SoftwareX 11 (2020) 100371

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

OPTool—An optimization toolbox for iterative algorithms

Daniel Silvestre

Department of Electrical and Computer Engineering of the Faculty of Science and Technology of the University of Macau, Macau, China
Institute for Systems and Robotics (ISR) from the University of Lisbon, Av. Rovisco Pais, 1, 1049-001 Lisboa, Portugal

ARTICLE INFO

Article history:
Received 31 July 2019
Received in revised form 28 October 2019
Accepted 26 November 2019

Keywords:
Optimization problems
Control-theoretical formalization
Gradient-descent-like algorithms

ABSTRACT

The current state-of-the-art in iterative optimization algorithms for differentiable cost functions is scattered throughout the literature, which hinders their comparison for the specific program at hand. Depending on the research area, theoretical optimal parameters and convergence rates are available in different formulations. Consequently, this toolbox aims at providing a benchmarking software for the various gradient-descent-based algorithms and implements functions to return the optimal parameters whenever possible. Researchers can focus on the development of new algorithms and test them against the ones present in the literature and made them available under a common framework. The structure of the software is tailored such that novel contributions can be easily added through the design of a function implementing a single step of the algorithm. The ease of usage is illustrated by examples in the literature.

Code metadata

Current code version	v1.2
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2019_250
Legal Code License	GNU GPLv3
Code versioning system used	none
Software code languages, tools, and services used	Matlab
Compilation requirements, operating environments & dependencies	Just place the folder in Matlab path
If available Link to developer documentation/manual	https://github.com/danielmsilvestre/OPTool/docs
Support email for questions	dsilvestre@isr.tecnico.ulisboa.pt

1. Motivation and significance

Numerous problems in engineering can be tackled by optimization techniques. Tasks such as robot path planning, image processing and collision detection can be achieved through the minimization of a function subject to constraints using, for example, the work in [1]. Computing the PageRank as in [2] or achieving desynchronization in access protocols in wireless sensor networks [3] can also be seen as the solution of an optimization.

In the literature, there are a number of possible algorithms and highly-efficient solvers implemented from which to choose. Selecting the appropriate one for a specific problem is a hard task as they are not easily compared. For example, in [4], it is proposed a toolbox to help the user select which optimization algorithm should be used depending on the characteristics of the cost functions and constraints. Most of this algorithms use step size parameters or weights in some operations. The aforementioned software does not provide a means compare algorithms performance and select optimal values for these parameters. Therefore, having software that can serve as a benchmarking tool is of prime importance. In [5], the authors have introduced a software following the same principle for feature selection algorithms. The code structure is designed in a similar fashion but the set of target problems is different (feature selection vs minimizing functions) and, as a consequence, the implemented algorithms. Complementing this work, the contributions of this paper can be summarized as:

E-mail address: dsilvestre@isr.tecnico.ulisboa.pt.

https://doi.org/10.1016/j.softx.2019.100371

- a toolbox implementation the most common algorithms to make an easier comparison of their complexity;
- optimal parameter computation based on control theoretical results.

In the remainder of this section, we introduce the main notation and scientific background of the topic. The interested reader is referred to [2,3] for references regarding recent advances in the design of optimization algorithms such as using Integral Quadratic Constraints (IQCs) in [6]. Let x denote the variables describing the problem and a cost function f measuring the quality of the solution x. We would like to select $x^\star$ such that the value of $f(x^\star)$ is the minimum over all possible values. The optimization is written as:

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

In this paper, we assume that function f is differentiable, i.e., there exists ∇f. In such cases, common algorithms to obtain distributed solutions, i.e., where parts of the state x are divided over different processing units and communicated to their neighbors, follow the idea of gradient descent with one of the most efficient being the Nesterov method:

$$\begin{aligned} \text{GRADIENT}: \quad & x^{(k+1)} = x^{(k)} - \beta \nabla f(x^{(k)}) \\ \text{NESTEROV}: \quad & x^{(k+1)} = \xi^{(k)} - \beta \nabla f(\xi^{(k)}) \\ & \xi^{(k)} = (1+\gamma)x^{(k)} - \gamma x^{(k-1)} \end{aligned} \tag{2}$$

with β, γ being parameters. In (2), $x^{(k)}$ is the estimate at iteration k, $\nabla f(x)$ is the gradient of f at the point x, and ξ is an internal variable computing the momentum term. The GRADIENT algorithm is the simplest version where the new estimate corresponds to a step along the direction of steepest descent $-\nabla f(x^{(k)})$ and a step size of β. The NESTEROV method adds a momentum term given by the difference between the two previous iterations. The other implemented algorithms introduce variants on the idea of following the descent direction given by the gradient.

