Software Tools for Nonlinear Optimization — Modern Solvers and Toolboxes for Robotics —

Article · January 2014

DOI: 10.7210/jrsj.32.536

CITATION
1

READS
237

3 authors:

Thomas Alexandre Moulard
Google Inc.
22 PUBLICATIONS **96** CITATIONS
SEE PROFILE

Benjamin Chrétien
Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier…
7 PUBLICATIONS **30** CITATIONS
SEE PROFILE

Eiichi Yoshida
National Institute of Advanced Industrial Science and Technology
279 PUBLICATIONS **5,054** CITATIONS
SEE PROFILE

解 説

Software Tools for Nonlinear Optimization
—Modern Solvers and Toolboxes for Robotics—

Thomas Moulard[*1] Benjamin Chrétien[*2] Eiichi Yoshida[*1]
[*1]CNRS-AIST JRL (Joint Robotics Laboratory), UMI3218/CRT [*2]CNRS-UM2 LIRMM UMR 5506, Interactive Digital Human group

This article reviews the state-of-the-art of nonlinear solvers as well as frameworks for numerical optimization, which is more and more utilized for robotics applications. We will discuss the features and project status for each solver and detail how one can use a numerical optimization framework to avoid being limited to a particular solver. The comparison allows to choose the appropriate strategy in robotics where trajectory generation, posture generation, control can be implemented as different types of optimization problems.

1. Introduction

Various problems in robotics are heavily relying on numerical optimization. However, implementing these algorithms requires a high level of expertise and also is error-prone and time consuming. Both the required theoretical background and the implementation issues such as numerical precision and parameter tuning makes the task of writing a new solver challenging. Therefore, this paper will introduce various libraries implementing solvers for constrained nonlinear optimization problems in C/C++ or Python. Linear and quadratic problems are, of course, of interest but as they are easier to implement, finding the right tool is nowadays straightforward. C and C++ languages have been chosen due to their popularity in intensive computation tasks such as numerical optimization. On the other hand, Python is gaining in popularity due to its flexible syntax and its ease of use.

Once the decision of using a third-party solver is made, it is sometimes difficult to find the right toolbox. Robustness of the algorithm as well as the license and the activity of the project should be considered when one chooses an algorithm from available ones (e.g. Interior Point or Sequential Quadratic Problems).

This paper is reviewing various nonlinear solvers in 2 as well as numerical optimization frameworks in 3. Solvers are implementing one algorithm or one family of algorithms whereas frameworks are designed as toolboxes with which many solvers can be used. The 4 provides a benchmark of various solvers based on the result obtained in the RobOptim framework. Finally, 5 discusses the benchmark results and concludes.

2. State-of-the-Art Nonlinear Optimization Toolboxes

2.1 COIN IPOPT

The COIN-OR (COmputation INfrastructure for Operational Research) is releasing an open-source, nonlinear solver based on the interior point method called IPOPT [1].

This solver is written in C++, provides a full object-oriented interface and the website provides up-to-date documentation. The solver can solve nonlinear problems and ask the user to compute the functions values, gradients and optionally hessian. If Hessian computation is not provided, the solver will approximate it using a limited quasi-Newton algorithm (L-BFGS). The Jacobian and Hessian matrices are passed as sparse matrices, hence this solver is particularly suitable for large problems such as trajectory optimization. IPOPT relies on linear solvers for its interior-point method. It supports different linear solvers that can have a strong impact on the convergence and the optimization result. One could for instance use MUMPS [2] (open-source), or HSL's proprietary solvers (e.g. MA27, MA57, HSL_MA97) [3]. For large-scale problems, MUMPS can distribute computation over a cluster of machines using MPI, and some of the HSL solvers also support parallel computation.

原稿受付 2014 年 4 月 16 日
キーワード：Numerical Optimization, Software
[*1]〒 305–8568 つくば市梅園 1–1–1
[*2]Université Montpellier 2, LIRMM UMR 5506, 161 rue Ada, 34095 Montpellier, FRANCE
[*1]Tsukuba–shi, Ibaraki

Finally, IPOPT includes a limited support for warm start.

2.2 CFSQP

CFSQP [4] is a nonlinear solver written in C. Its interface is limited to a single C function requesting pointers to function to evaluate functions as well as Jacobians. As a result, its use is more complicated than other solvers from a technical point of view, but is largely compensated by a very thorough documentation. The main drawback of this solver is that it is a closed-source solver distributed by AEM Design Inc. This company's website is not available anymore and it is unknown if the distribution of this software will continue. This software supports multi-objective minimization and requires the developer to provide a mean to evaluate functions as well as Jacobians. Sparse problems as well as warm start are not supported.

