Stopping.jl : A framework to implement iterative optimization algorithms

Tangi Migot[1]
tmigot@uoguelph.ca

Joint work with Jean-Pierre Dussault and Samuel Goyette[2]

Journées de l'optimisation, HEC Montréal, 2019

[1] Department of Mathematics and Statistics, University of Guelph, Ontario
[2] BISOUS, Université de Sherbrooke, Québec

Outline

In this talk:

Outline

In this talk:

- we discuss the implementation of iterative algorithms for non-linear (continuous) optimization

Outline

In this talk:

- we discuss the implementation of iterative algorithms for non-linear (continuous) optimization
- for researchers, programmers, (curious) practionners

Outline

In this talk:

- we discuss the implementation of iterative algorithms for non-linear (continuous) optimization
- for researchers, programmers, (curious) practionners

Disclaimer

Our aim is to open the discussion on what would help facilitate reuse of existing codes.

Our general motivation was to tackle some "sophisticated" non-linear problems:

Our general motivation was to tackle some "sophisticated" non-linear problems:

1. Linearly constrained optimization problems

$$\min_{x \in \mathbb{R}^n} f(x) \text{ s.t. } Ax \leq b$$

Our general motivation was to tackle some "sophisticated" non-linear problems:

1. Linearly constrained optimization problems

$$\min_{x\in\mathbb{R}^n} f(x) \text{ s.t. } Ax \leq b$$

2. Mathematical Programs with Complementarity Constraints (MPCC)

$$\min_{x\in\mathbb{R}^n} f(x) \text{ s.t. } 0 \leq G(x) \perp H(x) \geq 0$$

Our general motivation was to tackle some "sophisticated" non-linear problems:

1. Linearly constrained optimization problems

$$\min_{x\in\mathbb{R}^n} f(x) \text{ s.t. } Ax \leq b$$

2. Mathematical Programs with Complementarity Constraints (MPCC)

$$\min_{x\in\mathbb{R}^n} f(x) \text{ s.t. } 0 \leq G(x) \perp H(x) \geq 0$$

3. Generalized Nash Equilibrium Problems (GNEP)

$$\left\{ \begin{array}{l} \min\limits_{x\in\mathbb{R}^{n_1}} f_1(x,y) \\ \text{s.t. } l_{c_1} \leq c_1(x,y) \leq u_{c_1} \end{array} \right\} \text{ and } \left\{ \begin{array}{l} \min\limits_{y\in\mathbb{R}^{n_2}} f_2(x,y) \\ \text{s.t. } l_{c_2} \leq c_2(x,y) \leq u_{c_2} \end{array} \right\}$$

Linearly constraint optimization problem

$$\min_{x\in\mathbb{R}^n} f(x) \text{ s.t. } Ax \leq b.$$

Active set algorithm:

Linearly constraint optimization problem

$$\min_{x \in \mathbb{R}^n} f(x) \text{ s.t. } Ax \leq b.$$

Active set algorithm: given a feasible point x^0, identify the active constraints $\mathcal{I} \subset \{1, \dots, m\}$ at x^0. Let $j = 0$.

- S.1 $Z = \mathrm{Ker}(A_{\mathcal{I}})$
- S.2 Minimize in the working space:

$$d \in \arg\min_{d \in \mathbb{R}^n} f(x^j + Zd) \text{ s.t. } A(x^j + Zd) \leq b$$

- S.3 $x^{j+1} = x^j + Zd$
- S.4 Update the working space (add or remove constraints)
- S.5 x^{j+1} satisfies a criterion or go to S.1 $j = j + 1$

Linearly constraint optimization problem

$$\min_{x\in\mathbb{R}^n} f(x) \text{ s.t. } Ax \leq b.$$

Active set algorithm: given a feasible point x^0, identify the active constraints $\mathcal{I} \subset \{1,\dots,m\}$ at x^0. Let $j=0$.

S.1 $Z = \mathrm{Ker}(A_{\mathcal{I}})$

S.2 Minimize in the working space:

$$d \in \arg\min_{d\in\mathbb{R}^n} f(x^j + Zd) \text{ s.t. } A(x^j + Zd) \leq b$$

S.3 $x^{j+1} = x^j + Zd$

S.4 Update the working space (add or remove constraints)

S.5 x^{j+1} satisfies a criterion or go to S.1 $j = j+1$

This method iteratively solves an unconstrained optimization problem with a tweaked stopping criterion.

This method iteratively solves a **simpler** optimization problem with a tweaked stopping criterion.
This is a common theme to our problems...

