Optimization Library for Interactive Multi-Criteria Optimization Tasks

A. Pfeiffer
Institute of System Dynamics and Control, German Aerospace Center DLR, Oberpfaffenhofen
Andreas.Pfeiffer@dlr.de

Abstract

The commercial library Optimization 2.1 for interactive multi-criteria optimization tasks has been released along with Dymola 2013. The library offers several numerical optimization algorithms for solving different kinds of optimization tasks. User defined Modelica functions or models provide the basis for an interactive optimization process where the user keeps overview of complex multi-criteria optimization tasks that can take discrete parameters, several model operating points or trajectories into account. Computational performance of optimization runs can be significantly increased by parallel numerical integrations of the Modelica model on multi-core machines.

Keywords: Modelica; Optimization; Multi-Criteria; Trajectory Optimization; Parallel Simulation

1 Introduction

In principle, numerical optimization algorithms may be very powerful tools in engineering design processes like modeling, model validation or controller design. However, the fact that numerical algorithms are available does not necessarily encourage engineers to apply them. A user-friendly, easy handling of a well integrated optimization tool is necessary to make the advantages of automatic optimization available for non-experts. The presented Optimization library realizes this requirement in the Modelica world when working with Dymola [DS12b] or CATIA [DS12a].

1.1 Related Work

OMOptim [TNT+11] is an initiative to provide an open source optimization platform within OpenModelica. The emphasis of this platform is on using genetic algorithms, whereas interfacing gradient based optimization methods is planned for the future. The application is currently tailored to optimize model parameters of Modelica models. The library presented in the paper at hand provides a variety of different optimization tasks solved by several sophisticated local and global optimization algorithms.

In JModelica.org the Modelica extension Optimica is supported to solve dynamic optimization [AAG+10]. The approach in Optimica is different to the presented one, because Optimica defines additional Modelica language elements to describe Optimization problems directly in Modelica. Consequently, special compilers are needed to generate code for the optimization runs. JModelica.org supports collocation methods for dynamic optimizations. In the presented approach, (standard) Modelica models are compiled by Dymola. The well-proven numerical integration algorithms provided by Dymola are used in the optimization loop. Tailored graphical user interfaces support the user in several optimization tasks.

The library `Design.Optimization` [EOM+05] is the forerunner of the presented library. For the new version the library has been completely reimplemented with many new features. The new concept of different optimization *tasks* is enhanced by specialized graphical user interfaces (GUIs). The primary concept and the code of numerical algorithms for solving multi-criteria optimization problems are based on [JBL+02].

1.2 Optimization Problem Formulation

The multi-criteria optimization problems considered in the Optimization library can be formulated as follows:

$$\min_{p \in B} f\left(\mathrm{diag}(d_1)^{-1} c_1(p)\right)$$

$$\text{such that } c_2(p) \le d_2,\ c_3(p) = d_3$$

$$\text{with } c = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix},\ d = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} \text{ and}$$

$$f = \begin{cases} \max & \dots \text{ maximum of criteria values, or} \\ \|\cdot\|_2^2 & \dots \text{ sum of squared criteria values, or} \\ \|\cdot\|_1 & \dots \text{ sum of absolute criteria values.} \end{cases}$$

Free parameters p (e.g. some Modelica parameters in models) to be varied during the optimization process are called *tuner parameters* or tuners. The first part of the *criteria* vector c represents the objectives of the optimization (e.g. the overshoot of a variable in a model). The goal is to minimize all these objectives. The criteria components that define inequality or equality constraints are optional. They enable formulation of conditions on some criteria components if needed. The demand values d serve as reciprocal scaling factors of the criteria. They enable a different weighting of the individual criteria to be minimized. The tuner box B defines minimum and maximum values for each tuner parameter, thus limiting the range in which the tuner parameters can be varied.

For multi-criteria optimization problems a whole set of optimal solutions generally exists: the *Pareto* optimal solutions [E05]. For these solutions it is not possible to decrease one of the components of the objectives vector c_1 without increasing another one. It means the different criteria conflict each other. Finding all Pareto optimal solutions requires very high computational effort. In many cases it is sufficient to transform a multi-criteria problem to an optimization problem with a scalar objective function f. This approach is applied to the Optimization library with the maximum of the objectives, the sum of the squares of the objectives or the sum of the absolute values of the objectives.

1.3 Discrete Tuner Parameters

Discrete tuners are tuners that only have a finite number of values to be set. Examples for such tuners are configuration parameters that represent different topologies, e.g. switching modes in networks.

Three possibilities are available to define discrete tuners in the Optimization library. At the level of each tuner parameter, one can define the number of equidistant discrete values within the interval [min, max]. Only these points can be selected by the optimization algorithm to set the tuner value.

name	min	max	equidistant	discreteValues
"Kf"	-10	0	0	{-7.8, -2.5, -9.3}
"Ki"	-10	0	6	fill(0, 0)
"Kq"	0	10	0	fill(0, 0)

Figure 1: Discrete values for tuner parameters in the optimization setup GUI.

