The contributions of the ANR/OMD project to the optimization with Scilab: decoupling the optimizers from the simulators

ScilabTec'10

Presented by: Rodolphe Le Riche, CNRS and Ecole des Mines de St-Etienne

Thanks: Gilles Pujol, Yann Collette, Niko Hansen, Daniel Salazar, Rajan Filomeno Coelho, Régis Duvigneau, Pierre Alain Boucard, Florian DeVuyst, Sandrine Buytet, Piotr Breitkopf, Catherine Vayssade, Joel Clément, Michael Baudin.
Basic numerical optimization concepts

Numerical optimization is used to

- Improve the performance of a system
- Identify the parameters of a numerical simulator (inverse problem, optimal control)
- Solve coupled non linear equations

Optimization problem notation:

\[ \min_{x \in \mathbb{R}^n} f(x) \quad \text{e.g., } x \equiv \text{wing shape, } f(x) \equiv \text{drag}(x) \]
\[ \text{s.t. } g(x) \leq 0 \quad \text{e.g., } g(x) \equiv \text{weight}(x) - \text{lift}(x) \]

Typically, a dialog between two programs:

- Optimizer
- Simulator

\[ f(x), \quad g(x) \]
Motivations for the OMD project

( OMD = MultiDisciplinary Optimization )

Simulation (e.g., finite elements) and numerical optimization naturally become incompatible: simulation moves away from optimization, optimization does not use all expert information.

simulation vs. optimization

is one point in
Why simulation and optimization branch off?

- Computing time of the simulations is too long for mathematical optimization methods.
- Optimization and simulation need to account for uncertainties and model errors, which makes the computing time even longer.
- Most complex systems are described by different disciplines (hence different simulators) which are difficult to put together during the optimization for numerical and human reasons.
The OMD Project Work Packages

- Methodological: metamodeling, multi-level optimization, optimization with uncertain simulations, collaborative optimization.

- Test cases:

- Software: Scilab (greatest common free language) to encourage exchanges between the 14 partners during the 3 years of the project.
(Simple notations: neglect optimization constraints here)

\[ \min_{x \in \mathbb{R}^n} f(x) \]

An optimizer is an algorithm that iteratively proposes new \( x \)'s based on past trials in order to approximate the solution to the optimization problem

\[ x(t+1) = \text{Optimizer}[x(1), f(x(1)), \ldots, x(t), f(x(t))] \]
Typical procedural implementation of optimization programs:

Signature of the simulator,

\[
[f] = \text{simulator}(x)
\]

Signature of the optimizer,

\[
[f_{opt}, x_{opt}] = \text{optimizer}(\text{simulator}, x_0, \text{... stopping parameters ...})
\]

The « user » performs the optimization through one call to the optimizer. Hierarchical: optimizer on top.
Which user?

« The user performs the optimization through one call to the optimizer. » Which user?

- Real user: plugs the optimizer onto the simulation (e.g., by writing the \[ f = \text{simulator}(x) \] function), do a few optimizations.


- Researcher: proposes new optimizers, i.e., new relationships between optimizers and simulators.

OMD project: many « expert » and « researcher » type of users.
For experts and researchers, the procedural programming pattern is an obstacle to creativity: it hides simulators / optimizers possible relationships.

For example, how to mix, online, several optimizers? (e.g., local and global optimizers, an idea motivated by the No Free Lunch Theorem)

Note: Any optimization algorithm can be programmed in a procedural way, wrap everything inside one optimizer; or use of the ind parameter in optim from Scilab, ... . But: non intuitive, little code recycling.
The ask & tell pattern: intro

- OMD collaborators encouraged to program with the ask & tell pattern.
- Not new: cf. « reverse communication », e.g., ARPACK and ALGLIB libraries.
- ask & tell is pseudo object-oriented programming in Scilab.

Typical optimization loop:

```plaintext
opt = rsearch() // optimizer constructor

while ~stop(opt)
    x = ask(opt)
    y = simulator(x)
    opt = tell(opt, x, y)
end
[yopt, xopt] = best(opt)
```

→ optimizer and simulator calls are decoupled.
→ optimizer and simulator are at the same hierarchical level.
// constructor, notice the mlist
function this = rsearch()
    this = mlist(['rsearch','xmin','xmax','iter','_x','_y'])
    this.xmin = []
    (...)
endfunction
// + other constructor with parameter passing, not given here.

// ask, tell, stop, best : the % announces overloading

function x =%rsearch_ask(this)
    d = length(this.xmin)
    x = (this.xmax-this.xmin).*grand(1,d,'def')+this.xmin
endfunction

function this = %rsearch_tell(this,x,y)
    if (y < this._y) then this._x = x; this._y = y ; end;
    this.iter = this.iter -1;
endfunction
ask & tell in Scilab:
Example with rsearch (2/2)

// overloading mechanism with execstr

function x = %rsearch_ask(this)
    d = length(this.xmin)
    x = (this.xmax-this.xmin).*grand(1,d,'def')+this.xmin
endfunction

function x = ask(this)
    execstr('x = %' + typeof(this) + '_ask(this)')
endfunction

function this = tell(this,x,y)
    execstr('this = %' + typeof(this) + '_tell(this,x,y)')
endfunction

// and similarly with stop and best.
// follow the same pattern for other optimizers.