Regularization methods for MPCC ($t \downarrow 0$)

$$\min_{x \in \mathbb{R}^n} f(x) \text{ s.t. } G(x) \geq 0, H(x) \geq 0, \Phi(G(x), H(x); t) \leq 0.$$

Solve a sequence of "well-behaved" non-linear optimization problems.

J.-P. Dussault, M. Haddou, A. Kadrani, and T. Migot.
How to compute an M-stationary point of the MPCC.
optimization-online.org.

We showed that the regularization process works if for each t, we verify the following ϵ_k-optimality conditions:

$$\left\|\nabla \mathcal{L}_R(x, \lambda^g, \lambda^h, \lambda^G, \lambda^H, \lambda^\Phi)\right\|_\infty \leq \epsilon_k$$

with

$$\|h(x)\|_\infty \leq \epsilon_k,\ g(x) \leq \epsilon_k,\ \lambda^g \geq 0,\ \|\lambda^g \circ g(x)\|_\infty \leq \epsilon_k,$$
$$G(x) + \bar{t}_k \geq -\epsilon_k,\ \lambda^G \geq 0,\ \|\lambda^G \circ (G(x) + \bar{t}_k)\|_\infty \leq \epsilon_k,$$
$$H(x) + \bar{t}_k \geq -\epsilon_k,\ \lambda^H \geq 0,\ \|\lambda^H \circ (H(x) + \bar{t}_k)\|_\infty \leq \epsilon_k,$$
$$\Phi(G(x), H(x); t_k) \leq 0,\ \lambda^\Phi \geq 0,\ \|\lambda^\Phi \circ \Phi(G(x), H(x); t_k)\|_\infty \leq 0,$$

whenever $\epsilon_k = o(\bar{t}_k)$ and $t_k \leq \bar{t}_k - c\epsilon_k$

J.-P. Dussault, M. Haddou, A. Kadrani, and T. Migot.
How to compute an M-stationary point of the MPCC.
optimization-online.org.

We showed that the regularization process works if for each t, we verify the following ϵ_k-optimality conditions:

$$\left\|\nabla\mathcal{L}_R(x,\lambda^g,\lambda^h,\lambda^G,\lambda^H,\lambda^\Phi)\right\|_\infty \leq \epsilon_k$$

with

$$\|h(x)\|_\infty \leq \epsilon_k,\ g(x) \leq \epsilon_k,\ \lambda^g \geq 0,\ \|\lambda^g \circ g(x)\|_\infty \leq \epsilon_k,$$
$$G(x) + \bar{t}_k \geq -\epsilon_k,\ \lambda^G \geq 0,\ \|\lambda^G \circ (G(x) + \bar{t}_k)\|_\infty \leq \epsilon_k,$$
$$H(x) + \bar{t}_k \geq -\epsilon_k,\ \lambda^H \geq 0,\ \|\lambda^H \circ (H(x) + \bar{t}_k)\|_\infty \leq \epsilon_k,$$
$$\Phi(G(x),H(x);t_k) \leq 0,\ \lambda^\Phi \geq 0,\ \|\lambda^\Phi \circ \Phi(G(x),H(x);t_k)\|_\infty \leq 0,$$

whenever $\epsilon_k = o(\bar{t}_k)$ and $t_k \leq \bar{t}_k - c\epsilon_k$

This method iteratively solve a **simpler** optimization problem with a twicked stopping criteration.
This is a common theme to our problems...

Decomposition methods for the GNEP. Solve alternatively the two optimization problems:

$$x^{k+1} \in \arg\min_{x \in \mathbb{R}^{n_1}} f_1(x, y^k) \text{ s.t. } l_{c_1} \leq c_1(x, y^k) \leq u_{c_1}$$

$$y^{k+1} \in \arg\min_{y \in \mathbb{R}^{n_2}} f_2(x^{k+1}, y) \text{ s.t. } l_{c_2} \leq c_2(x^{k+1}, y) \leq u_{c_2}$$

This method iteratively solve a **simpler** optimization problem with a twicked stopping criteration.
This is a common theme to our problems...

Decomposition methods for the GNEP. Solve alternatively the two optimization problems:

$$x^{k+1} \in \arg\min_{x\in\mathbb{R}^{n_1}} f_1(x, y^k) \text{ s.t. } l_{c_1} \leq c_1(x, y^k) \leq u_{c_1}$$

$$y^{k+1} \in \arg\min_{y\in\mathbb{R}^{n_2}} f_2(x^{k+1}, y) \text{ s.t. } l_{c_2} \leq c_2(x^{k+1}, y) \leq u_{c_2}$$

1st Idea

In these 3 examples the stopping criterion of the subproblem(s) can be impacted by the fact that the algorithm is used in another loop.