For example, setting *equidistant* = 6 for *min* = −10, *max* = 0 enables the values −10, −8, −6, −4, −2, 0 for the tuner *Ki* in Figure 1. The second possibility to define discrete tuners is to give a Modelica vector of values that can be set to the tuner parameter, e.g. *discreteValues* = {−7.8, −2.5, −9.3} for tuner parameter *Kf*.

At the level of all tuner parameters a list of values of discrete tuner parameter sets can be defined in a matrix. Each column corresponds to a tuner parameter, see Figure 2. It is possible to simply import the matrix from and export it to file. This feature allows to automatically evaluate a long list of tuner values generated by a separate tool.

Rows 2

	Kf	Ki	Kq
1	-7.8	-2.54	6.1
2	-3.4	-8	2.883

Figure 2: Discrete tuner matrix in the optimization setup GUI.

1.4 Optimization and Evaluation Algorithms

The following numerical optimization algorithms are available in the Optimization library: *Sequential Quadratic Programming (SQP)*, *Quasi Newton (BFGS) method*, *Pattern Search*, *Simplex Method* and *Genetic Algorithm*. SQP and BFGS algorithms rely on derivatives of the criteria with respect to the tuner parameters and have good convergence properties for smooth optimization problems. Pattern Search and Simplex Method are more robust against nonsmoothness but generally need more criteria evaluations to converge. Genetic Algorithm is the only approach to find a global solution whereas the others are local convergent methods. Further details to the implemented optimization algorithms can be found in [J11].

All the optimization algorithms have in common that they work more or less sequentially. Most values for tuners depend on criteria values of previous evaluations. So, there are limited possibilities to parallelize the (time consuming) evaluations of criteria. In contrast to these algorithms, pure evaluation methods independently set tuner values at the beginning of the process. Of course, constraints fulfillment is therefore not guaranteed.

Two evaluation methods are implemented in the Optimization library: *Random Search* and *Systematic Tuner Variation*. Random Search takes uniformly distributed random values between minimum and

DOI 10.3384/ecp12076669

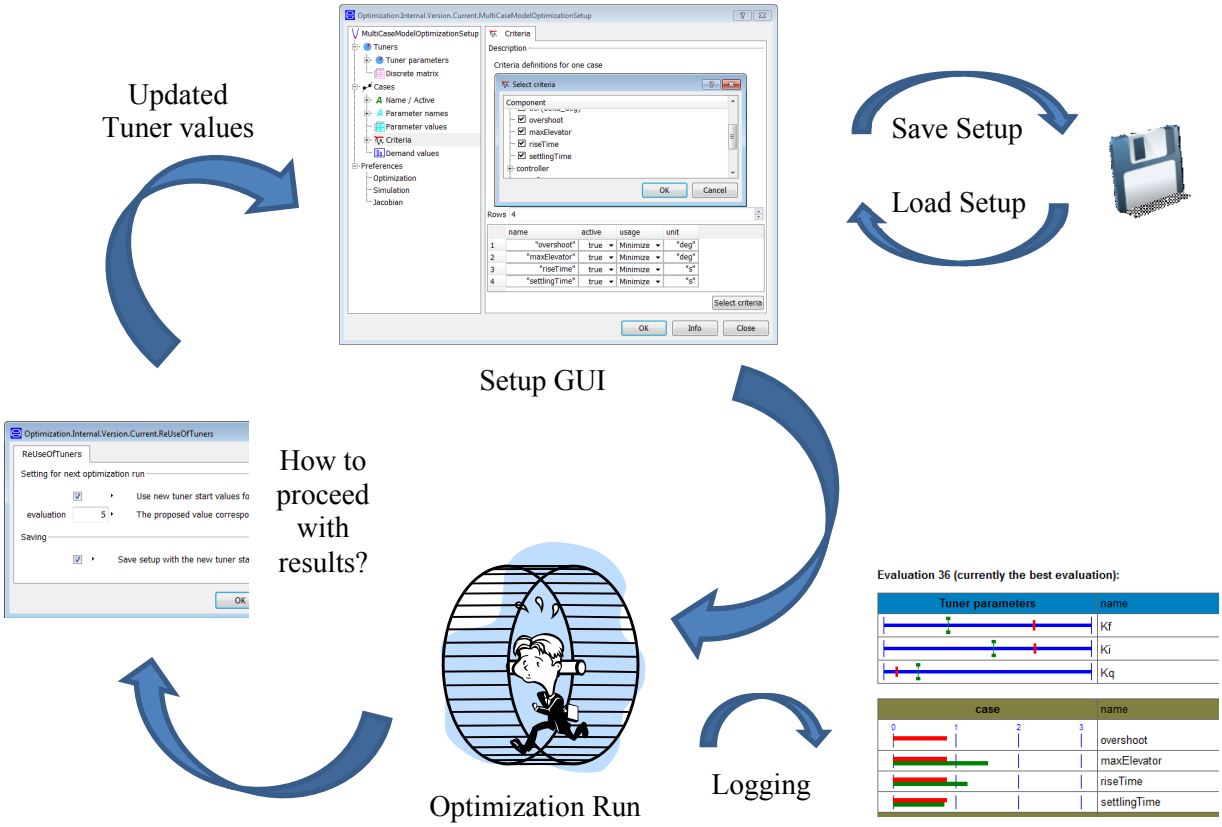


Figure 3: Optimization process for GUI supported Optimization tasks.

maximum of each tuner parameter. Systematic Tuner Variation is based on discrete tuners. If the discrete tuner matrix is activated, the corresponding tuner values are used row by row of the matrix. If the discrete tuner matrix is not used, all combinations of equidistant or given discrete tuner values are the basis for the criteria evaluations. For the example in Figure 1 there are $3 \cdot 6 \cdot 1 = 18$ different sets of discrete tuner values.