→ better modularity at the cost of an additional programming effort to manage the optimizer state.
ask & tell in Scilab: Multistart example (1/3)

Static multistart:
- repeat $N$ local optimizations, of $n$ simulator calls, starting from a randomly chosen point.
- Total budget: $\text{neval} = N \times n$.
- Easy to implement with procedural optimizers.
- But some local searches waste evaluations (early stop, convergence to already searched regions).

Dynamic multistart:
- while $\text{neval} < N \times n$ do,
  - start a new local search
  - interrupt if it converges to an already searched region
  - update $\text{neval}$ online
- Difficult to code using procedural optimizers (not easy to keep an archive and stop the optimization during the run).
neval = 1000 // total evaluation budget
Arch = [] // archive of visited points
opt_loc = descent() ; opt_glob = rsearch(); // optimizers

while neval > 0 do
    opt_loc.x0 = ask(opt_glob)
    opt_loc.iter = min(100,neval) // allocate opt budget
    // ... other opt_loc settings ...
    neval = neval - opt_loc.iter
    while ~stop(opt_loc) do
        x = ask(opt_loc)
        dist = calc_distance_to_archive(Arch,x)
        if (dist<0.1) then
            opt_loc.stop = %t
        else
            [y,dy]=branin(x) // 2D function as simulator
            opt_loc = tell(opt_loc,x,list(y,dy))
            Arch = update_archive(Arch,x)
        end
    end
    neval = neval + opt.iter // restore unused budget
end
For the same total budget, better covering of the domain by the dynamic multistart. Illustration on Branin function.
ask & tell in Scilab:
A multi-fidelity example (1/2)

**simulator:**
costly high fidelity simulation
= S1

**MM(x):** cheap metamodel
prediction at \( \mathbf{x} \),
= S2

**MM_error(x):** uncertainty of MM at \( \mathbf{x} \).
ask & tell in Scilab:
A multi-fidelity example (2/2)

```plaintext
neval = 1000 // total evaluation budget
MM = kriging(X0,Y0) // MetaModel, a simulator built from
    // X0,Y0 initial design of experiments
while neval > 0 do
  opt = descent(); // optimizer
  while ~stop(opt) & MM_error(x) <0.1 do
    x = ask(opt)
    y=MM(x)
    opt = tell(opt,x,y)
  end
  xnew = best(opt)
  ynew = simulator(xnew)
  neval = neval - 1 // remaining calls to real simulator
  MM = MM_update(MM,xnew,ynew);
end
```

**MM** and **opt** can easily be changed.
Online stopping decision not nicely accounted for in procedural optimizers.
Existing ask & tell optimizers in the OMD toolbox

- **descent**: steepest descent
- **rsearch**: random search (uniform or Gaussian with distribution updating)
- **mulambda**: evolution strategy, ES(μ '+' or ',' λ).
- **cma**: CMA-ES, a state-of-the-art stochastic optimizer.

Output expl.:
Other Scilab / OMD developments (1/2)

Metamodels
= simulators made from given [x,y] data

- Radial basis functions
- Kriging
- Moving least squares
- Proper Orthogonal Decomposition (POD), for fields.

Interfacing utilities with external codes
when the simulator calls an external software

- template_replace : replaces chunks of codes with values in a template file (useful to make input files to external softwares)
- parse : parse ASCII files from keywords (useful to read output files from external softwares)
- Use stdin to interface a Scilab optimizer with CAST3M (finite elements software) so that CAST3M does not hang up during the iterations.

Optimizer (Scilab) \( f, g \) stdin (perl) Simulator (CAST3M)
Other Scilab / OMD developments (2/2)

Graphics
for representing sets of x's, f's and g's

- **pcoorplot**: parallel coordinates plots (to represent multidimensional data)
- **pairsplot**: scatter plot matrix. Example:
OMD project is finished, but OMD2 (OMD Distributed) and ID4CS (Integrative Design for Complex Systems) are on-going.

Handling uncertainties in the optimization

- \( f(x) \) becomes \( f(x,U) \), where \( U \) is a random variable.
- encapsulation statistical procedures to compare sets of Monte Carlo simulations \([f(x,u^1), \ldots, f(x,u^k)]\) with \([f(x',u^1), \ldots, f(x',u^k)]\).

Distributed optimization

- distribute evaluations of the simulator through non blocking calls to the simulator (cf. preceding presentation by F. Viale and D. Caromel). Useful to be able to dynamically « ask » x's based on computing node availability.
- distribute optimizers. ask & tell optimizers are mlist's with well defined fields (states), which facilitates information exchanges.