2nd Idea

Is the stopping criterion well-defined in general?

2nd Idea

Is the stopping criterion well-defined in general?

For an unconstrained optimization problem, we are usually looking for x^* satisfying the 1st order optimality conditions

$$\nabla f(x^*) = 0.$$

2nd Idea

Is the stopping criterion well-defined in general?
For an unconstrained optimization problem, we are usually looking for x^* satisfying the 1st order optimality conditions

$$\nabla f(x^*) = 0.$$

Well, but in practice ?

2nd Idea

Is the stopping criterion well-defined in general?
For an unconstrained optimization problem, we are usually looking for x^* satisfying the 1st order optimality conditions

$$\nabla f(x^*) = 0.$$

Well, but in practice ?

$$\|\nabla f(x^*)\|_p \leq \epsilon$$

2nd Idea

Is the stopping criterion well-defined in general?
For an unconstrained optimization problem, we are usually looking for x^* satisfying the 1st order optimality conditions

$$\nabla f(x^*) = 0.$$

Well, but in practice ?

$$\|\nabla f(x^*)\|_p \leq \epsilon \text{ or } \|\nabla f(x^*)\|_p \leq \epsilon \|x^0\|$$

2nd Idea

Is the stopping criterion well-defined in general?
For an unconstrained optimization problem, we are usually looking for x^* satisfying the 1st order optimality conditions

$$\nabla f(x^*) = 0.$$

Well, but in practice ?

$$\|\nabla f(x^*)\|_p \leq \epsilon \text{ or } \|\nabla f(x^*)\|_p \leq \epsilon \|x^0\|$$

with different choices for $p = \{1, 2, \infty\}$.

2nd Idea

Is the stopping criterion well-defined in general?
For an unconstrained optimization problem, we are usually looking for x^* satisfying the 1st order optimality conditions

$$\nabla f(x^*) = 0.$$

Well, but in practice ?

$$\|\nabla f(x^*)\|_p \leq \epsilon \text{ or } \|\nabla f(x^*)\|_p \leq \epsilon \|x^0\|$$

with different choices for $p = \{1, 2, \infty\}$.
If f has a specific structure we might want

$$\|\nabla_x f(x, y)\|_p \leq \epsilon_x \text{ and } \|\nabla_y f(x, y)\|_p \leq \epsilon_y$$

etc...

2nd Idea

Is the stopping criterion well-defined in general?
For an unconstrained optimization problem, we are usually looking for x^* satisfying the 1st order optimality conditions

$$\nabla f(x^*) = 0.$$

Well, but in practice ?

$$\|\nabla f(x^*)\|_p \leq \epsilon \text{ or } \|\nabla f(x^*)\|_p \leq \epsilon \|x^0\|$$

with different choices for $p = \{1, 2, \infty\}$.
If f has a specific structure we might want

$$\|\nabla_x f(x, y)\|_p \leq \epsilon_x \text{ and } \|\nabla_y f(x, y)\|_p \leq \epsilon_y$$

etc...
It gets worse for non-linear optimization and the KKT conditions (in particular to anticipate arbitrarily large multipliers).

Motivation: Sum Up

Our starting point:

Motivation: Sum Up

Our starting point:

- Algorithms are designed to solve a given problem based on theoretical optimality conditions

Motivation: Sum Up

Our starting point:

- Algorithms are designed to solve a given problem based on theoretical optimality conditions
- Practical implementation of optimality conditions may vary (depending on the application, some heuristics, some parameters choices ...)

Motivation: Sum Up

Our starting point:

- Algorithms are designed to solve a given problem based on theoretical optimality conditions
- Practical implementation of optimality conditions may vary (depending on the application, some heuristics, some parameters choices ...)
- Some extra conditions in the stopping criterion can be useful (for instance when this problem serves as a subproblem of a more complex one)

Motivation: Sum Up

Our starting point:

- Algorithms are designed to solve a given problem based on theoretical optimality conditions
- Practical implementation of optimality conditions may vary (depending on the application, some heuristics, some parameters choices ...)
- Some extra conditions in the stopping criterion can be useful (for instance when this problem serves as a subproblem of a more complex one) and virtually impossible to anticipate

Motivation: Sum Up

Our starting point:

- Algorithms are designed to solve a given problem based on theoretical optimality conditions
- Practical implementation of optimality conditions may vary (depending on the application, some heuristics, some parameters choices ...)
- Some extra conditions in the stopping criterion can be useful (for instance when this problem serves as a subproblem of a more complex one) and virtually impossible to anticipate

The general idea

One convenient solution to these problems is to put the stopping criterion as an **input** of the problem.