Table 1: Overview of the optimization and evaluation algorithms with their capability to support continuous and / or discrete tuners.

Algorithm	Continuous	Discrete	Mixed
SQP	✓		
BFGS	✓		
Pattern Search	✓		
Simplex Meth.	✓		
Genetic Alg.	✓	✓	✓
Random Search	✓	✓	✓
Systematic Var.		✓	

Most of the interfaced algorithms are designed to handle continuous tuner parameters. It means that the tuner values can be arbitrarily varied inside a given interval. Table 1 gives an overview which algorithm also supports discrete tuners or problems with both continuous and discrete tuner parameters.

1.5 Optimization Process

For each of the GUI supported optimization tasks the process to configure the task, to start the optimization and to handle the results is nearly the same and is discussed in the following by means of Figure 3.

By starting the corresponding setup GUI for an optimization task, the user gets a hierarchical list of settings to be configured. For each task one has to specify tuners and criteria depending on the type of the task. For optimization tasks requiring a model, additional settings for the model simulation have to be provided. All the information given in the setup GUI can be saved to a Modelica file. The file contains a call starting the corresponding setup GUI filled with the saved entries. Of course, the textual file can be edited before starting the setup GUI. So, loading an optimization setup is simply running the Modelica function generated when saving the setup.

After the optimization setup is configured, the optimization run is started. During the run the current

solutions may be logged to an HTML-file, also interactively displayed in Dymola's Command window. The logging has two intentions. Firstly, the history of a complete optimization run can be reconstructed. Secondly, optimization runs may last hours or days. It is very important in these cases to have a feedback, what the optimization algorithm is currently doing, to quickly react on non-intended intermediate optimization results. The HTML-logging lists the current tuner and criteria values and visualizes them in different colors in comparison to values at the beginning of the optimization.

Beside the HTML-logging there is a logging of pure numeric data to be processed after the optimization run if it is necessary. After the optimization run is finished, the user is asked how he wants to proceed. There is the possibility to reset the tuner parameters by values generated by the optimization process. For example, one can select the tuner values of the best evaluation (= solution) of the optimization run. These settings can be used to proceed the optimization process with different settings, e.g. using another optimization algorithm. In any case, after an optimization run the setup GUI is displayed (with possibly changed tuner values) and can be configured as described above.

2 Function based Optimization

Two optimization tasks based on user-defined Modelica functions are described. Whereas *Function Optimization* is an interactive task, *Realtime Optimization* is designed to be called in model equations during the numerical integration.

2.1 Function Optimization

The task *Function Optimization* is designed for the most general case of an optimization problem in Modelica. The user has to provide a Modelica function that evaluates the criteria (and constraints) functions. Optionally, a user-defined function for the evaluation of the Jacobian matrix can be incorporated. The task can be used for simple academic optimization problems resulting in a criteria function of a few lines of code, or for every complex optimization problem including simulations and linearizations of several models. The user has to program and control the simulations and linearizations by available functions in Modelica and Dymola.

The main part of a function optimization problem is to program the criteria function in Modelica. The criteria function returns a criteria vector depending on the tuner values. The criteria can either be parts of the optimization's objective function or be one of the constraints of the optimization problem. A criteria function has to have defined interface variables from the partial function `PartialCriteriaVariables`:

```
partial function PartialCriteriaVariables
  input Real tuners[:];
  output Real criteria[:];
end PartialCriteriaVariables;
```

A typical criteria function looks like the following prototype. One can add own input variables to the criteria function. The values for these inputs have to be declared in the name of the criteria function in the setup, e.g. "`myCriteriaFunc(myVar=<value>)`".

```
function myCriteriaFunction
  extends PartialCriteriaVariables;
  input <AnyType> myVar;
algorithm
  criteria := ...(tuners, myVar);
end myCriteriaFunction;
```

Gradient based optimization algorithms (SQP, BFGS) need the Jacobian matrix of the criteria with respect to tuner parameters. The user can select between symmetric finite differences and forward difference quotients. There is also the possibility to program the Jacobian matrix by oneself, e.g. if one knows the analytical Jacobian matrix. The interface variables are defined in the following partial function:

```
partial function PartialJacobianVariables
  input Real tuners[:];
  input PartialCriteriaVariables CritFunc;
  output Real Jacobian[:,size(tuners,1)];
end PartialJacobianVariables;
```

To a Jacobian function one can also add own input variables, see the following prototype of a typical Jacobian function:

```
function myJacobianFunction
  extends PartialJacobianVariables;
  input <AnyType> myVar;
algorithm
  Jacobian := ...(tuners, myVar);
end myJacobianFunction;
```

