OPENPROD

ITEA 2

INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT

Prototype P2.28
Model-driven product optimization including use of expensive simulations

Peter Fritzson et al (LIU)

December, 2012

••

This document is public, as well as the associated deliverable.

Summary

This deliverable (P2.23 PU) includes two public prototypes on Modelica-based model-driven optimization listed below and appended to this document.

It also includes a public prototype on efficient optimization with heavy goal functions such as long-running simulations.

The first paper describes a tool integrated in OpenModelica provides optimization for parameter studies, including pareto front.

The second paper addresses dynamic optimization and uses collocation or multiple shooting to directly optimize the trajectory. One of the first parallel multi-core designs and implementations of these methods in a Modelica context is described and applied to several application models, on multi-core computers with up to 16 processors.

The companion deliverable in the T2.23 task is the P2.23b confidential prototype developed by SKF which focuses on efficient optimization with heavy goal functions including long-running simulations.

1. Optimization with Heavy Goal Functions

The goal function is produced by long-running simulations, e.g. in SKF Bearing simulations. A single evaluation of the goal function might take many hours up to several days. Derivatives of the goal function are not available.

There is a class of algorithms that can be used in this case, called model-building algorithms. Such algorithms build a model from the information returned by the few calls of the goal function. The model can be constructed by response surfaces, polynomials, or radial base functions. The model-building algorithm updates the model and its trust region (the region in hyperspace it applies to) in each step of the algorithm.

A well-known researcher in this area, Powell, developed such an algorithm in 2002 called UOBYQA, in Fortran. However, the implementation was hard to understand and extend.

A more modern re-implementation of most of UOBYQA in C/C++ called CONDOR was developed by other people 2005. This version was extended to handle constraints of three kinds: boundaries, linear, and nonlinear. The source code of CONDOR was available.

However, extension of CONDOR turned out to be difficult for the following reasons:

- Lack of use of object oriented programming techniques
- Own special-purpose implementations of everything: vectors/matrices, QR-factorization…
- No unit tests and extremely limited documentation

The following considerations wore done before starting the work on our improved implementation:

- More use of the C++ language for better abstraction and modularity.
- Boost for multithreading to be able to use parallelism.

- The Armadillo linear algebra library (uses LAPACK).
- Gtest för testing.

It turned out that one of us (Per Magnus Olsson) have now re-implemented most parts of CONDOR, including:

- Model building part
- Optimization algorithms

The improvements and extensions of CONDOR are the following:

- More modular architecture
- Parallellization at several levels, including (1) several starting points in parallel for the search with (different) parameters, and (2) parallel evaluation of the goal function at different points.
- Change of trust region through coordinate system change
- Documentation and test cases

The new CONDOR prototype is operational and can use parallelism on multi-core architectures. It is planned to be further extended and applied to industrial applications.

Publications included in this Document

1. Hubert Thieriot, Maroun Nemer, Mohsen Torabzadeh-Tari, Peter Fritzson, Rajiv Singh, and John John Kocherry. Towards Design Optimization with OpenModelica Emphasizing Parameter Optimization with Genetic Algorithms. In *Proceedings of the 8th International Modelica Conference (Modelica'2011)*, Dresden, Germany, March.20-22, 2011.
2. Bernhard Bachmann, Lennart Ochel, Vitalij Ruge, Mahder Gebremedhin, Peter Fritzson, Vaheed Nezhadali, Lars Eriksson, Martin Sivertsson. Parallel Multiple-Shooting and Collocation Optimization with OpenModelica. In *Proceedings of the 9th International Modelica Conference (Modelica'2012)*, Munich, Germany, Sept.3-5, 2012.

Towards Design Optimization with OpenModelica Emphasizing Parameter Optimization with Genetic Algorithms

Hubert Thieriot[a], Maroun Nemer[a], Mohsen Torabzadeh-Tari[b], Peter Fritzson[b], Rajiv Singh[c], John John Kocherry[c]

[a] *Center For Energy and Processes, MINES ParisTech, Palaiseau, France*
[b] *PELAB Programming Environment Lab, Dept. Computer Science, Linköping University, SE-581 83, Sweden*
[c] *Evonik Energy Services, Pvt. Ltd., Corporate Office, Noida 201 301, India*

Abstract

One of the main goals when modeling a physical system is to optimize its design or configuration. Currently existing platforms are often dependent on commercial software or are based on in-house and special-purpose development tools. These two alternatives present disadvantages that limit sharing and reusability. The same assessment has partly motivated the origin of the Modelica language itself. In this paper, a new optimization platform called OMOptim is presented. Intrinsically linked with OpenModelica, this platform is mainly aimed at facilitating optimization algorithm development, as well as application use together with models. A first version is already available and three test cases of which one using respectively Dymola and two using OpenModelica are presented. Future developments and design considerations of OMOptim but also of related OpenModelica computation functions are also discussed.

Keywords: Optimization, model-based, parameter, genetic algorithm, Modelica, modeling, simulation

1. Introduction

Model-based product development is an approach where a computer-based model of the product is built and refined before the actual production, to reduce costs, increase quality, and shorten time-to-market. Optimization is often used to improve product quality or design. Several types of optimizations can be used with these goals in mind. This can either concerns parameter or configuration optimization (e.g. which selection of the best components or connection paths to use in a defined process). Some design tasks also need a dynamic optimization to benchmark different configurations. For the user but also for the developer of such algorithms, two main issues can be noticed. The first issue concerns the development platform itself. The developers can either use a commercial plat-form (e.g. MatLab connected with a external simulator) or develop their own environment. The disadvantages of the first option are mainly the proprietary aspects of such tools which makes it harder to modify and extend, and also the involved license fees. The latter solution needs more development time and reduce exchange opportunity with other teams. Another important issue of model-based optimization lies in the computation time. Optimization applications often requires a large number of iterations and thus, a long time to give interesting results. This paper presents an initiative to limit these two main issues by developing an open-source optimization platform for OpenModelica (OMOptim) involving generation of efficient source code for multi-core computer architectures for increasing simulation performance.

1.1. Structure of the Paper

This paper first presents the context and motivation of the OMOptim development. A general review of optimization methods is then presented. The next sections successively describe the first version of OMOptim, an example of an application already

Email addresses: hubert.thieriot@mines-paristech.fr (Hubert Thieriot), maroun.nemer@mines-paristech.fr (Maroun Nemer), mohsen.torabzadeh-tari@liu.se (Mohsen Torabzadeh-Tari), peter.fritzson@liu.se (Peter Fritzson), r.singh@evonik-es.in (Rajiv Singh), jj.kocherry@evonik-es.in (John John Kocherry)

implemented and some concluding words about the intended future of this platform.

2. Requirements

The Center for Energy and Processes of Mines ParisTech school is involved in the CERES project [1] concerning industrial processes optimization. In this project, the best process technologies and heat recovery topology should be chosen simultaneously with mini-mum costs and environmental impacts. PELAB on the other hand is involved in the SSF Proviking EDOp project [2], concerning dynamic optimization for large industrial optimization problems, targeting both para-metric as well as dynamic optimization. This paper aims at building a bridge between these two projects with a common open source optimization platform. Thus, it should be ergonomic and efficient enough to use but also allow development of algorithms in the environment. A first version of this tool called OMOp-tim has been developed and is described below. Besides this goal, one critical issue will be the simulation (and thus) optimization time. Therefore, optimization algorithms but also the simulation tool efficiency should be very high. This paper briefly presents current and intended developments which go in this direction.

3. Optimization

This project aims at solving several different optimization problems, and in order to do this efficiently, a number of different solution techniques are required. Optimization problems can be classified according to several criteria e.g. existence of constraints, the nature of variables,and the nature of equations involved. A large number of optimization algorithms have been developed over the last decades to solve these different problems. One can roughly divide them in two families: gradient based methods and meta-heuristics algorithms.

3.1. Gradient based methods

The gradient based family contains numerical linear and non linear programming methods. These algorithms require substantial gradient information and are often used to improve a solution near a starting point. Applied on simple models, they offer an efficient way to find global optimum. However, many engineering optimization problems are highly nonlinear and present several optima. Such problems create numerical difficulties (like discontinuities) for this family algorithms and result can depend on initial point defined by the user.

3.2. Meta-heuristic algorithms

Meta-heuristic algorithms present a common characteristic: they combine rules and randomness to imitate natural phenomena. Within such methods, derivative computation is unnecessary. Most developed methods are evolutionary algorithms and genetic algorithms which are based on biological evolution formulation [3] but also tabu search, which reproduces animal behavior [4]. Simulated annealing is another meta-heuristic method based on physical annealing process [5].

3.2.1. Genetic algorithms and evolution strategies

A genetic algorithm (GA) is based on natural evolution and reproduces its main operations: reproduction, crossover and mutation. The initial theory has been proposed by Holland [6] and Goldberg [3] among others. An individual is represented by a genome which contains values of decisive parameters. For each individual, fitness values are calculated; these fitness values correspond to the objectives we want to minimize or maximize. A population is initially created by assigning random values to decisive parameters for each individual. New generations are created by combination of parents and innovation is introduced by mutation step. At each generation, a selection operation is followed which keep only the best individuals according to the fixed objectives but also following diversity parameters. Evolution strategies mainly differ from Genetic algorithms (GAs)in parameters coding: while GAs use binary coding and operations, evolution strategies use real coded parameters [7]. By extension, evolution strategies are often called genetic algorithms. These methods have largely been applied to estimate parameter values which minimize one or several objectives. It is indeed independent of problem type and can be applied to constrained or unconstrained problems, can have discrete or continuous variables, can follow one or several objectives and can be applied to linear or non-linear problems. Evolution strategies and more generally meta-heuristic algorithms present several advantages. First, they can be applied to complex engineering problems. They also do not need any

particular initialization point and are therefore independent of it. Finally, they tend to escape local optimum problems (e.g. with highly discontinuous problems). However, for linear and simple non-linear problems, linear or non-linear programming methods are much more suited and efficient (especially because of specific formulation and gradient information).