Motivation: Sum Up

Our starting point:

- Algorithms are designed to solve a given problem based on theoretical optimality conditions
- Practical implementation of optimality conditions may vary (depending on the application, some heuristics, some parameters choices ...)
- Some extra conditions in the stopping criterion can be useful (for instance when this problem serves as a subproblem of a more complex one) and virtually impossible to anticipate

The general idea

One convenient solution to these problems is to put the stopping criterion as an **input** of the problem.
Stopping.jl offers a structure to such implementation.

A Generic Algorithm

Given an initial point x^0. Let $j = 0$.

S.1 $d, infos$ = solve_a_subproblem(x^j, *parameters*)

S.2 $x^{j+1} = x^j + d$

S.3 Update the *infos* and algorithm *parameters*

S.4 x^{j+1} satisfies a criterion based on the *infos*, **or** go to S.1 $j = j+1$

A Generic Algorithm

Given an initial point x^0. Let $j = 0$.

S.1 $d, infos$ = solve_a_subproblem(x^j, *parameters*, optimality)

S.2 $x^{j+1} = x^j + d$

S.3 Update the *infos* and algorithm *parameters*

S.4 x^{j+1} satisfies a criterion based on the *infos*, **or** go to S.1 $j = j+1$

A Generic Algorithm

Given an initial point x^0. Let $j = 0$.

S.1 $d, infos$ = solve_a_subproblem(x^j, *parameters*, optimality)

S.2 $x^{j+1} = x^j + d$

S.3 Update the *infos* and algorithm *parameters*

S.4 x^{j+1} satisfies a criterion based on the *infos*, **or** go to S.1 $j = j + 1$

Two families of objects that are independent of the algorithm (i.e. more connected to the problem itself):

A Generic Algorithm

Given an initial point x^0. Let $j = 0$.

S.1 $d, infos$ = solve_a_subproblem(x^j, *parameters*, optimality)

S.2 $x^{j+1} = x^j + d$

S.3 Update the *infos* and algorithm *parameters*

S.4 x^{j+1} satisfies a criterion based on the *infos*, **or** go to S.1 $j = j+1$

Two families of objects that are independent of the algorithm (i.e. more connected to the problem itself):

- *infos*

A Generic Algorithm

Given an initial point x^0. Let $j = 0$.

S.1 $d, infos$ = solve_a_subproblem(x^j, *parameters*, optimality)

S.2 $x^{j+1} = x^j + d$

S.3 Update the *infos* and algorithm *parameters*

S.4 x^{j+1} satisfies a criterion based on the *infos*, **or** go to S.1 $j = j+1$

Two families of objects that are independent of the algorithm (i.e. more connected to the problem itself):

- *infos*
- stopping check (optimality function in step 1 and in step 4).

A Generic Algorithm

Given an initial point x^0. Let $j = 0$.

S.1 $d, infos$ = solve_a_subproblem(x^j, *parameters*, optimality)

S.2 $x^{j+1} = x^j + d$

S.3 Update the *infos* and algorithm *parameters*

S.4 x^{j+1} satisfies a criterion based on the *infos*, **or** go to S.1 $j = j+1$

2 main structures outside the algorithm

- **State**:
- **Stopping**:

A Generic Algorithm

Given an initial point x^0. Let $j = 0$.

S.1 d, *infos* $=$ solve_a_subproblem(x^j, *parameters*, optimality)

S.2 $x^{j+1} = x^j + d$

S.3 Update the *infos* and algorithm *parameters*

S.4 x^{j+1} satisfies a criterion based on the *infos*, **or** go to S.1 $j = j + 1$

2 main structures outside the algorithm

- **State**: keep track of the various informations connected to a problem.
- **Stopping**:

A Generic Algorithm

Given an initial point x^0. Let $j = 0$.

S.1 $d, infos$ = solve_a_subproblem(x^j, *parameters*, optimality)

S.2 $x^{j+1} = x^j + d$

S.3 Update the *infos* and algorithm *parameters*

S.4 x^{j+1} satisfies a criterion based on the *infos*, **or** go to S.1 $j = j + 1$

2 main structures outside the algorithm

- **State**: keep track of the various informations connected to a problem.
- **Stopping**: can say if we are done or not for a given State.

General Organization

Algorithm(stp)
Your favorite algorithm using Stopping to ask if it should stop or not

Stopping
- Attributes: Problem, State, optimality
- start!, stop!, fill_in!

Problem
Instance of a problem

State
Data relative to the problem at x^j

General Organization

Algorithm(stp)
Your favorite algorithm using Stopping to ask if it should stop or not

Stopping
- Attributes: Problem, State, optimality
- start!, stop!, fill_in!