2.2 Realtime Optimization

Realtime Optimization is in some way different to the other optimization tasks. Realtime Optimization provides the framework for an optimization function to be called during the numerical integration of a

DOI 10.3384/ecp12076669

model. A possible application of this optimization task is a discrete controller that solves an optimization problem to predict new controller values every sample time. The optimization problem itself is very similar to that of *Function Optimization*. User defined functions for criteria evaluation and optional functions for the Jacobian matrix provide the basis for the optimization task. Because Realtime Optimization is active during a simulation many times, there is no GUI support for it. A Modelica model calling the optimization function typically has the following structure:

```
model myModel
  Real resultTuners[...];
  Real resultCriteria[...];
  KernelProblem problem(...);
  ...
equation
  ...
  when sample(0, 0.1) then
    (resultTuners,resultCriteria) =
      run(problem, CriteriaFunc =
                function myCriteriaFunction);
  end when;
  ...
end myModel;
```

At each sample point the optimization run is started by the function *run*. The optimization problem is described by the record *problem* that includes the tuner and criteria definitions as well as the optimization options. The approach is currently used in model predictive control for an electric vehicle [K10].

3 Model based Optimization

This section deals with optimization tasks based on the numerical integration of a Modelica model. The computation of the optimization criteria is part of the numerical integration. Because model simulation is the main application of dealing with Modelica models, the following optimization tasks and their features may be considered as the core of the Optimization Library.

3.1 Criteria Library

To support all model based optimization tasks the sub-library `Optimization.Criteria` is part of the whole package. The library (see Figure 4) provides models that compute typical criteria from time dependent model variables. The collection of criteria models helps the user to prepare his system model for conducting an optimization on it. For Real signals the following models are included: minimum, maximum, mean value, moving average and integral norm. In Figure 5, some examples are illustrated. Computing deviations between two signals may be handled by the corresponding criteria models. In the field of controller design typical design criteria are *overshoot*, *rise time* and *settling time*. Each of them is represented by a corresponding criteria model. Some of the criteria models require the input signals to be differentiated.

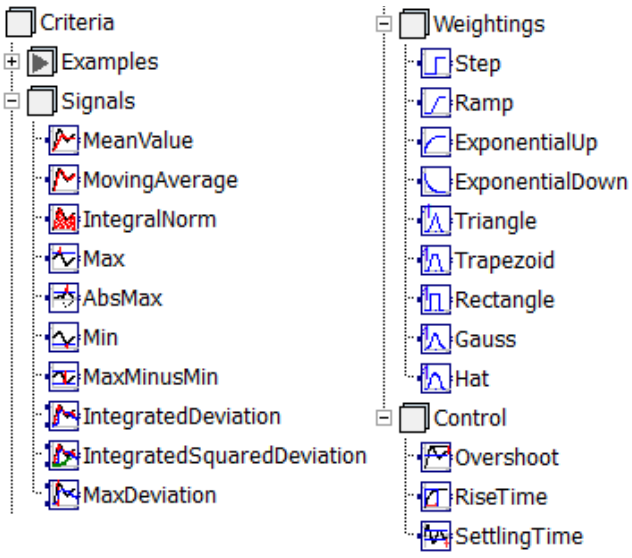


Figure 4: Criteria library.

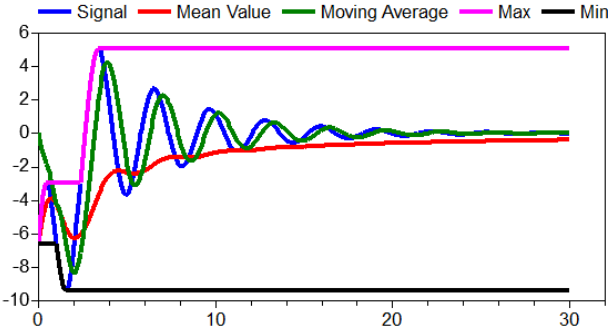


Figure 5: Typical signals of criteria models.

In some cases only parts of the whole time interval shall be used to compute a certain criterion, or some time areas shall be weighted more than others. For such needs several weighting models are provided: *Step*, *Ramp*, *Triangle*, etc.

3.2 Model Optimization

The task *Model Optimization* is designed to optimize parameters of a Modelica model. The user can select from a list of model parameters to define tuners, see Figure 6. Also it is possible to get a list of all time depending model variables to be selected for criteria

variables. The value of the criterion is defined by the final value of the criterion variable at the end of the integration interval.

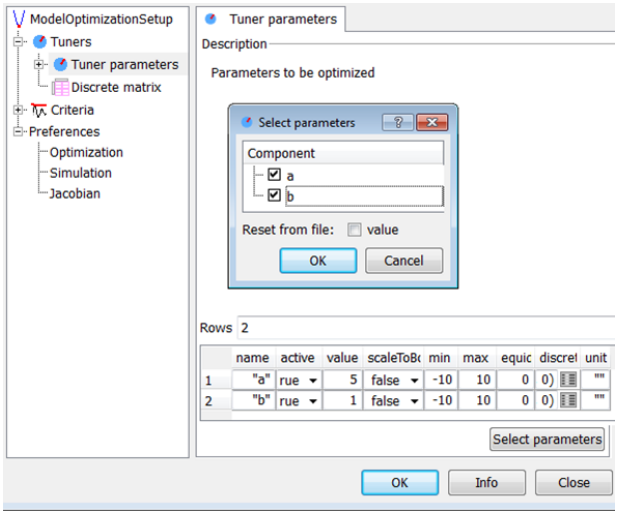


Figure 6: GUI for selecting model parameters as tuners.

