Using a MODULOPT Minimization Code

Claude Lemaréchal – Inria – BP 105 – 78153 Le Chesnay
Tel. (1) 39 63 56 81 – Fax. —- 57 86 – e-mail Claude.Lemarechal@inria.fr

1. Introduction

Modulopt is a French "club of minimizers" where nonlinear programming is studied. Its main activity consists in developing and exchanging Fortran programs, which are of essentially 2 kinds:
– Minimization algorithms, ranked in 4 classes: without constraints, with bounds, with linear constraints and with nonlinear constraints. Programs of this kind are called *Minimizers*.
– Test problems, ranked in the same classes. These programs are called *Users*.

A user is made of 2 parts:
– Part 1: A main program which makes the necessary dimensionments and initializations, then calls the minimizer, and then exploits the results.
– Part 2: A (set of) subroutine(s) which is called by the minimizer, and which computes the objective value and/or its gradient. We call this part the *Simulator*.

In Modulopt, the dialogue between the minimizer and the simulator is strictly normalized, and the aim of these notes is to describe the form of this dialogue, in the case of a problem without nonlinear constraints.

2. Entry to the simulator

It must have the form

```
SUBROUTINE SIMUL (INDIC, N, X, F, G, IS, RS, DS)
INTEGER INDIC, N, IS(*)
REAL X(N), F, G(N), RS(*)
DOUBLE PRECISION DS(*)
```

The name "SIMUL" is usually formal, i.e. it is a parameter in the calling sequence to the minimizer.

Some of the above parameters have an obvious meaning, namely:

N the number of variables involved in the optimization

X (dimension N) the value of the variables at which SIMUL must compute the function and/or its gradient

F the returned function value

G (dimension N) the returned gradient value.

The role of INDIC is double: to tell SIMUL what to do, and to answer what SIMUL has actually done. The role of IS, RS, DS is to allow SIMUL to share data with the main program, and to work with its own workspace.

3. The parameter INDIC

INDIC is an input-output integer; on input, it can have 4 values:

INDIC = 1 simply does not request anything from SIMUL. This can be useful for example with long problems that can be aborted unpredictably. Entering from time to time SIMUL with INDIC = 1 allows it to save the current iterate, to print useful information, etc.

INDIC = 2 means that SIMUL is requested to compute the function at the given X.

INDIC = 3 is to compute the gradient; in this case, the last previous call of SIMUL was with INDIC = 1 or 2, *and with the same x-values*; therefore, the possible overhead necessary to compute f and g can be considered already done.

INDIC = 4 is to compute the function *and* its gradients.

On output, INDIC can have 3 values

INDIC > 0: normal exit, every request has been accomplished.

`INDIC` $= 0$: please stop the iterations, the present `X` is good enough. This can be useful for example in least-squares identification: when the fitting is small enough, the *user* has a stopping criterion that is unavailable to a *non-specialized minimizer*.

`INDIC` < 0: something wrong, the request has not been accomplished. This can be useful for example in NLP with barrier method, when a problem such as

$$\min f(x) \quad \text{subject to } h(x) \leq 0$$

is replaced by something like

$$\min f(x) - \varepsilon \log(-h(x)).$$

When $h(x) \geq 0$, `SIMUL` can answer `INDIC` < 0 (and return no useful value in `F` or `G`) to warn the minimizer.

4. The parameters IS, RS, DS

These are sorts of superarrays, which work as follows:

- They appear in the calling sequence to the minimizer.
- The minimizer does not modify them.
- They are transmitted as such in the calling sequence to the simulator.

In this way, any data coming from the main program is available without `COMMON` block. Furthermore, parts of `IS`, `RS` and `DS` can be used as workspace by the simulator.

Of course, if there are several (say 2 real) vectors to be given to the simulator, then `SIMUL` will be just an interface dispatching them into appropriate places (say `RS(1)` and `RS(K)`) and calling the simulator properly said; otherwise the simulator would be unreadable.

5. An example

Take a problem in which one optimizes with respect to two (groups of) variables simultaneously: u (dimension n_1) and v (dimension n_2). In this problem, data are stored in a vector A; suppose also that the simulator needs as workspace two vectors of length n_1 and n_2 respectively. The following listing gives an idea of what the user could look like, in Fortran 4. Note

that the subroutine FUNGRA works with real names and does not bother with dimensionments.

```
      DIMENSION U(100), V(50), A(200), X(150), RS(350)
C          why not ?
      COMMON /INTEG/ N1,N2,NA,NFILE
      EQUIVALENCE (A,RS)
      NA=87
      N1=18
      N2=11
C          why not ?
      N=N1+N2
              etc.
      STOP
      END

      SUBROUTINE SIMUL (INDIC,N,X,F,G,IS,RS,DS)
      COMMON /INTEG/ N1,N2,NA,NIFLE
      DIMENSION X(*),G(*),IS(*),RS(*)
      DOUBLE PRECISION DS(*)
      IF (INDIC.GT.1) GO TO 100
      WRITE (NFILE) (X(I), I=1,N)
      RETURN
  100 CALL FUNGRA (N1,N2,X,X(N1+1),F,G,G(N1+1),
     1 RS,RS(NA+1),RS(NA+N1+1))
      RETURN
      END

      SUBROUTINE FUNGRA (N1,N2,U,V,F,GU,GV,A,Z1,Z2)
      DIMENSION U(*),V(*),GU(*),GV(*),A(*),Z1(*),Z2(*)
              etc.
      RETURN
      END
```