4. OMOptim 0.9

4.1. Goals

OMOptim intends to be a platform where different families of optimization algorithms can be implemented and linked with the OpenModelica simulator but also with other tools e.g. using FMI (Functional Mock-up Interface) [8]. Figure 1 illustrates its high-level design concept.

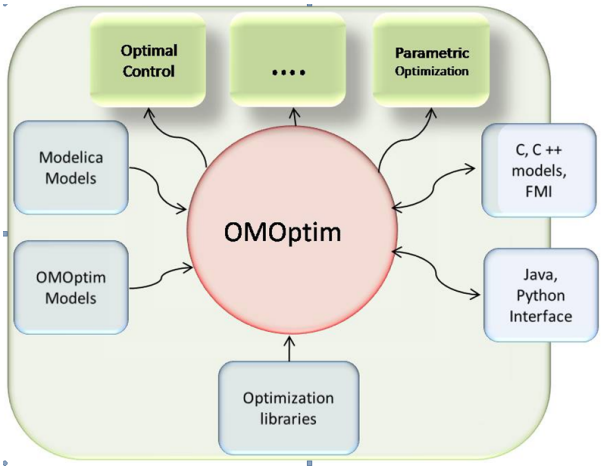

Figure 1: Top-level conceptual view of the OMOptim model-based optimization tool in OpenModelica.

4.2. Implementation

A first version of OMOptim including a graphical user interface has been developed in C++ and already tested on several use-cases (cf. Section 5). This version uses the OpenModelica API to read and eventually modify the model through the Corba communication protocol [9].

This version can only run meta-heuristic optimization methods since at this time, it does not have access to information about derivatives, even though OpenModelica can produce such information. As previously stated, only input variables specification and output variables reading are needed for such methods. Specifying input variables and reading results is done using input and output text files. To implement meta-heuristic algorithms, an efficient and adapted framework has been used (ParadisEO library [10]). OMOptim already includes several genetic algorithms, e.g. NSGA2, SPEA2 [11] or self-adaptative versions [12].

4.3. User interface

At the same time, a GUI has been developed allowing graphical selection of optimization variables, parameters and objectives (Figure 2) but also reading results.

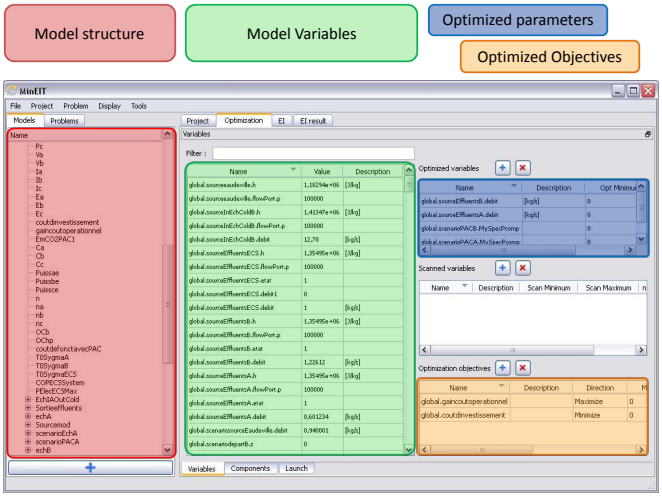

Figure 2: Parameters and objectives selection in the OMOptim optimization problem definition.

5. Test cases

Three test cases are presented here. The first uses Dymola as a simulation tool on an industrial application, but still uses OpenModelica to access the model structure. The second shows a small example application with OpenModelica. The third uses OpenModelica on an industrial application and an optimization module which is currently executed separately, but will be integrated with OpenModelica. As previously stated, meta-heuristic algorithms can interact with simulation tool using only input and output files. Thus, it is possible to interact with most simulation tools. However, in the future, all OMOptim algorithms may not be compatible with other simulators than OpenModelica (cf. section 6)

5.1. *Heat-pump application using Dymola for simulation*

A first application has been done which concerns a multi heat-pump system in a food industrial process [13]. This system consists of three heat-pumps used to heat-up solutions of the process. These three heat-pumps are connected to a heat-recovery stream. The model integrates dynamic items e.g. hot water tank emptying and filling during simulation (cf. Figure 3).

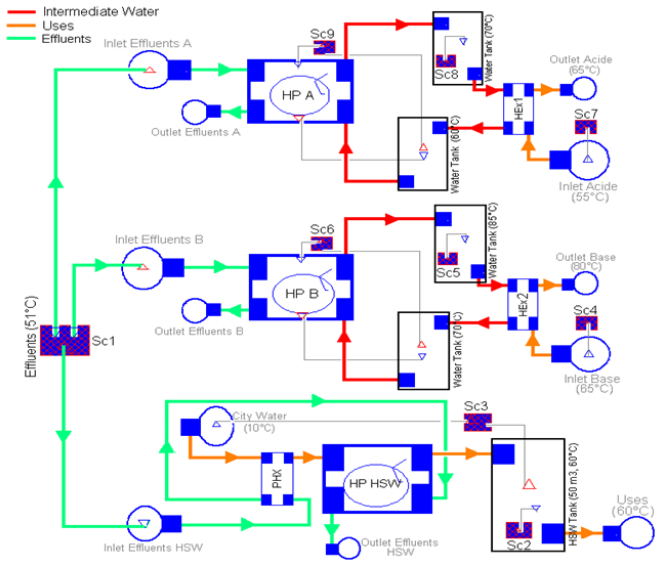

Figure 3: Modelica model of an industrial process being optimized.

The optimization consists in finding optimal flow repartitioning of the heat-recovery stream but also optimal powers of these heat-pumps, including the possibility to disable one or several heat-pumps. Two objectives are considered in this optimization: decreasing operational cost and investment cost. An auto-adaptive genetic algorithm has been developed for this study in OMOptim [13][12]. This genetic algorithm includes standard deviation of each genome parameter in the genome itself of the genetic algorithm. Therefore, the variation amplitude between each generation is itself submitted to modification and selection.

OMOptim allows the user to obtain several optimal configurations according to the two objectives fol-lowed i.e. investment and operating cost. Moreover, a sensitivity analysis has been performed to analyze the impact of CO2 carbon tax on optimum configurations (Figure 4).

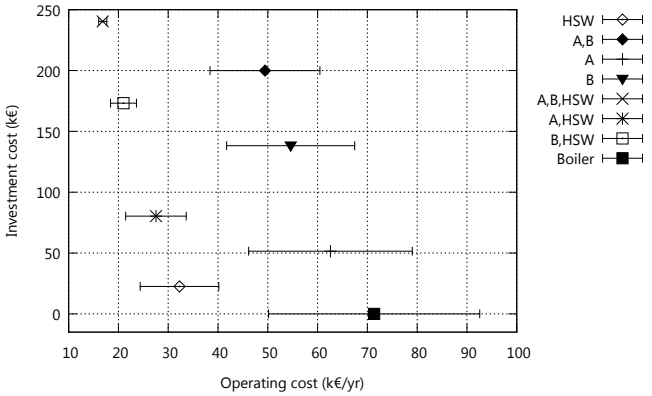

Figure 4: Investment and operation costs for optimal configurations (horizontal bars correspond to carbon tax variation sensitivity).

5.2. *A Linear actuator application using OpenModelica*

The model here consists of a linear actuator with a spring damped stopping [14, p. 583]. The model configuration is presented on Figure 5.

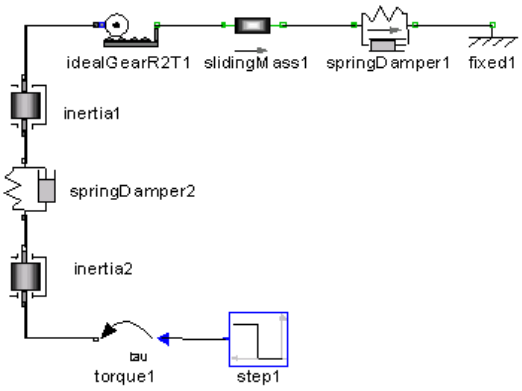

Figure 5: Linear actuator model

A reference response is generated considering a first order system. This response is defined by a first order ODE : $0.2 * \dot{y}_{ref}(t) + y_{ref}(t) = 0.05$. The optimization consists in making the resulting linear actuator behavior be as close as possible to this reference response. To achieve this, the damping parameters d_1 and d_2 of both spring dampers are considered as free variables to be determined by the optimization algorithm. The objective function corresponds to the integral of square deviation along simulation time T: $f(d) = \int_0^T (y(t) - y_{ref}(t))^2 \ dt$.

With obtained parameters ($d_1 = 4.90$ and $d_2 =$

19.88), the behavior suits the reference response well (cf. Figure 6). These results were obtained in less than five minutes on a standard Intel Core2 Duo @ 2.53 GHz.