The simulation of the model to be optimized has to be specified by usual simulation preferences like start and stop time or the numerical integration algorithm. Additionally, different modes to accelerate the numerical integration of the model equations are implemented, see Section 3.5.

A typical application of Model Optimization is the identification of model parameters by comparing simulation results and corresponding measurements from a test bench. A further application is well known in the field of controller synthesis. To improve the controller performance automatic optimization is applied to the system model.

3.3 Multi Case Model Optimization

Multi Case Model Optimization is an extension of the task *Model Optimization* and has its origin in the field of model based controller design. Most controllers do not only have to guarantee performance and stability of a system in one, but in several operating points. The optimization of the controller parameters includes the simulation of a system model in different operating points that are characterized by different values of special model parameters, the *case parameters*. These model parameters are disjoint with the tuner parameters and are not varied by the optimization algorithm. The different model simulations that are defined by the case parameters are called *cases*. In Figure 7 main parts of the corresponding task setup GUI are shown.

Each case should have a name to distinguish it from the other cases. In Figure 7 there are three cases: *nominal*, *worstOvershoot* and *worstSettlingTime*. The case parameters (e.g. *Ma*, *Md*, …, *Zd*) can be selected from a list of all independent model parameters. For each case every case parameter gets a value, see the matrix in Figure 7. The model is simulated with these case parameter values for each case. The criteria of the optimization task are similarly specified as for the task Model Optimization.

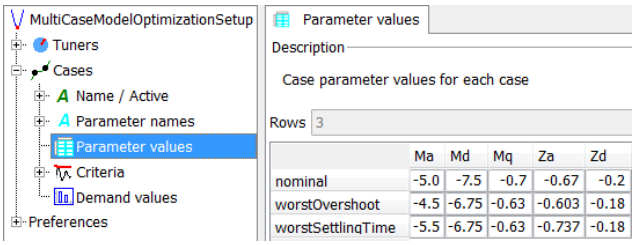


Figure 7: Optimization setup GUI for Multi Case Model Optimization.

In summary, every case contributes to the overall criteria vector of the optimization problem, see Figure 8 for an example. Depending on the objective function type all these criteria values are combined to the objective function value. In the example the value is the maximum of all criteria values: *riseTime* for the case *worstSettlingTime*.

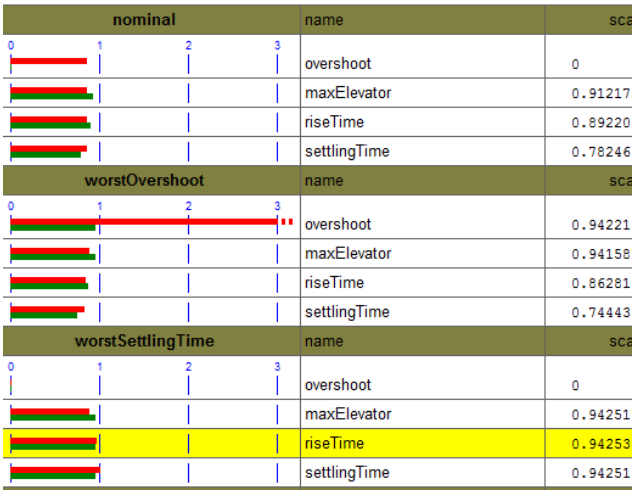


Figure 8: Logging of multi case criteria.

3.4 Trajectory Optimization

Problems of Optimal Control arise in different fields of applications. The goal is to minimize an objective functional with respect to one or more time dependent control trajectories. Various constraints are typical for optimal control problems. Dynamic model equations appear in most of the problems in technical

DOI 10.3384/ecp12076669

applications. Consequently, an optimization task *Trajectory Optimization* is provided in the library.

There are many techniques [B01] to numerically solve an infinite dimensional optimal control problem. In the Optimization library the solution procedure is according to the task *Model Optimization*. It means that tuners are varied by the optimization algorithm and for each computation of the criteria a model simulation is performed. This *Single Shooting Technique* is based on a finite dimensional optimization problem approximating the original problem.

The control trajectories are approximated by *B-splines* of degree k. The number of samples N and the interpolation degree k define the construction of a B-spline as control trajectory [DH02]. The B-spline has N equidistant knots on the time interval the spline is defined (normally this is the integration interval of the model). Further there are $N + k - 1$ *de Boor control points* that parameterize the spline. A spline $s(t)$ is a piecewise polynomial function between the knots. The individual polynomials have at most the degree k. The polynomials are appended such that the complete spline is $k - 1$ times continuously differentiable on the whole interval it is defined. Because a B-spline is contained in the convex hull of its de Boor points m, the box constraints $u_{\mathrm{Min}} \le m \le u_{\mathrm{Max}}$ (for lower and upper bounds u_{Min}, u_{Max}) are valid for the whole spline function: $u_{\mathrm{Min}} \le s(t) \le u_{\mathrm{Max}}$. Therefore, the control points m are selected as tuners to be varied by the optimization algorithm.