In essence, the convergence and its speed depend on the values selected for β, γ. Under a suitable choice of parameters, the algorithms converge to the global minimum provided f is convex. Whenever this is not the case, convergence occurs for one 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., $f(x) = \frac{1}{2}\|Mx - b\|_2^2$, the algorithms are linear as ∇f is a linear function of x. Addressing the problem as the solution of equation $\nabla f(x) = 0$, results in the possibility to use iterative algorithms to solve linear equations of the form:

$$Ax = b \tag{3}$$

For a general function f, one can adopt a nonlinear equation solver such the one implemented in [7].

The next section details the organization, algorithms and main functions implemented in OPTOOL.

2. Software description

The OPTOOL toolbox implements a solver that can run a collection of iterative algorithm provided the user supplies a function to obtain the next iteration given the current one. This is accomplished by passing a cell array of all function handles the user would like to compare in the simulation. As an example, if the user is interested in running the gradient descent and Nesterov method in (2), a cell array containing these two function handles will be passed to the solver. OPTOOL currently implements other methods that can be found in the literature. In the category of optimization solvers: gradient descent, Heavy-ball, Nesterov, Momentum, Fast Iterative Shrinkage-Thresholding Algorithm (FISTA), Descent Fast Iterative Soft-thresholding Algorithm (DFISTA), Second Nesterov, Barzilai–Borwein, Random descent, Cauchy–Barzilai–Borwein and General Barzilai–Borwein. In the linear equation solvers, it implements: Jacobi, Weighted Jacobi, Gauss–Seidel, Successive Over-relaxation (SOR), Richardson, Conjugate Gradient, Biconjugate Gradient, Newton–Raphson, Sparse Broyden, Broyden, Bad Broyden, Delayed Over-relaxation (DOR), Minimal Residual DOR, Accelerated Over-relaxation (AOR), Practical Asymptotical Optimal SOR (PAOSOR), Alternating Anderson–Jacobi, Chebyshev, Quasi-Chebyshev, HSS iteration method, Kaczmarz, Coordinate Descent, Conjugate Gradient to the Normal Equation (CGNE) and Improved Biconjugate Gradient (IBiCG). Please see the package documentation for references and additional information [8].

Additionally, since most methods, such as the ones in (2), have parameters such as β and γ that account for weights to be designed. To this end, the user should supply a cell array of structures containing all these values to the solver. Alternatively, these data structures can be returned by the function *getparameters(A,b,r,methods)* that is specifically tailored for quadratic problems of the form $f(x) = 0.5x^\intercal Qx - p^\intercal x + r$, where $Q = A^\intercal A$ and $p = A^\intercal b$. If one was simulating both methods in (2), one of the entries would have a structure with a single parameter β for the gradient descent, whereas the second entry of the array would contain a structure with β and γ for the Nesterov method. Having separated the computation of the parameters from the solver task enables users to pass their own data structures to functions *optSolver* or *linSolver* or access and change the suggested parameters given by the function *getparameters*. The variable *methods* contains all strings identifying the algorithms for which the user wants to compute the parameters. The software also comes with a *quadSolver* function that performs the simulations and stores the results in files thus saving the need to call both *optSolver* or *linSolver* if the user is addressing a quadratic problem with algorithms of both classes.

2.1. Software architecture

In this implementation, the main objective was to produce a code as simple and scalable as possible for other researchers to implement and test their algorithms. Therefore, the architecture for the software is composed of a Solver (either of linear equations of optimization) that receives: simulation variables such as maximum number of iterations, initial state, tolerance before stopping, etc.; parameters of the algorithms and function handlers to the functions implementing one iteration. The software architecture is depicted in Fig. 1.