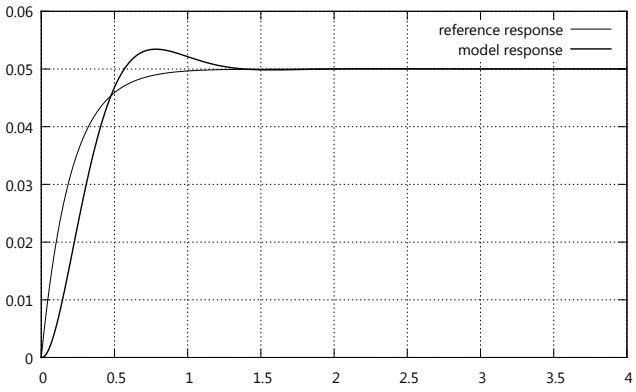

Figure 6: Model and reference responses after optimization of damper parameters

5.3. A dynamic optimization using an external SQP module

This third test case concerns a power plant regulation. It is only described very briefly here - a more detailed presentation is planned in a future paper. This application has been run using an external Sequential Quadratic Programming (SQP) optimization module.

In power plants, the main steam temperature control regulates the spray (attemperator) flow rate. Precise modeling of super heater dynamics and improving the quality of control of the superheated steam temperature is essential to improve the efficiency of the Boiler. In addition to this, the physical constraints of the turbine blades are also met using this control strategy. This control methodology is based on an adaptive prediction of the steam temperature trends. The architecture of the newly developed control system is similar to that of conventional boiler but the temperature feedback is given from the model instead of a sensor as shown in Figure 7.

A simple heat exchanger model is adapted to model the first stage of super heater regarding steam temperature, steam flow and flue gas temperature as measurements. This resulted into a set of algebraic differential equations which captured the behavior of the super heater along with the attemperator.

A SQP optimizer is used to calculate the spray flow, driven by an objective function to find the least

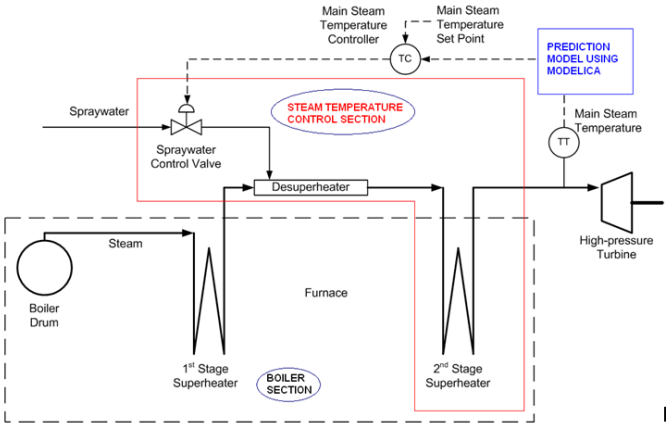

Figure 7: Advanced steam temperature control strategy for power plant

square error between the predicted and set point of steam temperature for a defined control horizon. Dynamic constraints are considered for spray and metal temperatures to consider the metal strains.

The first results are promising. However, this function is a separate module that is not yet integrated with the available version of OMOptim. This integration is planned in the near future.

6. Future work

6.1. OMOptim Structure Evolution

OMOptim intends to become an attractive framework to develop and execute optimization algorithms for Modelica users. To achieve this, its structure should be flexible enough to address the needs of many different kinds of optimization. The structure should also provide an efficient and ergonomic way to develop special-purpose algorithms including sharing and usage. Like a Modelica library, it would be pertinent and useful to list available optimization algorithms in libraries sharable within the Modelica community. Moreover, the structure should be able to support the combination of several algorithms working together. It should for example be used to apply a meta-heuristic optimization function with an objective function computed from another function (e.g. the objective could itself be the result of a sensitivity analysis). In some cases, it should also be possible to create new algorithms by graphically connecting existing optimization modules like in component-based modeling.

6.2. Hybrid Optimization

Meta-heuristic optimization algorithms can be coupled with local search functions [15]. This combination intends to combine advantages of both families. Meta-heuristics allow spreading populations over a large domain and thus limit the risk of obtaining a local optimum solution. Local search functions can lead to a faster convergence and to more precise results (e.g. [16] or [17]).

To achieve hybrid optimization implementation, a stronger link with OpenModelica should be built. In particular, gradient information should be communicated to optimization methods. The first developments in this direction are currently under way.

6.3. Dynamic optimization

Dynamic optimization requires modifying model parameters while performing the simulation. This functionality assumes the development an interface between OpenModelica and OMOptim while the former is computing. First trials have been done in this direction, using the new online interactive simulation facility of OpenModelica. More specifically, integration of a Sequential Quadratic Programming optimizer within OMOptim is planned in the near future (cf. section 5.3).

6.4. Parallelization for Efficient Computation

Applying parallelism and parallel compilation techniques at many levels of the problem, from problem formulation to inlining the solver and software pipelining, is being addressed in this project [18]. The constraints of the optimization problem can often be handled in parallel. In this case large system models can be restructured to smaller sub-system models. The PELAB research group at Linkping University has a long tradition of handling the compilation process in parallel, optimizing it, and adapting it for multi-core architectures. Some recent encouraging results[19] about using GPU architectures instead of CPU caused PELAB to invest in a two-teraflop (peak) Nvidia Fermi GPU that will be used in this project. Another step is to extend the support of efficient event-handling in parallelized code in order to also handle hybrid models.

6.5. Optimization Performance Profiling and Debugging

One current disadvantage of using high-level equation based languages [14] as well as other high-level simulation tools is the poor support for performance profiling and debugging. This will be even more pronounced when an engineer wants to trace the reason to why an optimization is too slow or has failed. There exists a substantial expertise at PELAB regarding debugging and traceability technology in integrated environments. We are planning to use this as a basis for a profiling feature in the optimization platform that is needed for tracing the causes of problems bottle-necks in the model.

7. RelatedWork

7.1. jModelica

The current Modelica language does not include formulating optimizations problems. However, a language extension called Optimica [20] has been developed by JModelica (www.jmodelica.org). JModelica offers an efficient platform for dynamic optimization and works in close collaboration with the model since it has an integrated Modelica compiler.

7.2. Dymola optimization library

The Dymola commercial tool from Dassault Systems [21], Dymola has its own optimization library, containing genetic algorithms. Another product from Dassault Systems is Isight [22] that supports process flow optimization with genetic algorithms. The main disadvantage of these two products is their closeness.

7.3. Meta-heuristic algorithms

Several tools may link meta-heuristic optimization methods to different simulators. One can cite OptiY [23], modeFrontier [24], Isight [22], or GenOpt [25]. They propose a rich list of implemented algorithms and can be used with nearly all simulators (all these tools interact with simulation software using input file modification and output file reading). Excepting GenOpt, all these softwares are commercial.

7.4. What should OMOptim offer

OMOptim aims to offer OpenModelica users an extension opening new opportunities. Especially, it intends to be a shared and open platform where scientists could develop optimization algorithms and apply them to Modelica models.

OpenModelica has been chosen for its opening and its substantial development rhythm. Also, OpenModelica supports symbolic differentiation which allows robust and advanced numerical methods, very

useful in optimization problems. This could be especially useful for the development of hybrid algorithms (cf. section 6.2).

Parallelism is also an intended development direction. Applying parallelism and parallel compilation techniques at many levels of the problem, from heuristic simulation repartition to inlining the solver and software pipelining, is being addressed in this project. For example, population based meta-heuristic optimization methods present high parallel scalability.

Concerning dynamic optimization or components/ connections that change during simulations, the Modelica language doesn't yet support structural dynamism, i.e. changes in the causality during simulations. However, with a little relaxation of this requirement the environments would be much flexible and better suited for optimization tasks [26].

References

[1] Ceres project : www.ecleer.com/ecleer_i_15/.

[2] Edop project : http://openmodelica.org/index.php/research/omoptim/edop.

[3] D. Goldberg, Genetic algorithms in search, optimization, and machine learning, Addison-wesley, 1989.

[4] F. Glover, Future paths for integer programming and links to artificial intelligence, Comp. Oper. Res. 13 (1986) 533–549.

[5] S. Kirkpatrick, C. Gelatt, M. Vecchi, Optimization by simulated annealing, Science 220 (1983).

[6] J. Holland, Adaptation in Natural and Artificial Systems, Ann Arbor, MI, 1975.

[7] T. Bck, H. Schwefel, An overview of evolutionary algorithms for parameter optimization, Evolutionary computation 1 (1993) 1–23.

[8] Itea2, modelisar : www.itea2.org.

[9] Openmodelica system documentation, available on http://www.openmodelica.org.

[10] A. Liefooghe, M. Basseur, L. Jourdan, E.-G. Talbi, Paradiseo-moeo: A framework for evolutionary multi-objective optimization, 2007.

[11] E. Zitzler, M. Laumanns, L. Thiele, Spea2: Improving the strength pareto evolutionary algorithm, in: EUROGEN, Citeseer, 2001, pp. 95–100.

[12] K. Deb, H. Beyer, Self-adaptation in real-parameter genetic algorithms with simulated binary crossover, in: Proceedings of the Genetic and Evolutionary Computation Conference, Citeseer, 1999, pp. 172–9.

[13] R. Murr, H. Thieriot, A. Zoughaib, D. Clodic, Multi-objective optimization of a multi water-to-water heat pump system using evolutionary algorithm, Submitted to Applied Energy. ().

[14] P. Fritzson, Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, Wiley-IEEE Press, 2004.

[15] A. Hopgood, L. Nolle, A. Battersby, Hybrid genetic algorithms: A review (2006) –.

[16] M. Fesanghary, M. Mahdavi, M. Minary-Jolandan, Y. Alizadeh, Hybridizing harmony search algorithm with sequential quadratic programming for engineering optimization problems, Computer methods in applied mechanics and engineering 197 (2008) 3080–3091.