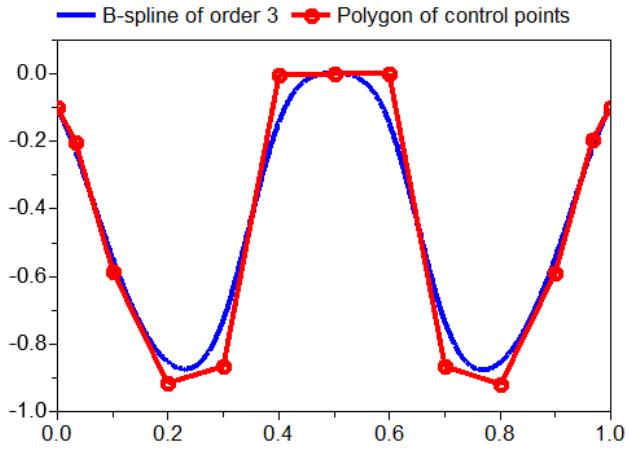


Figure 9: Polygon of B-spline control points and corresponding B-spline trajectory.

In Figure 9 the polygon of 13 control points and the corresponding B-spline of degree 3 ($N = 11,\ k = 3$) are shown. The control points correspond to the 13 time values 0.0, 0.033, 0.1, 0.2, …, 0.8, 0.9, 0.967, 1.0. Additional to the given time grid 0.0, 0.1, …, 1.0 there are two values at the boundaries: 0.033 and 0.967. They represent the free boundary conditions of the B-spline.

The optimization setup for Trajectory Optimization includes the selection of model inputs that represent the control trajectories. For these trajectories the number of sample points N and the interpolation degree k has to be specified by the user. Any starting trajectory may be provided in a separate file. An example using the Trajectory Optimization task is given in Section 4.

3.5 Parallel Numerical Integration

Because the numerical integration of model equations normally is the most time intensive part of any model based optimization tasks, several techniques are applied to reduce the computation time of the numerical integration inside the optimization loop. The default case is a sequential execution of the numerical integration runs by calling Dymola's simulation executable for each new set of model parameters. We call it *single* simulation technique.

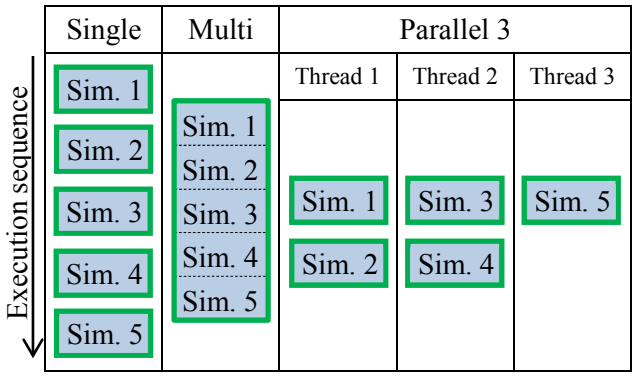
An optimized version of sequential integration runs is provided by Dymola. The executable is started only one time and independent model parameter values are sequentially read from file and processed by the numerical integration. Especially for many simulation runs with very short elapsed real times for one model simulation, this *multi* simulation approach accelerates the numerical integration in summary, because process overhead is avoided.

Independent simulation runs of a model may be executed in *parallel*. Especially for multi-core machines this may reduce the computation time of the whole optimization run. In the Optimization library the simulation runs are parallelized in different threads by calling several copied simulation executables in an OpenMP program. OpenMP is a software interface for shared-memory parallel programming on different platforms. It is supported by many computer hardware and software vendors [CJP08]. For parallel simulations the user can specify the number of threads up to the double of the number of available cores. Table 2 shows execution sequences for different simulation modes in principle.

To measure the acceleration in computation time, a test is performed for different simulation modes. The model `Electrical.Analog.Examples.Rectifier`

from the Modelica Standard Library 3.2 is simulated 1000 times with identical parameter values. To increase the elapsed real time of one numerical integration run, the stop time of the integration is increased. The test is executed on a PC with an Intel Xeon X5550 quad-core processor (2.67 GHz) with activated hyper-threading.

Table 2: Execution sequence for single, multi and parallel (with 3 threads) simulations.



In Figure 10 the results of the test are illustrated. Depending on the execution time for one model simulation, the speed factor with respect to the single simulation technique is plotted for multi and parallel simulations. Parallel simulations are performed with 2, 4 and 8 threads. For very fast model simulations the multi simulation approach is clearly superior. Compared to single simulation the multi simulation is up to 4 times faster although no parallelization technique is applied. The parallel execution of 1000 model simulations results in maximum speed factors of 1.9, 3 and 4 for 2, 4 and 8 threads. These maximum factors are reached if the execution time for one model simulation is greater than 1 second. Below this bound the speed factor is decreasing due to the process overhead. For machines with many cores the limiting influence for parallelization is probably memory access.