Problem
Instance of a problem

State
Data relative to the problem at x^j

General Organization

Algorithm(stp)
Your favorite algorithm using Stopping to ask if it should stop or not

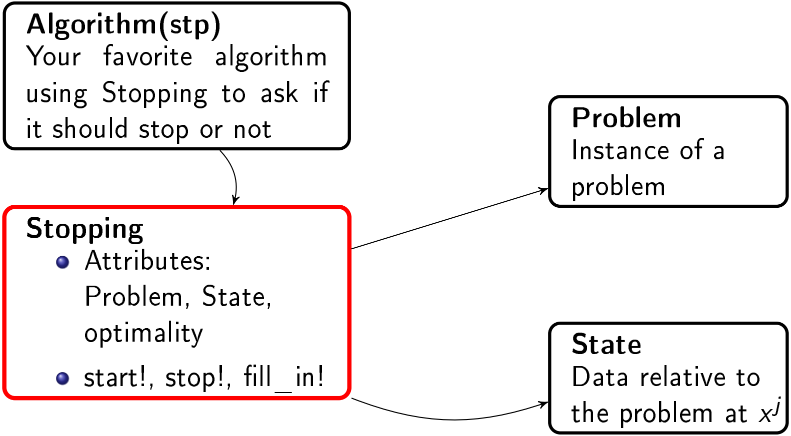
Stopping

- Attributes: Problem, State, optimality
- start!, stop!, fill_in!

Problem
Instance of a problem

State
Data relative to the problem at x^j

General Organization



General Organization

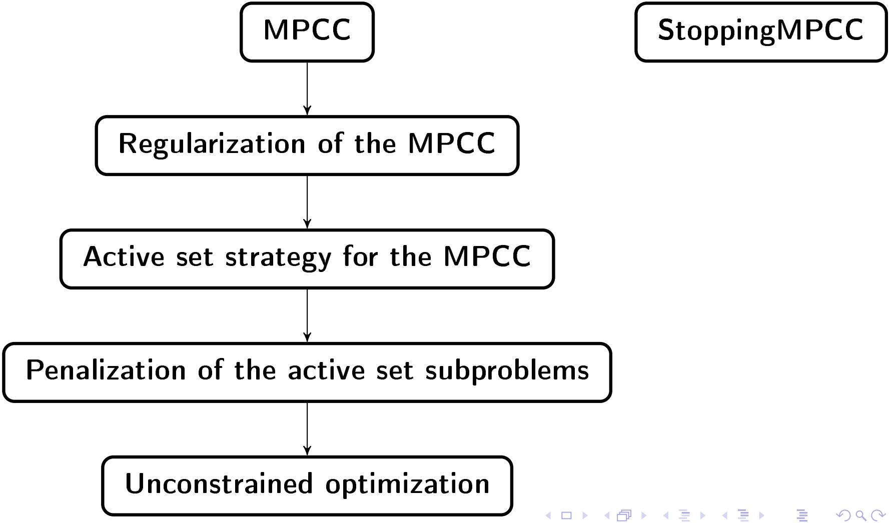
Algorithm(stp)
Your favorite algorithm using Stopping to ask if it should stop or not

Stopping
- Attributes: Problem, State, optimality
- start!, stop!, fill_in!

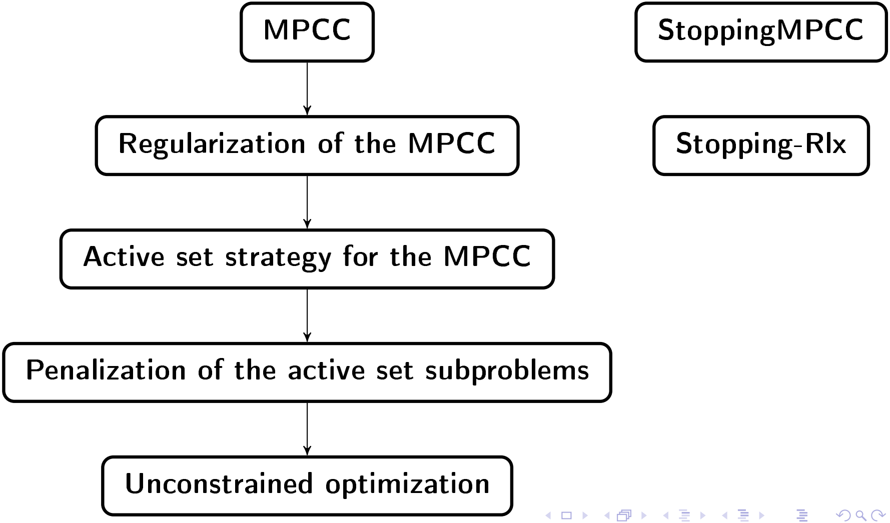
Problem
Instance of a problem

State
Data relative to the problem at x^j

Stacking the Stoppings: regularization-active set-penalization algorithm for MPCCs