2.3 NAG

NAG [5] is a commercial library including a large number of numerical algorithms, including numerical optimization algorithms. In particular, two nonlinear solvers are provided for both dense and sparse problems. It provides a C API and is a closed-source software which makes it more difficult to use. The documentation is good from a technical point of view, but it does not cover the full details of the underlying algorithms. The nonlinear solver is using the Sequential Quadratic Programming strategy (SQP). The solver supports warm start.

2.4 NLopt

NLopt is an open-source library dedicated to nonlinear optimization [6]. It includes several well-known optimization algorithms, either global (e.g. MLSL, StoGO), local derivative-free (e.g. BOBYQA) or local gradient-based (e.g. MMA [7], SLSQP) algorithms, as well as the Augmented Lagrangian algorithm [8].

This library aims at supporting different types of large-scale optimization problems: unconstrained, bound-constrained, or the general equality/inequality-constrained problem. Nonetheless, support for sparse matrices has not been added yet. NLopt can be interfaced with multiple languages: C/C++, Fortran, Matlab, Python, Julia etc.

2.5 PaGMO

PaGMO, which stands for Parallel Global Multiobjective Optimizer, is an open-source optimization toolbox developed by the ESA [9]. Even though this library was made with interplanetary trajectory optimization in mind, the methods can still be applied to robotics problems. Indeed, PaGMO provides an implementation of the Generalized Island Model [10], which allows the solver to distribute genetic algorithms over multiple processors. PaGMO can parallelize the workload on multiple local threads as well as on multiple machines thanks to MPI support. A Python interface, PyGMO, is also available.

Contrary to the other solvers presented here, PaGMO relies mostly on evolutionary algorithms to solve optimization problems, but it also supports nonlinear solvers such as GSL, IPOPT, NLopt or SNOPT. One of the advantages of its method is that it is less prone to entrapment by local minima. PaGMO offers a wide range of evolutionary or stochastic algorithms: Differential Evolution, Particle Swarm Optimization, Monotonic Basin Hopping, Ant Colony Optimization, Monte Carlo, etc.

2.6 Summary

Open-source software for numerical optimization is becoming increasingly common. However, nonlinear solvers remain a niche where using proprietary tools can remain a necessity. Open-source projects like NLopt or PaGMO are younger, yet promising projects, but even nowadays the older, proprietary solvers such as CFSQP keep the lead when it comes to small problems. IPOPT achieves intermediate results but it has been designed more for larger problems and thus is penalized when one has to deal with smaller, dense, problems. NAG is also a possible alternative with medium performances.

Moreover, when dealing with a specific problem, the theoretical best off-the-shelf solver may not be the one providing the best results. For instance, some problems may be easily solvable with SQP-based solvers while interior-point solvers will take more time to converge. These problem-specific considerations can be easily evaluated if the library used to implement the problems can handle multiple solvers. Frameworks as the one which will be introduced in the 3 try to answer this problem.

Nonlinear solvers still rely heavily on parameter tuning (tolerances, scaling, etc.), and the users may need to set these parameters according to their goal: quality of the solution, computation speed etc. Yet, the more complex the problem, the more unclear the influence of these parameters on the convergence.

Table 1 Nonlinear solver comparison

Solver	Project Status	License	Strategy	Features
IPOPT	active	open-source (EPL)	Interior Point	Warm Start, Sparse
CFSQP	inactive	closed-source	SQP	none
NAG	active	closed-source	SQP	Warm Start, Sparse
NLopt	active	open-source (LGPL)	Multiple	Global/Local
PaGMO	active	open-source (GPLv3)	Evolutionary	Global, Multithreading, MPI

The **Table 1** will summarize the advantages and drawbacks of every solver.

3. Numerical Optimization Frameworks

An efficient solver is the key to solve robotics problems in a timely manner. However, it is difficult to choose the right solver, and even harder to determine the required features or to formulate the optimization problem in an appropriate manner. In these scenarii, instead of relying on a particular solver, a more general "optimization framework" is advantageous because it allows us to change the resolution strategy after running tests on the full problem. Moreover, this kind of framework could offer a high abstraction level as well as an easy way to implement the problems. For offline problems, advanced strategies using multiple solvers can also be of interest. The numerical optimization frameworks provide this level of abstraction. Furthermore, they offer additional tools to ease the implementation of the problems. Among them, two interesting techniques are numerical differentiation and automatic differentiation. Numerical differentiation is based on local linearization to estimate the function gradient locally, while automatic differentiation uses advanced techniques to override the function evaluation in order to compute transparently the local gradient at the same time. Both of these techniques are useful for fast prototyping, but may result in slower code than manual implementations of Jacobian computation.