[17] S. Katare, A. Bhan, J. Caruthers, W. Delgass, V. Venkatasubramanian, A hybrid genetic algorithm for efficient parameter estimation of large kinetic models, Computers & chemical engineering 28 (2004) 2569–2581.

[18] H. Lundvall, P. Fritzson, Automatic parallelization using pipelining for equation-based simulation languages, in: In proceedings of the 14th Workshop on Compilers for Parallel Computing (CPC'2009), Zurich, Switzerland.

[19] P. stlund, K. Stavker, P. Fritzson, Parallel simulation of equation-based models on cuda-enabled gpus, Submitted to EuroPar (2010).

[20] J. Akesson, K.-E. Arzen, M. Gafvert, T. Bergdahl, H. Tummescheit, Modeling and optimization with optimica and jmodelica.org–languages and tools for solving large-scale dynamic optimization problems, Computers & Chemical Engineering 34 (2010) 1737–1749.

[21] Dassault systemes, www.3ds.com.

[22] Isight product page : http://www.simulia.com/products/isight.html.

[23] Optiy, multidisciplinary analysis and optimization, http://www.optiy.eu.

[24] Modefrontier, http://www.modefrontier.com.

[25] Genopt, http://simulationresearch.lbl.gov/go/.

[26] H. Nilsson, G. Giorgidze, Exploiting structural dynamism in functional hybrid modeling for simulation of ideal diodes, in: Eurosim 2010.

Parallel Multiple-Shooting and Collocation Optimization with OpenModelica

Bernhard Bachmann[3], Lennart Ochel[3], Vitalij Ruge[3],
Mahder Gebremedhin[1], Peter Fritzson[1], Vaheed Nezhadali[2], Lars Eriksson[2], Martin Sivertsson[2]

[1]PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden
[2]Vehicular Systems, Dept. Electrical Engineering
Linköping University, SE-581 83 Linköping, Sweden
[3] Dept. Mathematics and Engineering, University of Applied Sciences,D-33609 Bielefeld, Germany

{bernhard.bachmann,lennart.ochel,vitalij.ruge}@fh-bielefeld.de,
{peter.fritzson,mahder.gebremedhin,vaheed.nezhadali,marsi}@liu.se

Abstract

Nonlinear model predictive control (NMPC) has become increasingly important for today's control engineers during the last decade. In order to apply NMPC a nonlinear optimal control problem (NOCP) must be solved which in general needs high computational effort.

State-of-the-art solution algorithms are based on multiple shooting or collocation algorithms, which are required to solve the underlying dynamic model formulation. This paper describes a general discretization scheme applied to the dynamic model description which can be further concretized to reproduce the multiple shooting or collocation approach. Furthermore, this approach can be refined to represent a total collocation method in order to solve the underlying NOCP much more efficiently. Further speedup of optimization has been achieved by parallelizing the calculation of model specific parts (e.g. constraints, Jacobians, etc.) and is presented in the coming sections.

The corresponding discretized optimization problem has been solved by the interior optimizer Ipopt. The proposed parallelized algorithms have been tested on different applications. As industrial relevant application an optimal control of a Diesel-Electric power train has been investigated. The modeling and problem description has been done in Optimica and Modelica. The simulation has been performed using OpenModelica. Speedup curves for parallel execution are presented.

Keywords: Modelica, Optimica, optimization, multiple shooting, collocation, parallel, simulation

1 Introduction

This paper presents efficient parallel implementations and measurement results of solution methods for nonlinear optimal control problems (NOCP) relevant for nonlinear model predictive control (NMPC) applications.

NMPC as well as NOCP have become increasingly important for industrial applications during the last decade [3], [4]. State-of-the-art solution algorithms [4] are based on multiple shooting or collocation algorithms, which are needed to solve the underlying dynamic model formulation. This paper concentrates on parallelizing these time-consuming algorithms, which finally lead to a very fast solution of the underlying NOCP. Moreover, a general discretization scheme applied to the dynamic model description is introduced, which can be further concretized to reproduce the common multiple shooting or collocation approach [7] and can also be refined to represent total collocation methods [4] in order to solve the underlying NOCP much more efficiently. The modeling and problem description is done in Modelica [2] extended with optimization goal functions and constraints specified as in Optimica [15]. The simulation is performed using OpenModelica [1]. Speedup curves for parallel execution are presented for application examples.

Section 2 describes the underlying mathematical problem formulation including the objective function and constraints to the state and control variables. The general discretization scheme applied is discussed in Section 3. This approach can be further refined to represent multiple shooting or collocation algorithms for the solution process, which is described in Section 4.

In section 5 the general discretization scheme is further developed towards total collocation methods.

Industrial relevant Modelica applications are presented in Section 6. Parallel execution of the constraint equations of the NOCP is performed in Section 7. The results show reasonable speedups of the optimization time when it comes to time consuming calculation of the model equations. The necessary implementations are partly realized in the OpenModelica Compiler, which is described in Section 8. The paper concludes with a summary of the achieved results.

2 The Nonlinear Optimal Control Problem (NOCP)

The numerical solution of NOCP is performed by solving the following problem formulation [7][8]:

$$\min_{u(t)} J(x(t),u(t),t) = E_{t_f}\left(x(t_f)\right) + \int_{t_0} L(x(t),u(t),t)\,dt \tag{2.1}$$

subject to

$$x(t_0) = h_0 \tag{2.2}$$
$$\dot{x}(t) = f(x(t),u(t),t) \tag{2.3}$$
$$g(x(t),u(t),t) \geq 0 \tag{2.4}$$
$$r\left(x(t_f)\right) = 0 \tag{2.5}$$

where $x(t) \in \mathbb{R}^{\eta_x}$ and $u(t) \in \mathbb{R}^{\eta_u}$ are the state and control variables, respectively. The receding time horizon is given by the interval $[t_0, t_f]$. The constraints (2.2), (2.3), (2.4) and (2.5) describe the initial conditions, the nonlinear dynamic model description based on differential algebraic equations (DAEs, Modelica), the path constraints ($g(x(t),u(t),t) \in \mathbb{R}^{\nu}$) and the terminal constraints.

Support for time-optimal control and corresponding terminal constraints is work-in-progress and are not yet provided by the current implementation.

2.1 Boundary Value Problems

The objective function (2.1), that needs to be minimized, includes conditions at the boundary time point t_f stated by the function $E\left(x(t_f)\right)$ as well as conditions taking into account the whole time horizon stated by $\int_{t_0}^{t_f} L(x(t),u(t),t)\,dt$.

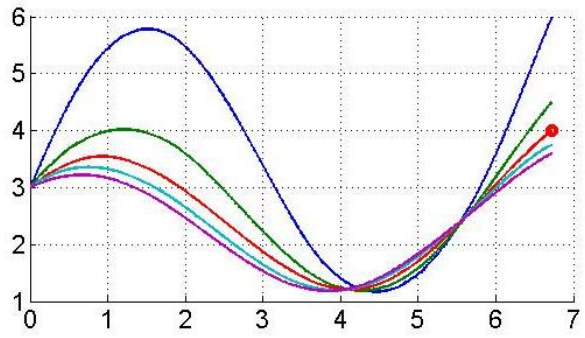

Figure 1. Different trajectories achieved by varying control variables. Only one trajectory fulfills the terminal constraint (red dot).

The function $E\left(x(t_f)\right)$ describes conditions that should be fulfilled at the final time point similar to the terminal constraint (2.5). Since $E\left(x(t_f)\right)$ is part of the objective function $J(x(t),u(t),t)$ the applied optimization methods may not find a solution that fulfills the corresponding terminal constraints, but should be very close to it. The trajectories are influenced by changing the control variables. Different trajectories using different control variables are visualized in **Figure 1**.

On the other hand, different trajectories could fulfill the same terminal constraints. Taking into account the whole time horizon by minimizing the second part $\int_{t_0}^{t_f} L(x(t),u(t),t)\,dt$ of the objective function will lead to the selection of the optimal trajectory. This behavior is visualized in **Figure 2**.

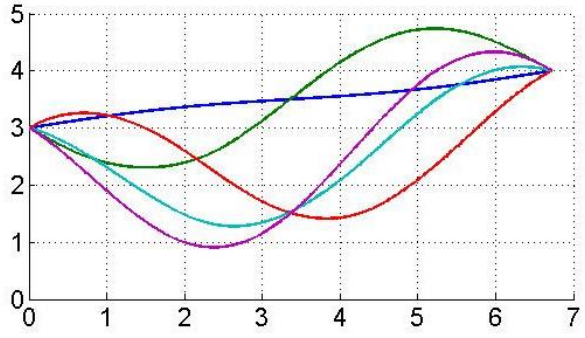

Figure 2. Different trajectories that fulfill the terminal constraint.