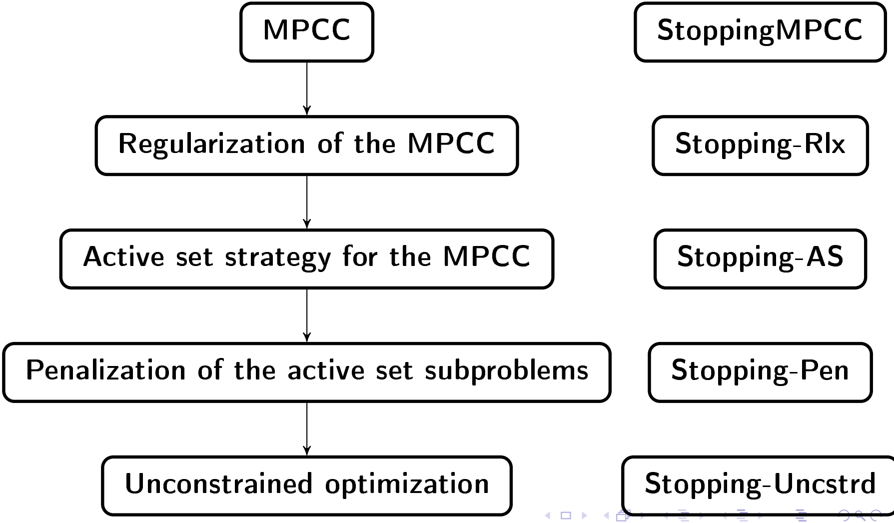
Stacking the Stoppings: regularization-active set-penalization algorithm for MPCCs



Stacking the Stoppings: regularization-active set-penalization algorithm for MPCCs

MPCC | StoppingMPCC
↓
Regularization of the MPCC | Stopping-Rlx
↓
Active set strategy for the MPCC | Stopping-AS
↓
Penalization of the active set subproblems
↓
Unconstrained optimization

Stacking the Stoppings: regularization-active set-penalization algorithm for MPCCs



Newton method

Example

Sometimes we think in term of codes.

Newton method

Example

Sometimes we think in term of codes.

Say we want to solve an unconstrained quadratic optimization problem

$$\min_{x \in \mathbb{R}^n} f(x) := x^T Q x$$

Newton method

Example

Sometimes we think in term of codes.

Say we want to solve an unconstrained quadratic optimization problem

$$\min_{x \in \mathbb{R}^n} f(x) := x^T Q x$$

One instance using NLPModels.jl package:

```
A = rand(5, 5); Q = A' * A;
f(x) = 0.5 * x' * Q * x
nlp = ADNLPModel(f,  ones(5))
```

Newton method

Example

Sometimes we think in term of codes.

Say we want to solve an unconstrained quadratic optimization problem

$$\min_{x \in \mathbb{R}^n} f(x) := x^T Q x$$

One instance using NLPModels.jl package:

```
A = rand(5, 5); Q = A' * A;
f(x) = 0.5 * x' * Q * x
nlp = ADNLPModel(f,  ones(5))
```

Now, we create the State and Stopping associate to this instance

```
nlp_at_x = NLPAtX(ones(5))
stop_nlp = NLPStopping(nlp,
                      (x,y) -> Stopping.unconstrained(x,y),
                      nlp_at_x)
```

Newton method

```
function newton(stp :: NLPStopping)
  state = stp.current_state

  update!(state, x = state.x, gx = grad(stp.pb, state.x),
                Hx = hess(stp.pb, state.x))
  OK = start!(stp)

  while !OK
    d = -inv(state.Hx) * state.gx
    update!(state, x = state.x + d, gx = grad(stp.pb, xt),
                  Hx = hess(stp.pb, xt))
    OK = stop!(stp)
  end

  return stp
end
```

Newton method

```
stop_nlp = newton(stop_nlp)

@show stop_nlp.meta.tired #ans: false
@show stop_nlp.meta.unbounded #ans: false
@show stop_nlp.meta.optimal #ans: true
```

Sum up

- **Stopping.jl**

Sum up

- **Stopping.jl** is a set of tools to ease the uniformization of stopping criteria in iterative solvers.

