OPTool - Documentation

Daniel Silvestre

Contact:
dsilvestre@isr.tecnico.ulisboa.pt
May 05, 2019

Abstract

The OPTool package is an implementation of various state-of-the-art iterative optimization algorithms for differentiable cost functions along with algorithms to solve linear equations. Users can use the toolbox to solve optimization problems, although the code was written to researchers that want to compare their proposals with state-of-the-art implementation. New algorithms can be easily added and the software will be updated to have the most comprehensive list of solvers possible. It also comes with implemented functions to return optimal parameters for these algorithms based on a control-theoretical formulation of the algorithms.

Keywords: Optimization Problems, Control-theoretical Formalization, Gradient-descent-like Algorithms

1. Introduction and background

The package OPTool has various algorithms to solve optimization problems where x denote the variable and for a general cost function f written as:

$$\underset{x}{\text{minimize}} \quad f(x) \tag{1}$$

Function f is assumed differentiable, i.e., there exists ∇f, so subgradient methods are currently not implemented. There exist multiple gradient-descent algorithms and we use two in this tutorial to help the user successfully run and understand its first example. The steepest gradient descent labeled as Gradient and the Nesterov method are:

$$\begin{aligned} &\textsc{Gradient}: \ x^{(k+1)} = x^{(k)} - \beta \nabla f(x^{(k)}) \\ &\textsc{Nesterov}: \begin{cases} x^{(k+1)} = \xi^{(k)} - \beta \nabla f(\xi^{(k)}) \\ \xi^{(k)} = (1+\gamma) x^{(k)} - \gamma x^{(k-1)} \end{cases} \end{aligned} \tag{2}$$

with β, γ selected as parameters.

The aforementioned methods are general to any function f and, given appropriate choice for parameters, they converge to the global minimum provided f is convex. Whenever this is not the case, convergence occurs for on of the points where $\nabla f = 0$, which can be a minimum, maximum or saddle point.

If we specialize function f to be quadratic, i.e., $f = \frac{1}{2}\|Mx - b\|_2^2$, the algorithms are linear as ∇f is a linear function of x. Therefore, iterative algorithms to solve linear equations can also be applied to the equation $Mx = b$.

OPTool implements many methods that are present in the literature. See the README file for the complete list.

2. Installation

To install simply download the zip file and extract it to the Matlab folder and add it to the path.

3. Usage

To use the package to solve a general optimization algorithm, one can use the function:

```
function [ stateVectors , errors ] = optSolver(
    algorithms , parameters , grad , errorFunction,
    max_iterations , initialState, projectionFunction
    , tol)
```

With inputs:

algorithms a cell array containing pointers to functions implementing the next iteration of the algorithms;

parameters a cell array containing structures with the parameters for each algorithm in **algorithms**;

grad a pointer to the gradient function;

errorFunction a pointer to the function that computes the error for a particular x;

max_iterations maximum allowed number of iterations to achieve the solution;

initialState initial guess for the minimum of f;

projectionFunction optional input containing a pointer to a function that projects the state onto some constraining set;

tol tolerance before the algorithm halts and sets all subsequent time steps equal to the current one.

and outputs:

stateVectors a cell array containing the matrices $\begin{bmatrix} x(0) & \cdots & x(\text{max\_iterations}) \end{bmatrix}$ for each of the selected algorithms;

errors cell array containing the vectors of errors for each of the algorithms using **errorFunction**.

Similarly, the function:

```
function [ stateVectors , errors ] = linSolver(
   algorithms , parameters , A , b , errorFunction,
   max_iterations , initialState, projectionFunction
   , tol)
```

solves the linear equation $Ax = b$. Instead of providing the gradient function, the user must supply A and b and all the rest of the inputs are equal.

In case the optimization problem is quadratic, the user can call function:

```
function [ stateVectors , errors ] = quadSolver(
   algorithmNames , parameters , A , b ,
   errorFunction, max_iterations , initialState ,
   experimentName, projectionFunction, tol,
   errorDescription)
```

giving the additional **experimentName** that will be used to save the resulting plots and variables in the folder *Stored Outputs*.