3 General Discretization Scheme

In order to apply a general discretization scheme the NOCP formulation is rewritten to a general form which later can be used to derive the different possible numerical algorithms e.g. multiple shooting, multiple or total collocation algorithm, etc. [6]. Equations (2.2) and (2.3) can be rewritten as follows:

$$x(\tau) = h_0 + \int_{t_0}^{\tau} f(x(t),u(t),t)\,\mathrm{dt}. \tag{3.1}$$

When discretizing the time horizon $[t_0, t_f]$ into a finite number of intervals $[t_0, t_1], \ldots, [t_{n-1}, t_n]$ (e.g. equidistant partitioning: $t_j := t_0 + l \cdot j,\ j = 0, \cdots, n,\ l := \frac{t_f - t_0}{n}$) integral in (3.1) can be reformulated to

$$\int_{t_0}^{\tau} f(x(t), u(t), t)\ \mathrm{dt} = \sum_{i=0}^{n-1} \int_{t_i}^{t_{i+1}} f(x(t), u(t), t)\ \mathrm{dt}. \tag{3.2}$$

Each integral

$$\int_{t_i}^{t_{i+1}} f(x(t), u(t), t)\ \mathrm{dt} \tag{3.3}$$

on a subinterval can now be treated independently, if additional constraints are added to the NOCP formulation to force the calculation of an overall continuous solution. Therefore, locally the problem reduces to a boundary value problem [5] stated by

$$x_i(t_{i+1}) = h_i + \int_{t_i}^{t_{i+1}} f(x_i(t), u(t), t)\ dt \tag{3.4}$$

where $x_i(t) := x(t)$ for $t \in [t_i, t_{i+1}]$, $i = 0, \cdots, n-1$. It yields $x_i(t_i) = h_i$ and continuity is forced by additional constraints $x_i(t_{i+1}) = h_{i+1}$ added to the NOCP formulation, which finally leads locally to a boundary value problem. Each sub-problem (3.4) can be solved independently and in parallel, if multiple shooting/collocation is applied. By varying the control variable $u(t)$ in each sub-interval the solution of (3.4) can be influenced in order to fulfill the overall continuity constraints. In the current approach it is assumed that $u(t) = u_i$ is constant for each subinterval $[t_i, t_{i+1}]$.

4 Multiple Shooting or Collocation

Different numerical methods are available to solve equation (3.4). The first approach presented within this paper is the reformulation of (3.4) to an ordinary differential equation

$$\dot{x}_i(t) = f(x_i(t), u_i, t) \tag{4.1}$$

with the initial condition $x_i(t_i) = h_i$.

In order to solve equation (4.1) an appropriate (e.g. explicit/implicit) integration algorithm can be applied that is already available in OpenModelica. A schematic view of the algorithmic dependencies is presented in **Figure 3**.

Alternatively, equation (3.4) or (4.1) can locally be solved using collocation methods, which also can be interpreted as numerical treatment of integration. Detailed descriptions of the multiple shooting algorithm using local collocation can be found in [7]. The solution process for equation (3.4) in each subinterval can be performed in parallel. The necessary calculation time depends certainly on the chosen integration method. In case of an explicit integration algorithm, e.g. Runge-Kutta based, more intermediate integration steps might be necessary for certain accuracy than using an implicit integration method, e.g. local collocation methods. On the other hand, explicit integration methods just perform at each intermediate step an evaluation of the model equations, whereas implicit methods in general need to solve a system of non-linear equations, which might also be time consuming. Nevertheless, when the underlying system of ordinary differential equations is stiff, implicit methods need to be applied.

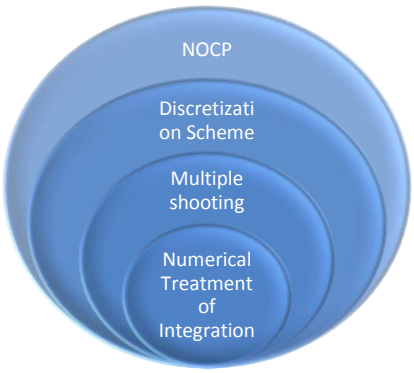

Figure 3. Schematic view of the algorithmic dependencies.

Although, equation (3.4) can be solved in parallel a lot of time is used for finding exact solutions to a locally defined problem, which might not be relevant for the over-all problem stated by the (NOCP) formulation (2.1)-(2.5). Therefore, the solution process for the NOCP still needs a lot of computation time. The next section describes methods to overcome this deficiency by adding the locally derived residual equations (based on locally applied collocation methods) to the over-all NOCP formulation.

5 Total Collocation

Applying collocation methods for solving equation (3.4) locally leads in general to a system of non-linear equations for each sub-interval. The solution process of these equations might be time consuming and with respect to the NOCP not efficient. If the corresponding non-linear equations are added to the NOCP formulation and corresponding optimization algorithms have access to the intermediate points used by the local collocation method a more efficient solution process can be formulated [4]. This section presents two different collocation methods.

Based on the common Lagrangian polynomial $p_j(z)$ for interpolation purposes, following abbreviations are introduced for $j = 0, \cdots, m$ and $k = 1, \cdots, m$:

$$p_{k,j} \coloneqq p_j(z_k) \coloneqq \prod_{\substack{l=0 \\ l \neq j}}^{m} \frac{z_k - z_l}{z_j - z_l},$$

$$\partial p_{k,j} \coloneqq \frac{\partial p_j}{\partial z}(z_k) \coloneqq \sum_{\substack{l1=0 \\ l1 \neq j}}^{m} \frac{1}{z_{l1} - z_j} \cdot \prod_{\substack{l2=0 \\ l2 \neq j \\ l2 \neq l1}}^{m} \frac{z_k - z_{l2}}{z_j - z_{l2}} \quad \text{and}$$

$$\int p_{k,j} \coloneqq \int_0^{z_k} p_j(z)\, dz$$

where $z_0, \cdots, z_m$ are the supporting points within the reference interval [0,1]. Further abbreviations are defined by $t_{j,i} = t_i + l \cdot z_j$, $x_{m,0} \coloneqq h_0, x_{j,i} \coloneqq x_i(t_{j,i})$, and $f_{j,i} \coloneqq f(x_{j,i}, u_i, t_{j,i})$.

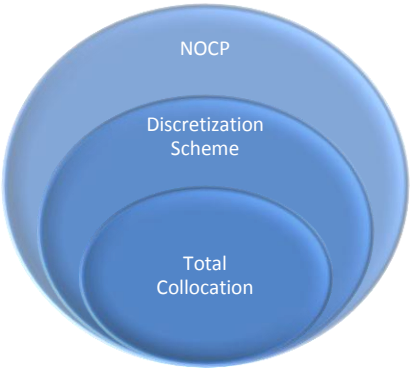

Figure 4. Schematic view of the algorithmic dependencies.

The first variant is dealing with the approximation of the states which leads to the following formulas:

$$\begin{aligned} x_{k,i} &= p_{k,0} \cdot x_{m,i-1} + \sum_{j=1}^{m} p_{k,j} \cdot x_{j,i} \\ l \cdot f_{k,i} \approx \phi_{k,i} &\coloneqq \partial p_{k,0} \cdot x_{m,i-1} + \sum_{j=1}^{m} \partial p_{k,j} \cdot x_{j,i} \end{aligned} \tag{5.1}$$

In case of $m = 1$ this approach reduces to the implicit Euler formula with approximation order 1.

The second variant is dealing with the approximation of the derivatives of the states and leads to the formulas:

$$\begin{aligned} x_{k,i} \approx \psi_{k,i} &\coloneqq x_{m,i-1} + l \cdot \sum_{j=0}^{m} \int p_{k,j} \cdot f_{j,i} \\ f_{k,i} &= \sum_{j=0}^{m} p_{k,j} \cdot f_{j,i} \end{aligned} \tag{5.2}$$

In case of $m = 1$ this approach reduces to an implicit Runge-Kutta formula (trapezoidal rule) with approximation order 2.

The discretized NOCP using total collocation and corresponding Gaussian quadrature formula for the integral part of the goal function is finally described by:

$$\begin{aligned} &\min_{u(t)} J(x(t), u(t), t) = E(x_{m,n}) + \\ &l \cdot \sum_{i=1}^{n} \sum_{j=0}^{m} \omega_j \cdot L(x_{j,i}, u_i, t_{j,i}) \end{aligned} \tag{5.3}$$

subject to

$$\begin{aligned} s(x_{k,i}, u_i, t_{k,i}) &= 0 \\ u(t_{k,i}) &= u_i \\ g(x_{m,i-1}, u_i, t_{k,i}) &\geq 0 \\ r(x_{m,n}) &= 0 \end{aligned} \tag{5.4}$$

for $i = 1, \cdots, n$, $k = 1, \cdots, m$. For variant 1 the supporting points $z_1, \cdots, z_m$, and weights $\omega_1, \cdots, \omega_m$ are given based on Radau formulas. $z_0 = 0$, $\omega_0 = 0$. $s(x_{k,i}, u_i, t_{k,i}) = l \cdot f_{k,i} - \phi_{k,i}$ are the additional residual equations from (5.1). For variant 1 the supporting points $z_0, \cdots, z_m$, and weights $\omega_0, \cdots, \omega_m$ are given based on Lobatto formulas. $s(x_{k,i}, u_i, t_{k,i}) = x_{k,i} - \psi_{k,i}$ are the additional residual equations from (5.2).

6 Modelica Applications

To investigate the performance of the proposed optimization algorithm, industrial relevant optimal control problems are solved and corresponding results are presented in this section.

6.1 Batch Reactor

We begin by considering a simple model from the chemical reactor described in [7] to maximize the yield of $x_2(t)$ by manipulation the reaction temperature $u(t)$, with the following problem formulation:

$$\min_{u(t)} J(x(t), u(t), t) = -x_2(1) \tag{6.1}$$

subject to

$$\begin{aligned} \dot{x}_1(t) &= -\left(u(t) + \frac{u^2(t)}{2}\right) \cdot x_1(t) \\ \dot{x}_2(t) &= u(t) \cdot x_1(t) \end{aligned} \tag{6.2}$$

$$\begin{pmatrix} x_1(t) \\ 1 - x_1(t) \\ x_2(t) \\ 1 - x_2(t) \\ u(t) \\ 5 - u(t) \end{pmatrix} \geq 0 \tag{6.3}$$

$$x(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{6.4}$$

where $x(t) = (x_1(t), x_2(t))^T$ and $t \in [0,1]$.