The addition of a new algorithm requires two steps: (i) implementing a function returning the next value approximating the solution, and (ii) adding to the *getParameters* an if-clause similar to the ones for the implemented algorithms, where it is given the function handler from (i) and how to obtain the optimal parameters for the algorithm. No other changes are required, and thereby future developments are facilitated. Since the solvers already include the possibility to perform projections or apply a general function to the next approximation value, researchers have a further degree of freedom in the behavior of their algorithms and can apply the same transformation to all simulated methods without having to correct the code for the function providing the next iteration of each algorithm.

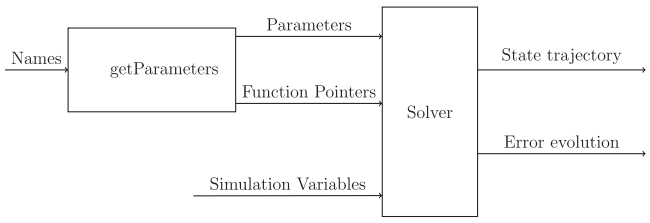

Fig. 1. OPTool architecture.

2.2. Software functionalities

The OPTool software is composed of three main solver functions: *linSolver*, *optSolver*, and *quadSolver*. The first two run algorithms for the solution of linear equations and general optimization algorithms, respectively. They both accept a vector of function handlers implementing the next iteration of each algorithm, along with variables describing the problem to be solved and other simulation constants. The function *quadSolver* aims to call both *linSolver* and *optSolver* for the same quadratic problem.

In the package, two folders *Linear Equation Solvers* and *Optimization Algorithms* store all the functions for each of the implemented algorithms. If the user wants to expand the software to its own designed algorithm, he/she needs to implement a function that provides the next estimation of $x^\star$.

A major contribution of this software is the computation of the optimal parameters for each of the algorithms whenever the choice of parameters allows to view them as LTI systems. To that end, there is a function *getParameters* within the folder *Optimal Parameters* that retrieves the data structures needed for *linSolver* and *optSolver* as well as invokes the specific function for each algorithm that returns its optimal parameters. Some of these are direct implementation from the expressions found in the literature whereas the others explore the LTI view.

Lastly, a folder of *Auxiliary Functions* implements methods to generate adjacency matrices for scale free corresponding network topology, projection on the simplex and normalization to one (used in the PageRank example) and functions to randomly select updating nodes based on a prescribed probability vector or compute the spectral radius. The network generation follows the Barabási–Albert model that is useful to test the PageRank algorithm against a network resembling the characteristics of the Internet.

3. Illustrative examples

A search engine is a tool that allows users to supply a query and to obtain web pages related to the information they are searching for. A crucial task for these tools is some sort of ranking mechanism that selects meaningful links to be presented first. One of the most well-known algorithms is the PageRank from Google, which was initially proposed in [9], to rank pages based on their relative importance and on the number of links to each specific page. In [9], it is proposed that the ranking be the eigenvector of the following matrix $M \in \mathbb{R}^{n\times n}$:

$$M := (1-m)A + \frac{m}{n}S \tag{4}$$

where $m \in (0, 1)$ is a parameter defining the convex combination of the adjacency matrix A with the matrix $S := 1_n 1_n^\intercal$ (1_n is the n-dimensional vector of ones). A typical choice is $m = 0.15$ [9]. The standard formulation can be efficiently computed through the power method:

$$x(k+1) = Mx(k) = (1-m)Ax(k) + \frac{m}{n}1_n \tag{5}$$

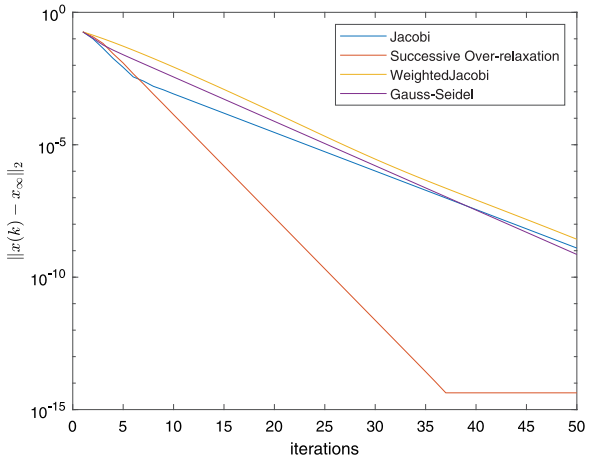

Fig. 2. The error evolution for each of the tested algorithm in the PageRank case for a 20-node network.