To facilitate the setup of the problems to be computed, inside the folder *Optimal Parameters* there is the function:

```
function [algorithms, parameters, names] =
   getParameters(A, b, r, methods)
```

With inputs:

A a matrix such that $Q = A'A$;

b a vector such that $p = A'b$;

r a value such that $f(x) = 0.5x^\intercal Qx - p^\intercal x + r$;

methods a string array with all names of desired algorithms to solve the problem.

and outputs:

algorithms a structure containing **optimization** that is the **algorithms** input to **optSolver** and **linearEquations** which should be used for **linSolver**;

parameters a structure containing **optimization** that is the **parameters** input to **optSolver** and **linearEquations** which should be used for **linSolver**;

names a sorted string array of the names of the selected methods present in the above data structures (badly specified methods will not appear in this output).

If this function is used for a non-quadratic problem, matrix A should be such that its minimum and maximum eigenvalues are the smoothness and strong convexity values of the function. For a list of the available methods, just run "getParameters" with no inputs or to select all "getParameters(A, b, r, 'all ')".

4. Implementing additional algorithms

The OPTOOL was implemented such that adding other algorithms to the package is straightforward. If a NOVELALGORITHM needs to be added, the developer must:

1. Define the NOVELALGORITHM function that implements a single iteration of the proposed algorithm. The function should follow the definition:

```
function [ x , parameters] = novelAlgorithm( x ,
    previous_x , grad, parameters)
```

With inputs:

x current vector estimation of the minimum of f;

previous_x previous vector estimation of the minimum of f;

grad pointer to the gradient function;

parameters structure used to pass parameters between algorithm iterations.

and outputs:

x new vector estimation of the minimum of f;

parameters updated structure of the parameters for this algorithm.

2. Store the above file in either folder *Optimization Algorithms* if it is a general gradient descent algorithm or in *Linear Equation Solver* if it solves $Ax = b$;
3. Add reference to the paper defining the algorithm in *References*;
4. In the file *getParameters.m* add the correspondence between name of the algorithm and the typical name of the parameters either in the variable **optAlgorithmNames** or **linEqAlgorithmNames** according to the type of algorithm;
5. Still in the file *getParameters.m* add an if clause similar to the one presented for the gradient descent:

```
%==== Optimization Algorithms ====
% Gradient Descent
if strcmp(methods(i),"Gradient Descent") ||
   allMethods
optAlgorithms{optIndex} = @gradientDescent;
optParameters{optIndex} = struct('alpha',2/(L +
   m));
optIndex = optIndex + 1;
end
```

for example:

```
%==== Optimization Algorithms ====
% Gradient Descent
if strcmp(methods(i),"Gradient Descent") ||
   allMethods
optAlgorithms{optIndex} = @gradientDescent;
optParameters{optIndex} = struct('alpha',2/(L +
   m));
optIndex = optIndex + 1;
end
% Novel Algorithm
if strcmp(methods(i),"Novel Algorithm") ||
   allMethods
optAlgorithms{optIndex} = @novelAlgorithm;
optParameters{optIndex} = struct('myParameter',<
   myparameterValue>);
```

```
12 optIndex = optIndex + 1;
13 end
```

5. Illustrative Example

5.1. PageRank

The following example comes in file *PageRank.m*. The PageRank problem consists of a ranking mechanism from Google, which was initially proposed in [1]. It corresponds to finding the eigenvector of the following matrix $M \in \mathbb{R}^{n\times n}$:

$$M := (1-m)A + \frac{m}{n}S \tag{3}$$

where $m \in (0,1)$ is a parameter defining the convex combination of the adjacency matrix A of the network with the matrix $S := \mathbf{1}_n\mathbf{1}_n^\intercal$ ($\mathbf{1}_n$ is the n-dimensional vector of ones). A typical choice is $m = 0.15$ [1]. The standard formulation can be efficiently computed through the power method:

$$x(k+1) = Mx(k) = (1-m)Ax(k) + \frac{m}{n}\mathbf{1}_n \tag{4}$$

where $x(k) \in \mathbb{R}^n$ and $\forall k \geq 0 : \mathbf{1}_n^\intercal x(k) = 1$.

The PageRank problem can also be formulated as an optimization problem or as the solution to a linear equation. In the former, the PageRank is the solution to the following optimization problem

$$\underset{x}{\text{minimize}} \quad \frac{1}{2}\|((1-m)A - I_n)x + \frac{m}{n}\mathbf{1}_n\|_2^2$$

If seen as the solution of a linear equation in matrix format we get:

$$(I_n - (1-m)A)x = \frac{m}{n}\mathbf{1}_n. \tag{5}$$

In [2], it is shown that the standard power method for the PageRank is equivalent to the Jacobi method applied to (5). The solution of the PageRank for a random Barabási–Albert generated network is included in the OPTOOL and the plot of the errors produced by the toolbox is given in Figure 1, where it is shown better alternatives to the PageRank if the optimal parameters are known by each node.