Sum up

- **Stopping.jl** is a set of tools to ease the uniformization of stopping criteria in iterative solvers.
- When a solver is called on an optimization model, four outcomes may happen:

Sum up

- **Stopping.jl** is a set of tools to ease the uniformization of stopping criteria in iterative solvers.
- When a solver is called on an optimization model, four outcomes may happen:
    1. the approximate solution is obtained, the problem is considered solved
    2. the problem is declared unsolvable (unboundedness, infeasibility ...)
    3. the maximum available ressources is not sufficient to compute the solution
    4. some algorithm dependent failure happens

Sum up

- **Stopping.jl** is a set of tools to ease the uniformization of stopping criteria in iterative solvers.
- When a solver is called on an optimization model, four outcomes may happen:
  1. the approximate solution is obtained, the problem is considered solved
  2. the problem is declared unsolvable (unboundedness, infeasibility ...)
  3. the maximum available ressources is not sufficient to compute the solution
  4. some algorithm dependent failure happens

  This tool eases the first 3 items above.

Sum up

- **Stopping.jl** is a set of tools to ease the uniformization of stopping criteria in iterative solvers.
- When a solver is called on an optimization model, four outcomes may happen:
  1. the approximate solution is obtained, the problem is considered solved
  2. the problem is declared unsolvable (unboundedness, infeasibility ...)
  3. the maximum available ressources is not sufficient to compute the solution
  4. some algorithm dependent failure happens

  This tool eases the first 3 items above.
- It is not hard to translate a code in the Stopping framework + the code keeps the algorithmic form.

Sum up

- **Stopping.jl** is a set of tools to ease the uniformization of stopping criteria in iterative solvers.
- When a solver is called on an optimization model, four outcomes may happen:
  1. the approximate solution is obtained, the problem is considered solved
  2. the problem is declared unsolvable (unboundedness, infeasibility ...)
  3. the maximum available ressources is not sufficient to compute the solution
  4. some algorithm dependent failure happens

  This tool eases the first 3 items above.
- It is not hard to translate a code in the Stopping framework + the code keeps the algorithmic form.
- and more...

Library of functions

- Context: you have a library of functions to solve a given problem

Library of functions

- Context: you have a library of functions to solve a given problem
- However, the stopping criterion may differ or you want to compare several stopping criteria

Library of functions

- Context: you have a library of functions to solve a given problem
- However, the stopping criterion may differ or you want to compare several stopping criteria
- The Stopping framework allows such behavior with no modifications of the code

Library of functions

- Context: you have a library of functions to solve a given problem
- However, the stopping criterion may differ or you want to compare several stopping criteria
- The Stopping framework allows such behavior with no modifications of the code

In practice

Work of S. Goyette and JP. Dussault on 1d minimization methods - **LineSearch.jl**
For 1d methods, results differ if you want exact minimization, Armijo condition, Armijo-Wolfe condition ...

Benchmarking

Benchmarking

- In the following classical paper on benchmarking with performance profiles

Dolan, Elizabeth D. and Moré, Jorge J. and Munson, Todd S.
Optimality Measures for Performance Profiles.
SIAM Journal on Optimization, 16(3):891–909, jan. 2006.

The authors set up an optimality criterion to compare non-linear optimization solvers.

Benchmarking

- In the following classical paper on benchmarking with performance profiles

  Dolan, Elizabeth D. and Moré, Jorge J. and Munson, Todd S.
  Optimality Measures for Performance Profiles.
  SIAM Journal on Optimization, 16(3):891–909, jan. 2006.

  The authors set up an optimality criterion to compare non-linear optimization solvers.
- Ok, but what if the solver ends with a sucess but fails to satisfy the optimality measure?

Benchmarking

- In the following classical paper on benchmarking with performance profiles

  Dolan, Elizabeth D. and Moré, Jorge J. and Munson, Todd S.
  Optimality Measures for Performance Profiles.
  SIAM Journal on Optimization, 16(3):891–909, jan. 2006.

  The authors set up an optimality criterion to compare non-linear optimization solvers.
- Ok, but what if the solver ends with a sucess but fails to satisfy the optimality measure?"*if the convergence test proposed in section 3 [...] is not satisfied, then the convergence tolerances used by the solvers are reduced and the problem is solved again.*"

Benchmarking

- In the following classical paper on benchmarking with performance profiles

  Dolan, Elizabeth D. and Moré, Jorge J. and Munson, Todd S.
  Optimality Measures for Performance Profiles.
  SIAM Journal on Optimization, 16(3):891–909, jan. 2006.

  The authors set up an optimality criterion to compare non-linear optimization solvers.
- Ok, but what if the solver ends with a sucess but fails to satisfy the optimality measure?"*if the convergence test proposed in section 3 [...] is not satisfied, then the convergence tolerances used by the solvers are reduced and the problem is solved again.*"

Another evidence of the usefullness of the Stopping framework.