Solutions such as Matlab [11] provide a fast way to develop applications at the cost of efficiency. In constrast, C++ frameworks aim at reconciling ease of definition and efficiency.

3.1 OpenOpt

OpenOpt [12] is an optimization framework developed in Python and using NumPy. It interfaces with numerous state-of-the-art solvers such as IPOPT, Algencan, GLPKG, CPLEX, KNITRO, etc. It is bundled with FuncDesigner, allowing users to define their functions while the associated Jacobian is computed by automatic differentiation. If the user wants to implement the Jacobian manually, it can be checked by DerApproximator which validates computation using numerical differentiation. The framework is open-source and is distributed under the BSD licence. OpenOPT has been designed with offline optimization of large problems in mind, and thus supports sparse matrices.

3.2 RobOptim

RobOptim [13] is a numerical optimization framework written by the author of this paper and several other contributors. It is a templated C++ library designed to limit overhead as much as possible when solving problems, while providing various additional tools to define optimization problems easily in C++. RobOptim is organized in three layers: Core, Plug-Ins and Toolboxes. The Core layer provides a computational model for the problem: how to define a function, a constraint, build a problem, etc. The plug-ins contain the solver. IPOPT (dense and sparse), CFSQP, NAG (dense and sparse), CMinPack, NLopt, PaGMO are currently binded. The last layer, the toolboxes, provide useful functions for a particular problem. A toolbox for trajectory optimization is currently available and stable. Another one provides a way to generate optimal bounding capsules from polyhedrons. Future plans include the release of a toolbox for posture generation. RobOptim Core also provides tools to combine functions together (joining functions, splitting, binding one or more variables as well as basic mathematical operations like addition, chaining, etc.). This can be used to split the optimization problem into several smaller subproblems, thus easing development.

4. Performance Comparison with RobOptim

One of the major advantage of using a framework over using a particular solver directly is the ability to switch from one solver to another transparently. In particular, this characteristic allows users to realize benchmarks to choose the most appropriate solver in a particular case. This section compares the performances of

