than 1000 variables (or when working on a computer with very low CPU) it is necessary to change this value
- `MMAX` is the maximum number of constraints. The same argument used for `NMAX` is applied here
- `JCNNZMAX` is the maximum number of non null elements in the Jacobian of the constraints. The default value is set to be the full Jacobian (`NMAX` $\times$ `MMAX`)

- `HCNNZMAX` is the maximum number of non null elements in the Hessian of the objective function and of each constraint. The default value is set to be the case where all the elements of the Hessians are non null (`NMAX`$^2 \times$ (1+ `MMAX`))
- `MAXXEL` is the maximum number of elements of the vector `X` that are displayed in the output
- `INN` is a internal parameter that in most cases should have the same value of `NMAX`. So, if the user changes the value of `NMAX` it is recommended to also change this parameter
- `OUTPUT` is a logical parameter that enables the output of TRDF if set to `.true.` and disables it, if set to `.false.`

6 The `SOLVER` interface

At iteration k, $k = 1, \ldots,$ the algorithm needs to solve the following constrained nonlinear subproblem:

$$\begin{array}{ll} \text{minimize} & m_k(x) \\ \text{subject to} & x \in \Omega \cap B_k \end{array} \tag{2}$$

where m_k is the quadratic model for the objective function f at iteration k, Ω was defined in problem (1) and B_k is the trust-region at iteration k.

The implementation of the TRDF algorithm is independent on the solver used for the constrained trust-region subproblems. It is important to note that problem (2) has derivatives if the constraints of the original problem (1) have derivatives, since the quadratic model has first and second derivatives.

In order to use a nonlinear solver for problem (2), it is necessary to write an interface between TRDF and the desired solver. This interface is given by subroutine `SOLVER`:

```
      SUBROUTINE SOLVER(N, L, U, X, M, EQUATN, LINEAR, CCODED,
     +                  PHASE0, EPS, CNORM, FLAG)
```

The current implementation of TRDF provides two examples of interfaces: `algencan_solver.f` is an interface for the nonlinear solver ALGENCAN at version 3.0.0 and `algencan_old_solver.f` is an interface for version 2.2.1. Each time that TRDF subroutines call `SOLVER`, they expect that it returns the solution `X`, its sup-norm of infeasibility `CNORM` and a flag `FLAG`. If the solver finished correctly, the flag

must be 0. If problems have occurred during optimization of (2), then the flag must be 2.

Below we describe each argument of the solver interface.

- `N` (integer, input): the dimension of the problem
- `L(N)` and `U(N)` (double precision, input): the lower and upper bounds on variable `X`, respectively
- `X(N)` (double precision, input/output): on entry, is the point used by the solver as starting point. On return, is the solution of problem (2)
- `EQUATN(M)`, `LINEAR(M)` and `CCODED(2)` (logical, input): this arguments have already been explained in Section 4
- `PHASE0` (logical, input): if `.true.` indicates that a projection onto Ω should be performed (thus, the objective function should be ignored), otherwise indicates that the full problem (2) should be solved
- `EPS` (double precision, input): the feasibility tolerance
- `CNORM` (double precision, output): the sup-norm of the infeasibility at the solution found by the solver
- `FLAG` (integer, output): indicates the status when returning. Must be 0 if the solver converged correctly or 2 if errors have occurred

After creating the specific solver interface, variables `SOLVERLIB`, `SOLVER` and `SLOPTS` in `Makefile` have to be changed, in order to use the new solver's files and libraries when building TRDF.

More information

If there are any doubts, fell free to send an email to Paulo D. Conejo

pconejo33@gmail.com

or Francisco N. C. Sobral

fncsobral@uem.br

References

[1] P. D. Conejo, E. W. Karas, L. G. Pedroso. A trust-region derivative-free algorithm for constrained optimization. *Optimization Methods & Software*, to appear, 2015.

[2] R. Andreani, E. G. Birgin, J. M. Martnez, and M. L. Schuverdt. On Augmented Lagrangian Methods with General Lower-Level Constraints SIAM Journal on Optimization, vol. 18, no. 4, pp. 12861309, 2008.

[3] "Trustable Algorithms for Nonlinear General Optimization", www.ime.usp.br/∼egbirgin/tango.

[4] K. Schittkowski. Test Examples for Nonlinear Programming Codes - All Problems from the Hock-Schittkowski-Collection, 2009.