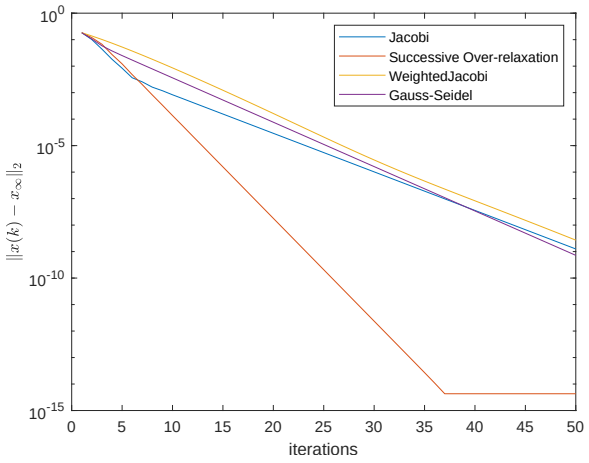

Figure 1: The error evolution for each of the tested algorithm in the PageRank case for a 20-node network.

5.2. Desynchronization in Decentralized Medium Access

The example of the distributed desynchronization in Decentralized Medium Access control comes in the file *Desync.m*. In the literature following the Pulsed-Couple Oscillators (PCO) model. In this framework, nodes form a ring network where each agent broadcasts periodically a *fire message* or a *pulse*. Such dynamics is modeled by a *phase* variable $\theta_i(t)$ for each node $i \in \{1, \cdots, n\}$

$$\theta_i(t) = \frac{t}{T} + \phi_i(t) \mod 1, \tag{6}$$

where $\phi_i \in [0, 1]$ is the so called phase offset of node i and mod represents the modulo arithmetics. The idea behind (6) is to consider the phase going from zero to one along a circle. Every node i broadcasts a pulse when its phase reaches the unity (i.e., every T time units) and then resets it to zero. When the nodes listen to other nodes pulses, they adjust their ϕ variable according to an update equation based on the PCO dynamics. The algorithm would then update the phase after receiving the beacons from node $i-1$

$$\theta_i'(t_{i-1}) = (1-\alpha)\theta_i(t_{i-1}) + \alpha\frac{\theta_{i-1}(t_{i-1}) + \theta_{i+1}(t_{i-1})}{2} \tag{7}$$

where t_{i-1} is the time instant at which fire message from node $i-1$ was received by node i and assuming we consider the nodes to be placed on a circle such that node 1 and n are neighbors. The *jump-phase parameter* $\alpha \in (0, 1)$ translates how much node i changes its phase in response to the phase of its neighbors.

Following a slight modification where node n updates its phase at update cycle k using $\theta_{n-1}^{(k-1)}$ instead of $\theta_{n-1}^{(k)}$, the algorithm is defined by:

$$\begin{aligned}
\phi_1^{(k)} &= (1-\alpha)\phi_1^{(k-1)} + \frac{\alpha}{2}\left(\phi_2^{(k-1)} + \phi_n^{(k-1)} - 1\right) \\
\phi_i^{(k)} &= (1-\alpha)\phi_i^{(k-1)} + \frac{\alpha}{2}\left(\phi_{i-1}^{(k-1)} + \phi_{i+1}^{(k-1)}\right), 2 \le i \le n-1 \\
\phi_n^{(k)} &= (1-\alpha)\phi_n^{(k-1)} + \frac{\alpha}{2}\left(\phi_{n-1}^{(k-1)} + \phi_1^{(k-1)} + 1\right)
\end{aligned} \tag{8}$$

which is equivalent to the steepest descent algorithm applied to

$$\underset{\phi}{\text{minimize}} \quad g(\phi) := \frac{1}{2}\|D\phi - v\mathbf{1}_n + \mathsf{e}_n\|_2^2 \tag{9}$$

where $v = 1/n$, $\mathbf{1}_n$ is the vector of ones, $\mathsf{e}_n = (0, 0, \cdots, 0, 1)$, and

$$D = \begin{bmatrix} -1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & & \ddots & & \vdots \\ 0 & \cdots & 0 & 0 & -1 & 1 \\ 1 & \cdots & 0 & 0 & 0 & -1 \end{bmatrix}. \tag{10}$$