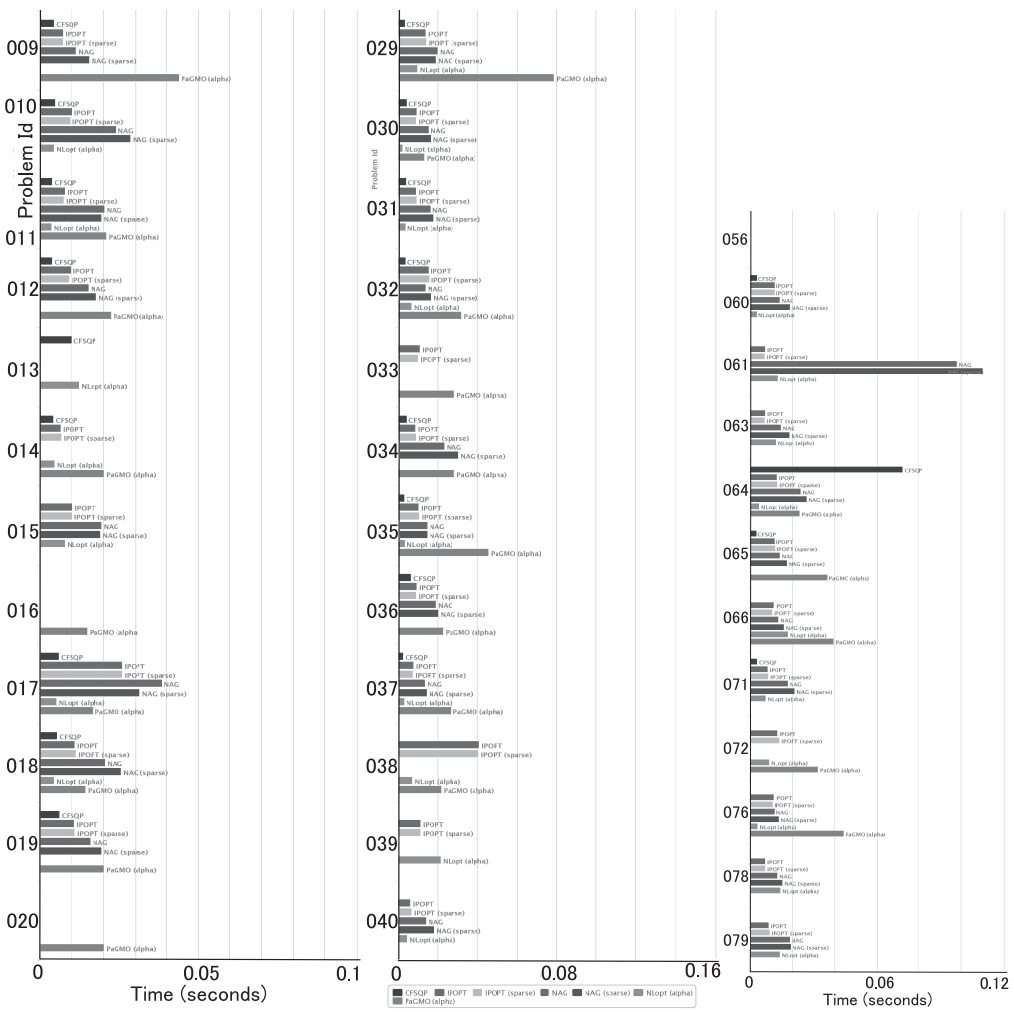

Fig. 1 Benchmarks of different solvers using RobOptim[†] running on the Hock-Shittkowski optimization test suite [14].

five solvers for a subset of the Hoch-Schittkowski test suite [14]. This testsuite has been published to cover a large panel of small nonlinear numerical optimization problems. The problem id refers to the problem number in the original paper. Problems whose id is below 57 are theoretical problems while numbers over or equal to 57 are matching practical problems. As the number increases, the number of variables increases as well to go from two variables (problem 1 to 24) to five variables (from problem 77). The small number of parameters is a limitation when compared to robotics problems usually treating much more variables. However, it allows us to implement gradients, Jacobian computations manually and verify that a global optima is reached with success at the end of the optimization process. Practical tests also include analytical solutions describing all local minima allowing us to compare even solvers which will not converge toward the same solution.

As shown on Fig. 1, the problem size, complexity, the parameter tuning may impact the resolution efficiency and even lead a solver to fail or succeed. It is recommended to benchmark specifically the problem that should be solved to choose the best solver.

In this benchmark, the parameters used by each solver are described below:

CFSQP the FSQP-AL algorithm is used (mode is set to 100)

IPOPT HSL's MA57 linear solver is used.

NAG the `nag_nlp_opt` routine is used

NLopt Augmented Lagrangian [8] with MMA [7] as local algorithm. NLopt plug-in is still in alpha phase. Only dense matrices are supported.

PaGMO MDE_pBX (Differential Evolution variant) algorithm [15]. PaGMO plug-in is still in alpha phase. Computation is done on just one thread. Weighted death penalty method is used for constrained problems. Dense matrices are used. Individual times vary between runs, but the total time remains consistent.

If a parameter value is unspecified in this list, the default one set by the solver is used.

Many solvers have been developed to solve efficiently nonlinear problems. However, roboticists are usually not expert in numerical optimization and choosing the right tool is a difficult task. Also, changing the robot structure or the robot task may change the problem complexity and the best set of parameters. Therefore, one can benefit a lot from choosing a versatile, flexible platform which can solve a wide variety of problems. Such frameworks exist and can also provide interesting additional tools as detailed in this paper. Testing and benchmarking robotics problems could lead to a better understanding of the best strategy to solve these problems.

From the results obtained while realizing RobOptim benchmarks and implementing robotics applications, a trend can be determined: solvers such as CFSQP tend to be often efficient. However, after tuning optimization parameters, it is common for solvers which were not particularly fast to get a significant performance improvement. Also, the number of function evaluations with respect to the number of gradient evaluations plays an important role when mechanisms such as automatic differentiation is used. Studying solvers can also help to choose between relying on automatic tools or going through the burden of implementing exact gradient computations.

5. Conclusion

This paper has presented various nonlinear solvers, numerical optimization frameworks as well as benchmarks of small nonlinear problems resolution. Given the fact that the solvers present very different efficiencies, may fail in some cases while other will succeed and provide different advantages and drawbacks (warm start, parallel function evaluation, etc.), it appears clearly that the best solver is tightly coupled to the problem one is dealing with. The use of an intermediate layer to decouple the problem from the solver can prove to be rewarding by allowing to try different approaches. The review of state-of-the-art solvers demonstrates that numerous solvers are available freely and be used for fast development and resolution of robotics problem.