Stacking the Stoppings: the come back

Stacking the Stoppings: the come back

Recall from the beginning the GNEP

$$\left\{ \begin{array}{l} \min\limits_{x \in \mathbb{R}^{n_1}} \; f_1(x, y) \\ \text{s.t.} \;\; l_{c_1} \leq c_1(x, y) \leq u_{c_1} \end{array} \right\} \text{ and } \left\{ \begin{array}{l} \min\limits_{y \in \mathbb{R}^{n_2}} \; f_2(x, y) \\ \text{s.t.} \;\; l_{c_2} \leq c_2(x, y) \leq u_{c_2} \end{array} \right\}$$

Stacking the Stoppings: the come back

Recall from the beginning the GNEP

$$\left\{ \begin{array}{l} \min\limits_{x \in \mathbb{R}^{n_1}} f_1(x,y) \\ \text{s.t. } l_{c_1} \leq c_1(x,y) \leq u_{c_1} \end{array} \right\} \text{ and } \left\{ \begin{array}{l} \min\limits_{y \in \mathbb{R}^{n_2}} f_2(x,y) \\ \text{s.t. } l_{c_2} \leq c_2(x,y) \leq u_{c_2} \end{array} \right\}$$

How to design a Stopping for such more complicated problem?

Stacking the Stoppings: the come back

Recall from the beginning the GNEP

$$\left\{ \begin{array}{l} \min\limits_{x \in \mathbb{R}^{n_1}} f_1(x,y) \\ \text{s.t. } l_{c_1} \leq c_1(x,y) \leq u_{c_1} \end{array} \right\} \text{ and } \left\{ \begin{array}{l} \min\limits_{y \in \mathbb{R}^{n_2}} f_2(x,y) \\ \text{s.t. } l_{c_2} \leq c_2(x,y) \leq u_{c_2} \end{array} \right\}$$

How to design a Stopping for such more complicated problem?

1. The GNEP is a concatenation of NLP, therefore we can use each NLP-Stopping

Stacking the Stoppings: the come back

Recall from the beginning the GNEP

$$\left\{ \begin{array}{l} \min\limits_{x \in \mathbb{R}^{n_1}} \; f_1(x,y) \\ \text{s.t.} \;\; l_{c_1} \leq c_1(x,y) \leq u_{c_1} \end{array} \right\} \text{ and } \left\{ \begin{array}{l} \min\limits_{y \in \mathbb{R}^{n_2}} \; f_2(x,y) \\ \text{s.t.} \;\; l_{c_2} \leq c_2(x,y) \leq u_{c_2} \end{array} \right\}$$

How to design a Stopping for such more complicated problem?

1. The GNEP is a concatenation of NLP, therefore we can use each NLP-Stopping
2. stop!(GNEP-Stopping) is now just a loop on the stop!(NLP-Stopping)

The Crazy Unexpected Stopping Criterion

The Crazy Unexpected Stopping Criterion

- We have already seen from the **active set algorithm** the case of a stopping criterion that was unexpected for an unconstrained problem

The Crazy Unexpected Stopping Criterion

- We have already seen from the **active set algorithm** the case of a stopping criterion that was unexpected for an unconstrained problem
- Some applications may bring some unexpected criterion like:

$$\text{max\_time} \leq 1\text{hour } \textbf{or } f(x) \leq 50$$

The Crazy Unexpected Stopping Criterion

- We have already seen from the **active set algorithm** the case of a stopping criterion that was unexpected for an unconstrained problem
- Some applications may bring some unexpected criterion like:

$$\text{max\_time} \leq 1\text{hour } \textbf{or } f(x) \leq 50$$

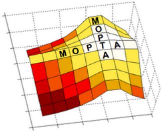
In this case, the Stopping framework eases the use of your algorithm and makes it more flexible.

Thank you for your attention !

This August in Lehigh: Optimization, Games ...

Important Dates

- **MOPTA** 14-16 August, 2019
- **Abstract Deadline** June 30, 2019
- **Registration Deadline** August 10, 2019 (on site after that)
- **Early Registration Deadline** June 30, 2019
  All speakers must register by this date.
- **Competition Deadline** Sunday May 19, 2019, 23:59 Pacific time



spread over three days. Our target is to present a diverse set of exciting new developments from different optimization areas w the same time providing a setting which will allow increased interaction among the participants. We aim to bring together rese from both the theoretical and applied communities who do not usually have the chance to interact in the framework of a medi event.

Plenary speakers


Boris Mordukhovich


Natalia Alexandrov


Tamer Basar


Fernando Brandão


Martin Grötschel


Antonio Conejo


Alexander Shapiro