Specifically, the updates in (8) can be written as

$$\phi^{(k)} = \phi^{(k-1)} - \frac{\alpha}{2}\nabla g(\phi^{(k-1)}). \tag{11}$$

If the Gauss-Seidel method is used, it results in the update:

$$\begin{aligned}
\phi_1^{(k+1)} &= \frac{1}{2}\left(1 - \phi_2^{(k)} - \phi_n^{(k)}\right) \\
\phi_i^{(k+1)} &= \frac{1}{2}\left(-\phi_{i-1}^{(k+1)} - \phi_{i+1}^{(k)}\right), 2 \le i \le n-1 \\
\phi_n^{(k)} &= \frac{1}{2}\left(-1 - \phi_1^{(k+1)} - \phi_{n-1}^{(k+1)}\right)
\end{aligned} \tag{12}$$

which requires communication with the immediate neighbors akin the original problem and exploits the inherent sequential behavior of the DESYNC algorithm to have nodes using the most updated values for the phases.

Using the toolbox, it is possible to produce the plot in Figure 2 that presents the error evolution for the PCO-based (Gradient Descent), Nesterov, LTV Nesterov, Heavy-Ball and Gauss-Seidel algorithms for a 6-node network.

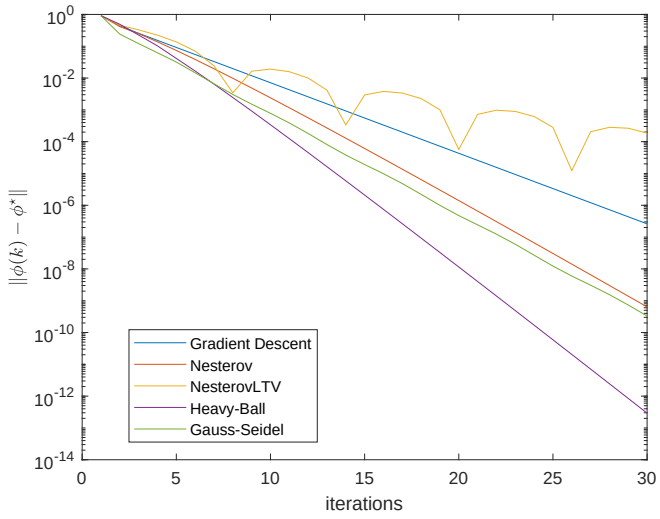

Figure 2: Logarithmic evolution of the error norm for the PCO-based (Gradient Descent), Nesterov, LTV Nesterov, Heavy-Ball and Gauss-Seidel algorithms for a 6 node network.

6. Plans for future releases

In future versions, we expect to add subgradient methods and also data-based training such as the Stochastic Gradient Descent. It is also going to be added work in progress on novel algorithms for optimization and other convergence rates calculations.

We also would like to make available additional examples of paper [3] and other works under development on power networks. Topics such as consenus, both deterministic and stochastic [4], [5].

References

[1] S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine, Computer Networks and ISDN Systems 30 (1) (1998) 107 – 117. doi:http://dx.doi.org/10.1016/S0169-7552(98)00110-X.

[2] D. Silvestre, J. Hespanha, C. Silvestre, A pagerank algorithm based on asynchronous gauss-seidel iterations, in: 2018 Annual American Control Conference (ACC), 2018, pp. 484–489. doi:10.23919/ACC.2018.8431212.

[3] D. Silvestre, J. Hespanha, C. Silvestre, Desynchronization for decentralized medium access control based on gauss-seidel iterations, in: 2019 Annual American Control Conference (ACC), 2019, pp. 4049–4054.

[4] D. Antunes, D. Silvestre, C. Silvestre, Average consensus and gossip algorithms in networks with stochastic asymmetric communications, in: 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC), 2011, pp. 2088–2093. doi:10.1109/CDC.2011.6161444.

[5] D. Silvestre, J. P. Hespanha, C. Silvestre, Broadcast and gossip stochastic average consensus algorithms in directed topologies, IEEE Transactions on Control of Network Systems (2018) 1–1doi:10.1109/TCNS.2018.2839341.