Acknowledgements This research was partially supported by the Japan Society for the Promotion of Science (JSPS; Grant-in-Aid for JSPS Fellows P12803) and the FP7 IP RoboHow.Cog project (www.robohow.eu). FP7-ICT-2011-7 Contract No 288533.

[†]This benchmark will be regularly updated on the RobOptim website at http://roboptim.net/benchmark.html.

References

[1] A. Wächter and L.T. Biegler: "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," Mathematical Programming, vol.106, pp.25–57, 2006.

[2] P. Amestoy, I. Duff and J.-Y. L'Excellent: "Multifrontal parallel distributed symmetric and unsymmetric solvers," Computer Methods in Applied Mechanics and Engineering, vol.184, no.2, pp.501–520, 2000.

[3] HSL (2013). A collection of Fortran codes for large scale scientific computation, http://www.hsl.rl.ac.uk, 2013.

[4] C. Lawrence, J.L. Zhou and A.L. Tits: User's guide for CFSQP version 2.5: A C code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints, 1997.

[5] The Numerical Algorithms Group (NAG), Oxford: The NAG library (2013).

[6] S.G. Johnson: The NLopt nonlinear-optimization package, http://ab-initio.mit.edu/nlopt.

[7] K. Svanberg: "A class of globally convergent optimization methods based on conservative convex separable approximations," SIAM Journal on Optimization, vol.12, no.2, pp.555–573, 2002.

[8] A.R. Conn, N.I.M. Gould and P.L. Toint: "A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds," SIAM J. Numer. Anal., vol.28, no.2, pp.545–572, 1991.

[9] F. Biscani, D. Izzo and C.H. Yam: "A global optimisation toolbox for massively parallel engineering optimisation," CoRR, vol. abs/1004.3824, 2010.

[10] D. Izzo, M. Ruciski and F. Biscani: 'The generalized island model,' Parallel Architectures and Bioinspired Algorithms. pp.151–169, Springer, 2012.

[11] The MathWorks: Matlab Optimization Toolbox User's Guide.

[12] D. Kroshko: "OpenOpt: Free scientific-engineering software for mathematical modeling and optimization," 2007. [Online]. Available, http://www.openopt.org/

[13] T. Moulard, F. Lamiraux, K. Bouyarmane and E. Yoshida: "RobOptim: an Optimization Framework for Robotics," The Robotics and Mechatronics Conference (ROBOMEC), 2013.

[14] W. Hock and K. Schittkowski: "Test examples for nonlinear programming codes," Journal of Optimization Theory and Applications, vol.30, no.1, pp.127–129, 1980.

[15] S. Islam, S. Das, S. Ghosh, S. Roy and P. Suganthan: "An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization," Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, vol.42, no.2, pp.482–500, 2012.

Thomas Moulard

Thomas Moulard is JSPS fellowship at CNRS-AIST JRL since November 2012. He graduated from the EPITA engineering school in 2008. He prepared his thesis under the supervision of Florent Lamiraux and obtained his Ph.D. in 2012 from the INP Toulouse (Institut National Polytechnique). His research interests are: humanoid robotics, numerical optimization, robotics architecture and system integration.

Eiichi Yoshida

Eiichi Yoshida received M.E and Ph.D degrees on Precision Machinery Engineering from Graduate School of Engineering, the University of Tokyo in 1993 and 1996 respectively. From 1990 to 1991, he was visiting research associate at Swiss Federal Institute of Technology at Lausanne (EPFL). In 1996 he joined former Mechanical Engineering Laboratory, later reorganized as National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba, Japan. He served as Co-Director of AIST/IS-CNRS/ST2I Joint French-Japanese Robotics Laboratory (JRL) at LAAS-CNRS, Toulouse, France, from 2004 to 2008. Since 2009, he is Co-Director of CNRS-AIST JRL (Joint Robotics Laboratory), UMI3218/CRT, Tsukuba, Japan. His research interests include robot task and motion planning, modular robotic systems, and humanoid robots.

Benjamin Chrétien

Benjamin Chrétien is a Doctoral student at CNRS-UM2 LIRMM since October 2012. He received his M.S. degree in Aerospace Engineering from ISAE SUPAERO, Toulouse, France, in 2012, and currently prepares his thesis under the supervision of Abderrahmane Kheddar. His research interests are: humanoid robotics, GPGPU algorithms, numerical optimization and motion planning.