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

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

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

which corresponds to the format in (1).

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

$$(I_n - (1-m)A)x = \frac{m}{n}1_n \tag{6}$$

corresponding to the linear equation view in (3).

In [2], it is shown that the standard power method for the PageRank is equivalent to the Jacobi method applied to (6). 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 Fig. 2, where it is shown better alternatives to the PageRank if the optimal parameters are known by each node.

The desynchronization of transmitters to achieve a fair multiple access schedule at the Medium Access Control (MAC) layer in the context of Wireless Sensor Networks (WSNs) can be done, for example, using the Time-Synchronized Channel Hoping (TSCH) [10]. The problem can be formulated [3] in the format of (1):

$$\underset{\phi}{\text{minimize}} \quad f(\phi) := \frac{1}{2}\|D\phi - v1_n + \mathrm{e}_n\|_2^2 \tag{7}$$

where ϕ is the phase offset of each transmitter, $v = 1/n$, 1_n is the vector of ones, $\mathrm{e}_n = (0, 0, \ldots, 0, 1)$, and

$$D = \begin{bmatrix} -1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & & \ddots & & \vdots \\ 0 & \cdots & 0 & 0 & -1 & 1 \\ 1 & \cdots & 0 & 0 & 0 & -1 \end{bmatrix}. \tag{8}$$

In this case, the linear equation view in (3) produces poor results given the very specific structure of D. Using OPTool, in [3] it is possible to benchmark the gradient descent (equivalent to a standard algorithm referred to as Pulsed-Couple Oscillators (PCOs)), against the Nesterov and Gauss–Seidel algorithms and plot their errors as a function of iteration number, depicted in Fig. 3.

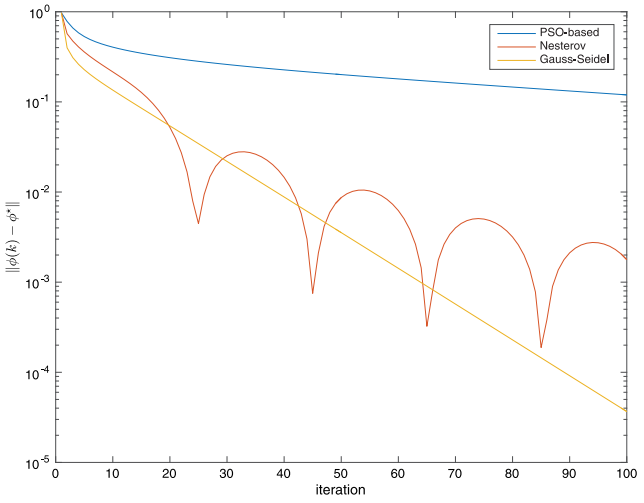

Fig. 3. Logarithmic evolution of the error norm for the PCO-based, Nesterov and Gauss–Seidel algorithms for the 20 node network.

4. Impact

As seen from Section 3, the OPTool package allows to pose questions as whether a current state-of-the-art algorithm is the best to solve a given problem and compare other solutions. Additionally, it motivates the study of these algorithms as LTI systems and brings other techniques from this area to the design of optimization algorithms, all of which can be simulated by simply implementing a function giving the next iteration and another for how to select the parameters. The software has already been used for two problems: PageRank in [2] and Desynchronization in Medium Access Control in [3]. Many other problems ranging from power networks to image processing are possible future work.

Currently there are focus on studying fast first-order optimization algorithms and also their resilience to noise. These cases can be tested in the toolbox with minor changes by appropriately adding the next iteration functions. Since the software is in its first version, its usage has been rather limited with researchers implementing their own simulations with no sharing of a common platform. This toolbox aims at bringing together implementation practices in the community.