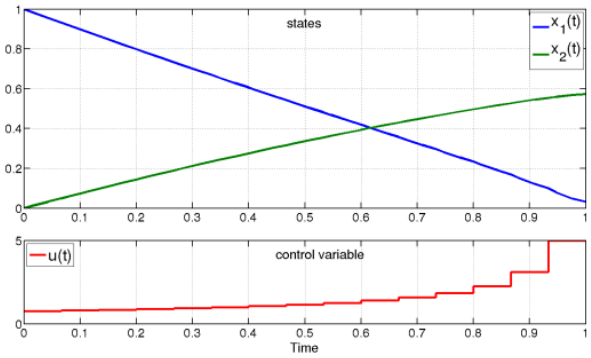

Figure 5. Trajectories of state and control variables

6.2 Optimal control of Diesel-Electric powertrain

The Diesel-electric model based on [10] is presented in Appendix A. This concept is modeled according to a nonlinear mean value engine model (MVEM) containing four states and three control inputs while the generator model is simplified by considering constant efficiency and maximum power over the entire speed range.

In a Diesel-electric powertrain the operating point of the Diesel engine can be freely chosen which would potentially decrease fuel consumption. Moreover, the electric machine has better torque characteristics. These are the main reasons making the Diesel-electric powertrain concept interesting for further studies.

To investigate the fuel optimal transients of the powertrain from idling condition to a certain power level while the accelerator pedal position is interpreted as a power level request, the following optimal control problem is solved:

$$\text{states } x = \begin{pmatrix} \omega_{ice} \\ p_{im} \\ p_{em} \\ \omega_{tc} \end{pmatrix}, \text{ controls } u = \begin{pmatrix} u_f \\ u_{wg} \\ P_{gen} \end{pmatrix}$$

$$\min \int_0^T \dot{m}_f \, dt$$

subject to

$$\begin{aligned}
\dot{x}_1 &= f_2(x_2, x_3, u_1, u_3) \\
\dot{x}_2 &= f_3(x_1, x_2, x_4) \\
\dot{x}_3 &= f_4(x_1, x_2, x_3, u_1, u_2) \\
\dot{x}_4 &= f_5(x_2, x_3, x_4, u_2) \\
0 &= f_6(x_2, x_4) - f_7(x_1, x_2) \\
0 &= f_7(x_1, x_2) + f_8(x_1, u_1) - f_9(x_3) - f_{10}(x_3, u_3) \\
0 &= \frac{f_{11}(x_3) - f_{12}(x_1)}{f_{13}(x_4)} - f_{14}(x_4)
\end{aligned}$$

$$\begin{aligned}
54\,rps &\leq x_1 \leq 220\,rps \\
0.8\,P_{\text{amb}} &\leq x_2 \leq 2P_{\text{amb}} \\
P_{\text{amb}} &\leq x_3 \leq 3P_{\text{amb}} \\
300rps &\leq x_4 \leq 10000\,rps \\
0 &\leq u_1, u_2 \leq 1
\end{aligned}$$

and boundary conditions are:

$$\text{at } t = 0, \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \text{ idle operating values,}$$

$$\text{at } t = T, \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} = 0, \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \text{desired values,}$$

and $u_3 = P_{\text{required}}$.

The constraints are originated from components' limitations and the functions f_i are described in the appendix [10].

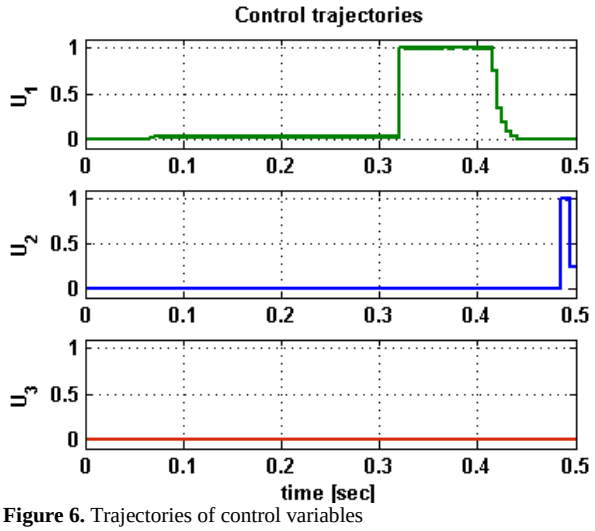

Figure 6. Trajectories of control variables

In this work, we try to find the fuel optimal control and state trajectories in a certain time interval [0,0.5]. For simplicity, only diesel operating condition is assumed which means ($u_3 = P_{gen} = 0$).

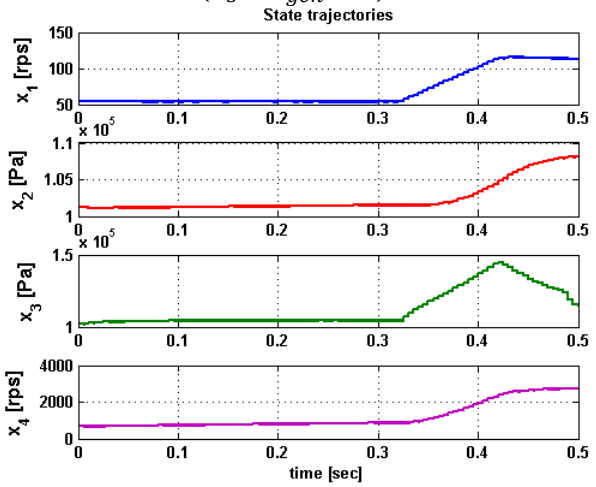

Figure 7. Trajectories of state variables

The dynamic system is solved after it is discretized into subintervals. **Figure 6** and **Figure 7** show the obtained control and state trajectories. As it is expected, the fuel optimal results happen when engine is accelerated only near the end of the time interval ($t \approx 0.32\ s$) to meet the end constraints while minimizing the fuel consumption.

In section 7 it is shown how the parallel execution increases the performance of the optimization process.

7 Parallel Execution and Performance Measurements

We have performed measurements for the different algorithms (multiple shooting/collocation and total collocation with variant 1 and 2) applied to the above described applications. The C/C++ source code has been compiled by gcc version 4.6.3 (GCC) with OpenMP support. The measurements are done on an Intel Core i7 CPU 870 with 8 cores @ 2.93 GH (4 real cores and 4 virtual cores).

The corresponding optimization problem is solved by the interior point optimizer Ipopt [16]. **Figure 8** shows the different functions and derivative information that need to be provided to Ipopt for the solution process. In the current implementation the Hessian matrix of the corresponding Lagrangian formulation is calculated numerically by Ipopt. The other information (see **Figure 8**) is provided numerically by external routines. When calculating the Jacobian and Hessian matrices the treatment of the sparsity patterns, is important for the performance of the multiple shooting and total collocation methods [9]. This has been realized for the Jacobian matrix calculation.

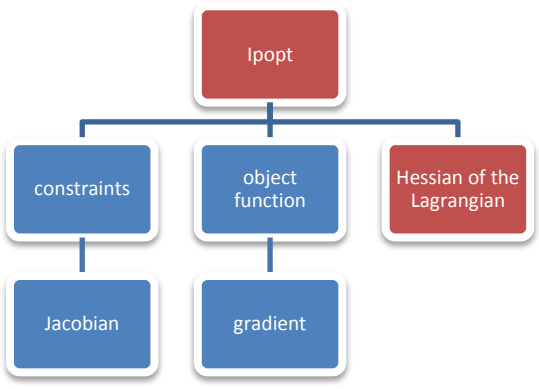

Figure 8. Schematic view of the required components of Ipopt

The multiple shooting algorithm uses an explicit Runge-Kutta formula of order 3 as well as 3 steps within each interval. The multiple collocation method uses 3 intermediate interval points based on Radau formulas. The total collocation uses variant dependent intermediate interval points as described in section 5. The tests have been performed using 128 intervals when dealing with sparse matrix representation. The user defined functions (see blue boxes of **Figure 8**) have been parallelized.

7.1 Batch Reactor

The speedups obtained and the computation times for the batch reactor are shown in **Table 1** and **Figure 9**.

	multiple shooting		multiple collocation	
threads	Ipopt	jac_g	Ipopt	jac_g
1	1,5742s	28,93ms	18,47s	343,3ms
2	1,0164s	16,77ms	10,25s	188,3ms
4	0,6691s	9,37ms	5,825s	104,7ms
8	0,6539s	8,52ms	5,055s	89,57ms

Table 1. Computation times for the Jacobian of the constraints and the over-all optimization using multiple shooting/collocation method for the batch reactor

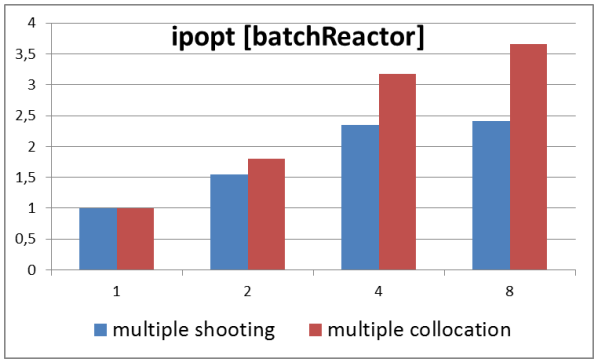

Figure 9. Speedups and computation times of the whole optimization process

Table 1 shows that multiple collocation is much more expensive than the multiple shooting. Reason for this is the computational time needed to solve non-linear systems coming from the implicit discretization. Therefore, by parallelizing the user defined functions a better speedup (**Figure 9**) for the whole optimization can be performed for the multiple shooting method, whereas the speedup for the user defined function (e.g. **Figure 10**) is comparable.