An important assumption for the performance test is the independency of all evaluated model parameter values. The Optimization library supports two algorithms that fulfill this assumption: Random Search and Systematic Tuner Variation (see Section 1.4). For these algorithms the tuner values of all evaluations may be determined before running any simulation, therefore full parallel evaluations are possible. So, speed factors as shown in Figure 10 can be reached.

Accelerating the computation time in nonlinear optimization by parallel evaluations of the criteria has been investigated since several years, e.g. see [LAS97]. The optimization algorithms of the Optimization library partially support parallel criteria evaluations. During an optimization run there are both evaluations of the criteria that can be parallelized and such ones that cannot be parallelized. The evaluation of numerical Jacobian matrices typically needs the most computation time for optimization runs with SQP and BFGS methods. Consequently, the Optimization library supports computing numerical Jacobian matrices by parallel model simulations. It also supports parallel criteria evaluations of multi case optimization tasks (see Section 3.3). The simulation runs of a model with different case parameter values are independent and therefore can be computed in parallel. It is planned to support parallel model simulations for independent criteria evaluations of the genetic algorithm.

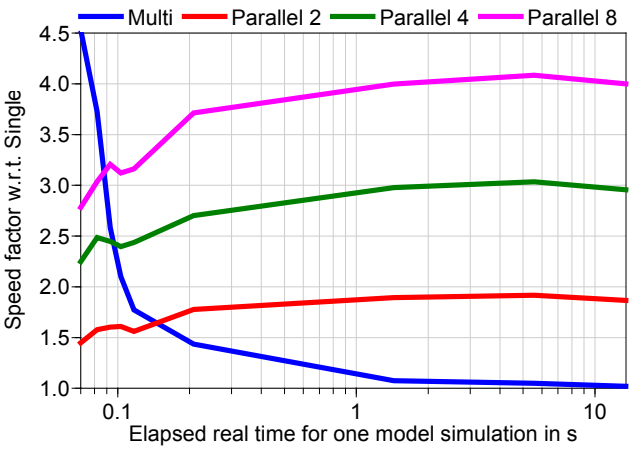


Figure 10: Speed factors for different simulation modes.

Depending on the used optimization algorithm, the Modelica model and the number of tuners, the speed factor for a *complete optimization run* differs. On the test machine a factor of 3 in computation time has been observed for optimization examples using a model that needs more than 1 second of elapsed real time per simulation, see Section 4.3 for an example.

4 Application Example

In [EOM+05] the full robot model of the Modelica standard library is used to demonstrate a multi case optimization for controller design. Of course, the current version of the Optimization library can still handle this kind of optimization task (see Section 3.3). In the following a trajectory optimization setup for the robot model is presented to find reference trajectories for the robot's movements from one point to another point in space.

DOI 10.3384/ecp12076669

4.1 Robot Model

The robot model (see Figure 11) mainly consists of a 3-D mechanical structure model and 6 axis models including electrical motors, controllers and mechanical components of the axes (gear and friction). The reference trajectories for the angles and velocities of the axes are provided by a separate path planning model. The path planning is based on an algorithm that finds trajectories for the fastest movement for a given start position α to a given end position β under kinematical constraints. The constraints are defined by the maximum velocity and the maximum acceleration of the axis movements.

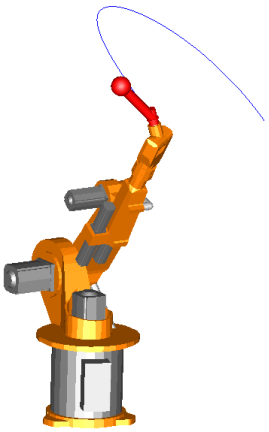


Figure 11: Animation of robot model from Modelica Standard Library.

The drawback of the path planning model is that the available maximum torque of the electrical motors is not considered. We may include them in the path planning by solving a trajectory optimization problem with the inverse dynamics model [R11] using the Optimization library. For these purposes we have to adapt the robot model. The motor, controller and friction model of the axes are removed. The rotational power train of each axes is driven by a signal based torque source. The non-causal approach of Modelica automatically leads to the inverse dynamics model when giving input signals for the robot positions [TOB01].

4.2 Trajectory Optimization Problem

The goal of the trajectory optimization problem is to find movements for the axes' angles $q(t)$. The movement from the start angles α to the end angles β should be as fast as possible under the constraints that the maximum velocity and the maximum motor torques are bounded by given values. Additionally, the angular accelerations shall be zero at the start and the end position to avoid oscillations for the controlled robot using the computed paths as reference motion.