5. Conclusions

In this paper, a Matlab toolbox OPTool was presented that implements iterative solvers using linear equation and optimization algorithms. The main focus was on creating a platform that is scalable to new algorithm implementation by dissociating the solver code from the function implementing the next iteration. We have shown that these methods share a common interpretation that motivated this toolbox by seeing them as LTI or LTV systems from control theory where the input is the gradient of the function to be optimized. Resorting to standard techniques from the linear systems theory, it is possible to select optimal parameters and those have been added to the toolbox.

Two publication have already used this toolbox to produce their simulations and new research is being conducted in power networks that will also benefit from the software. There is interest from two different communities – control and optimization – and is expected that OPTool can accelerate research in this topic by facilitating an easy alternative (the other being each individual implementing the desired algorithms from scratch) to test algorithms and compute theoretical convergence rates.

The case of PageRank from Google is included as an example where the software compares four linear equation algorithms to show that, if optimal parameters are known, Successive Over-Relaxation outperforms the Jacobi method that is equivalent to the power method currently employed as the state-of-the-art algorithm to solve the ranking problem.

Future work will focus the interesting problem of optimizing the initial estimate to achieve a faster convergence. Various possibilities are going to be explored, namely computing the norm of the LTI model executing the algorithm as a function of the initial state.

Acknowledgments

D. Silvestre was supported by the project MYRG2016-00097-FST from the University of Macau, and by the Portuguese Fundação para a Ciência e a Tecnologia (FCT) through Institute for Systems and Robotics (ISR), under Laboratory for Robotics and Engineering Systems (LARSyS) project UID/EEA/50009/2019.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Montanari M, Petrinic N. OpenGJK for C, C♯ and Matlab: Reliable solutions to distance queries between convex bodies in three-dimensional space. SoftwareX 2018;7:352–5. http://dx.doi.org/10.1016/j.softx.2018.10.002.

[2] Silvestre D, Hespanha J, Silvestre C. A PageRank Algorithm based on Asynchronous Gauss-Seidel Iterations. In: 2018 Annual American Control Conference (ACC). 2018, p. 484–9. http://dx.doi.org/10.23919/ACC.2018.8431212.

[3] Silvestre D, Hespanha J, Silvestre C. Desynchronization for Decentralized Medium Access Control based on Gauss-Seidel Iterations. In: 2019 American Control Conference (ACC). 2019, p. 4049–54. http://dx.doi.org/10.23919/ACC.2019.8814471.

[4] Popova O, Popov B, Romanov D, Evseeva M. Optimel: Software for selecting the optimal method. SoftwareX 2017;6:231–6. http://dx.doi.org/10.1016/j.softx.2017.08.001.

[5] Reis MS, Estrela G, Ferreira CE, Barrera J. Featsel: A framework for benchmarking of feature selection algorithms and cost functions. SoftwareX 2017;6:193–7. http://dx.doi.org/10.1016/j.softx.2017.07.005.

[6] Lessard L, Recht B, Packard A. Analysis and design of optimization algorithms via integral quadratic constraints. SIAM J Optim 2016;26(1):57–95. http://dx.doi.org/10.1137/15M1009597.

[7] Scheffel J, Lindvall K. SIR—AN efficient solver for systems of equations. SoftwareX 2018;7:59–62. http://dx.doi.org/10.1016/j.softx.2018.01.003.

[8] Silvestre D. OPTool - documentation v1.2. https://github.com/danielmsilvestre/OPTool/blob/master/docs/optool-documentation.pdf.

[9] Brin S, Page L. The anatomy of a large-scale hypertextual web search engine. Comput Netw ISDN Syst 1998;30(1):107–17. http://dx.doi.org/10.1016/S0169-7552(98)00110-X.

[10] Tinka A, Watteyne T, Pister K. A decentralized scheduling algorithm for time synchronized channel hopping. In: Zheng J, Simplot-Ryl D, Leung VCM, editors. Ad Hoc Networks. Berlin, Heidelberg: Springer Berlin Heidelberg; 2010, p. 201–16.