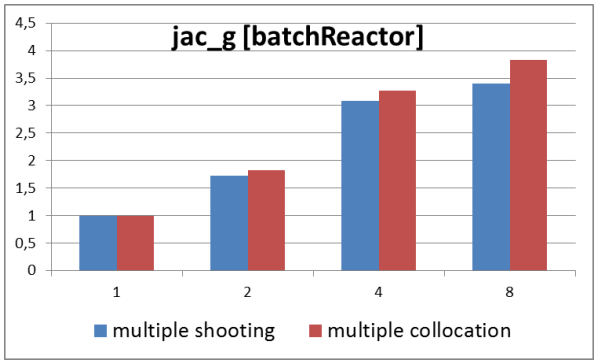

Figure 10. Speedups and computation times for the Jacobian of the constraints

7.2 Diesel Model

The solution process for the diesel model using multiple shooting and multiple collocation is quite time consuming (see **Table 2** and **Table 3**). Especially, the multiple collocation algorithm was only performed with 32 intervals in order to reduce execution time to an acceptable level. Although, parallelization of the user defined function leads to a great speed up, the overall performance of the multiple shooting or collocation method is still poor. The total collocation variants are superior with respect to the over-all performance as can be seen in **Table 3**.

	multiple shooting		multiple collocation	
threads	Ipopt	jac_g	Ipopt	jac_g
1	1518,4s	1,8196s	368,07s	2,6007s
2	917,17s	0,9671s	196,04s	1,3832s
4	608,29s	0,5286s	108,33s	0,7625s
8	508,71s	0,3861s	87,027s	0,6110s

Table 2. Computation times for the Jacobian of the constraints and the over-all optimization using multiple shooting/collocation method for the diesel model

	total collocation 1		total collocation 2	
threads	Ipopt	jac_g	Ipopt	jac_g
1	15,40s	8,215ms	14,07s	9,947ms
2	11,49s	4,356ms	10,10s	5,281ms
4	10,19s	2,553ms	8,342s	2,987ms
8	9,452s	1,713ms	7,897s	1,965ms

Table 3. Computation times for the Jacobian of the constraints and the over-all optimization using total collocation method for the diesel model

The speed-up regarding the user-defined function is comparable to the multiple shooting or collocation methods (see **Figure 12**). The speed-up of the whole optimization process is not optimal due to the serial computation and dense treatment of the Hessian matrix calculated internally by Ipopt (see **Figure 11**).

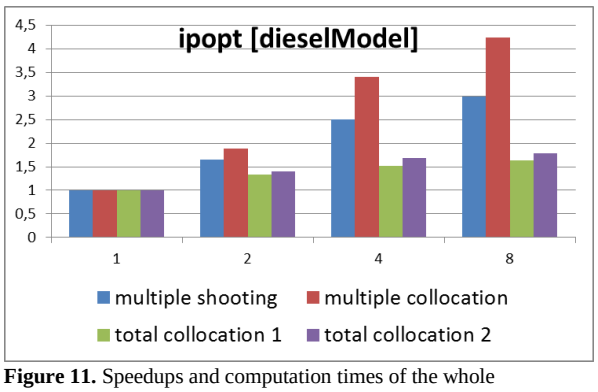

Figure 11. Speedups and computation times of the whole optimization process

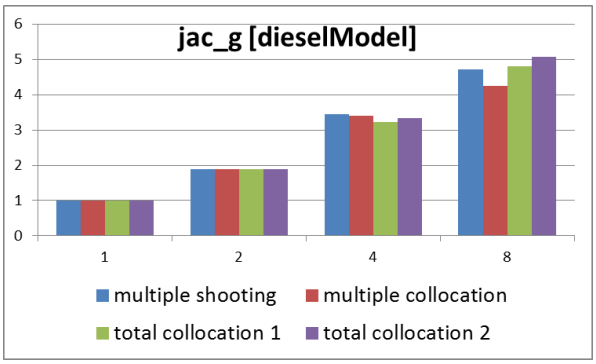

Figure 12. Speedups and computation times for the Jacobian of the constraints

8 Integration with OpenModelica

Support for specifying optimization goal functions and constraints together with Modelica models has now been implemented in OpenModelica. Such integrated models can now be exported via XML to tools such as CasADi [12] which can act as a frontend to ACADO [13].

In the current OpenModelica prototype all aspects of the tool chain are not yet completely implemented. For example, we are currently using numerically derived Gradients, Jacobians and Hessians since the automatic differentiation machinery in OpenModelica has not yet been extended to operate on the optimization problem goal function.

However, the prototype is complete enough to do the measurements of the included model applications on a parallel platform to obtain the speedup curves for parallel execution on 1-8 cores.

The OpenModelica compiler has been extended to export Modelica Models to XML based on an extended version of the FMI XML schema from [14]. The XML export, in addition to the standard Modelica syntax, supports the Optimica extensions from Jmodelica [15]. Theses extensions allow users to formulate dynamic optimization problems to be solved by a numerical algorithm. The extensions include several constructs including a new specialized class optimization, a constraint section, etc. See the batch reactor example below as well as the Optimica manual for complete information.

```
optimization BatchReactor
             (objective = -x2(finalTime),
              startTime = 0, finalTime =1)
  Real x1(start=1,fixed=true,min=0,max=1);
  Real x2(start=0,fixed=true,min=0,max=1);
  input Real u(free=true, min=0, max=5);
equation
  der(x1) = -(u+u^2/2)*x1;
  der(x2) = u*x1;
end BatchReactor;
```

The XML generated for flattened Optimica Models can be imported into other non-Modelica Optimization tools like ACADO.

Currently the OpenModelica compiler does not yet use the optimization problem formulation internally as input to automatic differentiation. The Modelica plus Optimica model description is flattened, some common compilation phases are applied e.g. syntax, semantics and type checking, simplification, constant evaluation etc. and then the complete flat model is exported to XML.

9 Conclusions

In this paper parallelized implementations of several different algorithms for solving NOCP have been presented. The well-known multiple shooting or collocation as well as total collocation methods are derived using a general discretization scheme. Total collocation methods have proofed at least in the current implementation and for the tested applications to be superior to the other algorithms.

The corresponding discretized optimization problem has been solved by the interior optimizer Ipopt. Further speedup of the optimization process for all described algorithms have been achieved by parallelizing the calculation of model specific parts (e.g. constraints, Jacobians, etc.). So far the evaluation of derivatives have been done numerically. This will be further improved using the already available symbolic differentiation capabilities of OpenModelica [11]. Finally, this work will be continued by applying the proposed algorithms on more industrial relevant applications together with a thorough testing on advanced parallel hardware architectures.

10 Acknowledgements

This work has been partially supported by Serc, by SSF in the EDOp project and by Vinnova as well as the German Ministry BMBF (BMBF Förderkennzeichen: 01IS09029C) in the ITEA2 OPENPROD project. The Open Source Modelica Consortium supports the OpenModelica work.

References

[1] Open Source Modelica Consortium. *OpenModelica System Documentation Version 1.8.1*, April 2012. http://www.openmodelica.org

[2] Modelica Association. *The Modelica Language Specification Version 3.2*, March 24th 2010. http://www.modelica.org. Modelica Association. *Modelica Standard Library 3.1*. Aug. 2009. http://www.modelica.org.

[3] Jasem Tamimi, Pu Li. A combined approach to nonlinear model predictive control of fast systems. Journal of Process Control, 20, pp 1092–1102, 2010.

[4] Biegler, Lorenz T. 2010. Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes. s.l. : Society for Industrial Mathematics, 2010.

[5] Munz, Claus-Dieter and Westermann, Thomas. 2009. Numerische Behandlung gewöhlicher und partieller Differenzialgleichungen. Berlin Heideberg : Springer Verlag, 2009

[6] Heuser, Harro. 2006. Gewöhnliche Differentialgleichungen. Wiesbaden : Teubner Verlag, 2006.

[7] Tamimi, Jasem. 2011. Development of Efficient Algorithms for Model Predictive Control of Fast Systems. Düsseldorf: VDI Verlag, 2011.

[8] Friesz, Terry L. 2007. Dynamic Optimization and Differential Games. US: Springer US, 2007.

[9] Folkmar, Bornemann und Deuflhard, Peter. 2008. Numerische Mathematik: Numerische Mathematik 2: Gewöhnliche Differentialgleichungen: Bd II: [Band] 2. s.l. : Gruyter, 2008.

[10] Martin Sivertsson and Lars Eriksson Optimal power response of a diesel-electric powertrain. Submitted to ECOSM'12, Paris, France, 2012.

[11] Braun, Willi, Ochel Lennart and Bachmann Bernhard. Symbolically Derived Jacobians Using Automatic Differentiation - Enhancement of the OpenModelica Compiler, Modelica Conference 2011

[12] Joel Andersson; Johan Åkesson; Moritz Diehl, CasADi - A symbolic package for automatic differentiation and optimal control, Proc. 6th International Conference on Automatic Differentiation, 2012.

[13] Houska, B., Ferreau, H.J., and Diehl, M. (2011). ACADO toolkit - an open source framework for automatic control and dynamic optimization. Optimal Control Applications & Methods, 32(3), 298-312.

[14] Functional Mock-up Interface: http://www.functional-mockup-interface.org/index.html

[15] Johan Åkesson. Optimica—An Extension of Modelica Supporting Dynamic Optimization. In 6th International Modelica Conference 2008. Modelica. Association, March 2008

[16] Interior Point OPTimizer (Ipopt) https://projects.coin-or.org/Ipopt

11 Appendix A

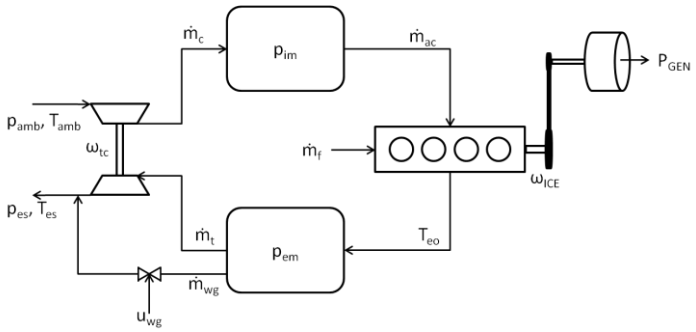

Figure 13. Diesel Engine Model

Powertrain model

$$\dot{\omega}_{ice} = \frac{P_{ice} - P_{gen}}{\omega_{ice}\, J_{genset}}$$

Intake System

- Compressor

$$\Pi_c = \frac{p_{im}}{p_{amb}}, \qquad \Pi_{c,max} = \left(\frac{\omega_{tc}{}^2\, R_c{}^2 \psi_{max}}{2 c_p T_{amb}} + 1\right)^{\frac{\gamma_a}{\gamma_a - 1}}, \quad \dot{m}_{c,corr} = \dot{m}_{c,corr,max}\sqrt{1 - \left(\frac{\Pi_c}{\Pi_{c,max}}\right)^2},$$

$$\dot{m}_c = \frac{\dot{m}_{c,corr}\, p_{amb}/p_{ref}}{\sqrt{T_{amb}/T_{ref}}}, P_c = \frac{\dot{m}_c c_{pa} T_{amb}\left(\Pi_c^{\frac{\gamma_a}{\gamma_a - 1}} - 1\right)}{\eta_c}, \quad \eta_c = c_c$$

- Intake manifold

$$\dot{p}_{im} = \frac{R_a T_{im}}{V_{is}}(\dot{m}_c - \dot{m}_{ci}), \quad T_{im} = T_{cool}$$

Cylinder

- Gas Flow

$$\dot{m}_{ci} = \frac{\eta_{vol} p_{im} \omega_{ice} V_D}{4\pi R_a T_{im}}, \quad \eta_{vol} = c_{vol}, \quad \dot{m}_f = \frac{10^{-6}}{4\pi} u_f \omega_{ice} \eta_{cyl}, \quad \lambda = \frac{\dot{m}_{ci}}{\dot{m}_f} \frac{1}{(A/F)_s}$$

- Torque

$$T_{ice} = T_{ig} - T_{fric} - T_{pump}, T_{pump} = \frac{V_D}{4\pi}(p_{em} - p_{im}), T_{ig} = \frac{u_f 10^{-6} n_{cyl} q_{HV} \eta_{ig}}{4\pi}, \qquad \eta_{ig} = \eta_{ig,ch}\left(1 - \frac{1}{r_c^{\gamma_{cyl} - 1}}\right)$$

$$T_{fric} = \frac{V_D}{4\pi} 10^5 \left(c_{fr1} \left(\frac{\omega_{ice} \frac{60}{2\pi}}{1000}\right)^2 + c_{fr2} \left(\frac{\omega_{ice} \frac{60}{2\pi}}{1000}\right)^2 + c_{fr3} \right)$$

- Temperature

$$\Pi_c = \frac{p_{em}}{p_{im}}, \quad \mathrm{q}_{in} = \frac{\dot{m}_f q_{HV}}{\dot{m}_f + \dot{m}_{ac}}, \quad x_p = \frac{p_3}{p_2} = 1 + \frac{q_{in} x_{cv}}{c_{va} T_{im} r_c^{\gamma_a - 1}}$$

$$T_{eo} = \eta_{sc} \Pi_e{}^{\left(1 - \frac{1}{\gamma_a}\right)} r_c{}^{(1 - \gamma_a)} x_p{}^{\left(\frac{1}{\gamma_a} - 1\right)} \left(\mathrm{q}_{in} \left(\frac{1 - x_{cv}}{c_{pa}} + \frac{x_{cv}}{c_{va}} \right) + T_{im}\, r_c{}^{(\gamma_a - 1)} \right)$$

Exhaust System

- Exhaust Manifold:

$$\dot{p}_{em} = \frac{R_e T_{em}}{V_{em}}\left(\dot{m}_{ci} + \dot{m}_f - \dot{m}_t - \dot{m}_{wg}\right), \quad T_{em} = T_{eo}$$

- Turbine

$$\Pi_t = \frac{p_{es}}{p_{em}}, \quad \Pi_t^* = max\left(\sqrt{\Pi_t}, \left(\frac{2}{\gamma_e - 1}\right)^{\frac{\gamma_e}{\gamma_e - 1}}\right), \quad \psi_t(\Pi_t^*) = \sqrt{\frac{2\gamma_e}{\gamma_e - 1}\left((\Pi_t^*)^{\frac{2}{\gamma_e}} - (\Pi_t^*)^{\frac{\gamma_e + 1}{\gamma_e}} \right)},$$

$$\dot{m}_t = \frac{p_{em}}{\sqrt{R_e T_{em}}} \psi_t A_{t,eff}, \quad P_t = \dot{m}_t c_{pe} T_{em} \eta_t \left(1 - \Pi_t^{\frac{\gamma_e - 1}{\gamma_e}}\right), \; \eta_t = c_t, \; J_{tc} \dot{\omega}_{tc} = \frac{P_t - P_c}{\omega_{tc}} - \omega_{fric} \omega_{tc}{}^2$$

- Wastegate

$$\Pi_{wg} = \frac{p_{es}}{p_{em}}, \quad \Pi_{wg}^* = max\left(\Pi_{wg}, \left(\frac{2}{\gamma_e + 1}\right)^{\frac{\gamma_e}{\gamma_e - 1}}\right), \quad \psi_{wg} = \sqrt{\frac{2\gamma_e}{\gamma_e - 1}\left(\left(\Pi_{wg}^*\right)^{\frac{2}{\gamma_e}} - \left(\Pi_{wg}^*\right)^{\frac{\gamma_e + 1}{\gamma_e}} \right)}, \qquad \dot{m}_{wg} = \frac{p_{em}}{\sqrt{R_e T_{em}}} \psi_{wg} u_{wg} A_{wg,eff}$$

Model Constants

Symbol	Description	Value	Unit
p_{amb}	Ambient pressure	1.011e5	Pa
T_{amb}	Ambient temperature	298.46	K
c_{pa}	Specific heat capacity of air, constant pressure	1011	J/(kg.K)
c_{va}	Specific heat capacity of air, constant volume	724	J/(kg.K)
γ_a	Specific heat capacity ratio of air	1.3964	-
R_a	Gas constant, air	287	J/(kg.K)
c_{pe}	Specific heat capacity of exhaust gas, constant pressure	1332	J/(kg.K)
γ_e	Specific heat capacity ratio of exhaust gas	1.2734	-
R_e	Gas constant, exhaust gas	286	J/(kg.K)
γ_{cyl}	Specific heat capacity ratio of cylinder gas	1.35004	-
T_{im}	Intake manifold temperature	300,6186	K
p_{es}	Pressure in exhaust system	1.011e5	Pa
$(A/F)_s$	Stoichiometric oxygen-fuel ratio	14.54	-
q_{HV}	Diesel heating value	42.9e6	J/kg

Model Parameters

Symbol	Description	Value	Unit
n_{cyl}	Number of cylinders	6	-
V_D	Engine displacement	0.0127	m^3
r_c	Compression ratio	17.3	-
J_{genset}	Inertia of the engine-generator	3.5	kgm^2
V_{is}	Volume of intake system	0.0218	m^3
R_c	Compressor radius	0.04	M
ψ_{max}	Max. compressor head parameter	1.5927	-
$\dot{m}_{c,corr,max}$	Max. corrected compressor mass flow	1.2734	-
η_c	Compressor efficiency	286	J/(kg.K)
η_{vol}	Volumetric efficiency	1.35004	-
$\eta_{ig,ch}$	Combustion chamber efficiency	0.6774	-
c_{fr1}	Friction efficiency	1.011e5	Pa
c_{fr2}	Friction efficiency	14.54	-
c_{fr3}	Friction efficiency	42.9e6	J/kg
η_{sc}	Non-ideal Seliger cycle compensation	1.054	-
x_{cv}	Ratio of fuel burnt during constant volume	0.4046	-
V_{em}	Volume of exhaust manifold	0.0199	m^3
J_{tc}	Turbocharger inertia	1.9662 e-4	kgm^2
ω_{fric}	Turbocharger friction	2.4358 e-5	kgm^2/rad
$A_{t,eff}$	Effective turbine area	9.8938 e-4	m^3
η_t	Turbine efficiency	0.7278	-
$c_{wg,1}$	Wastegate parameter	0.6679	-
$c_{wg,2}$	Wastegate parameter	5.3039	-
$A_{wg,eff}$	Effective wastegate area	8.8357 e-4	m^3