The mathematical formulation is as follows:

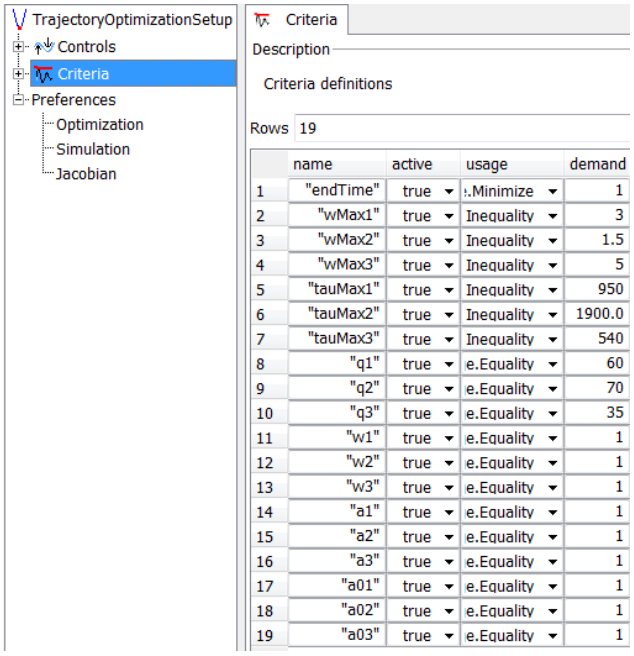
$$\min_{t_{End},\ u(t)} t_{End} \qquad \text{w. r. t.}$$

$$q(0) = \alpha, \quad q(t_{End}) = \beta, \quad \ddot{q} = u,$$

$$\dot{q}(0) = \dot{q}(t_{End}) = \ddot{q}(0) = \ddot{q}(t_{End}) = 0,$$

$$|\dot{q}(t)| \le v_{Max}, \quad |\tau(t)| \le T_{Max} \quad \text{for} \quad t \in [0, t_{End}].$$

In our investigations we only consider the main axes 1, 2 and 3. Axes 4, 5 and 6 are fixed and do not move. Reasonable values for the maximum angular velocities v_{Max} and the maximum torques T_{Max} can be found in [OT88]. We use $v_{Max} = (3, 1.5, 5)\ rad/s$ and $T_{Max} = (950, 1950, 540)\ Nm$ for axis 1, 2 and 3. The adapted robot model is prepared in such a way that $q(0) = \alpha$, $\dot{q}(0) = 0$ is inherently fulfilled. The trajectory $q(t)$ is implicitely defined by B-Splines for the controls $u(t) := \ddot{q}(t)$. The trajectories for $\dot{q}(t)$ and $q(t)$ are automatically computed in the robot model by the numerical integration algorithm during the simulation of the model.



	name	active	usage	demand
1	"endTime"	true	Minimize	1
2	"wMax1"	true	Inequality	3
3	"wMax2"	true	Inequality	1.5
4	"wMax3"	true	Inequality	5
5	"tauMax1"	true	Inequality	950
6	"tauMax2"	true	Inequality	1900.0
7	"tauMax3"	true	Inequality	540
8	"q1"	true	e.Equality	60
9	"q2"	true	e.Equality	70
10	"q3"	true	e.Equality	35
11	"w1"	true	e.Equality	1
12	"w2"	true	e.Equality	1
13	"w3"	true	e.Equality	1
14	"a1"	true	e.Equality	1
15	"a2"	true	e.Equality	1
16	"a3"	true	e.Equality	1
17	"a01"	true	e.Equality	1
18	"a02"	true	e.Equality	1
19	"a03"	true	e.Equality	1

Figure 12: Criteria of robot path planning in Optimization setup GUI.

The trajectory optimization setup (see Figure 12) consists of three input controls $\ddot{q}(t)$ and the free parameter t_{End}. The criterion to be minimized is the end time t_{End}, whereas 6 (= 2 · 3 axes) inequality constraints are defined for $\dot{q}_{Max}$ and τ_{Max}. The robot

model includes criteria models (see Section 3.1) to compute the absolute maxima $\dot{q}_{Max}$ of $\dot{q}(t)$ and τ_{Max} of $\tau(t)$. There remain 12 equality constraints for $q(t_{End})$, $\dot{q}(t_{End})$, $\ddot{q}(0)$ and $\ddot{q}(t_{End})$.

The advanced feature to handle a free end time t_{End} for the trajectory optimization is implemented and will be available in the next release of the Optimization library.

4.3 Trajectory Optimization Results

We set the start trajectories for $\ddot{q}(t)$ equal to 0 and choose $t_{End} = 5$ at the beginning of the optimization. These start conditions lead to violated optimization constraints for $q(t_{End})$. The SQP algorithm succeeds in finding input functions $\ddot{q}(t)$, such that all constraints are fulfilled. Important for SQP is a high accuracy of the criteria, therefore we set the error tolerance of the integration to 10^{-12}. The error tolerance for the solution of SQP is set to 10^{-6}.

Depending on the number N of sample points for the B-splines, different solution are found, see columns 1 and 2 in Table 3. The degree k of the polynomials is always set to $k = 3$. We tested the developed parallelization techniques (see Section 3.5) for this benchmark problem. In Table 3 the computation times for the single simulation approach are documented. Further, the speed factors using parallel simulations with 2, 4 and 8 threads are given. Since the computation of the numerical Jacobian matrix dominates the overall computation time, speed factors of pure independent simulations (compare Figure 10) can be reached for 2 and 4 parallel threads. The optimization with 8 threads is faster than using 4 threads, but the difference is smaller than in Figure 10.

Table 3: Results of the trajectory optimization with different number N of sample points for the B-splines.

N	t_{End}	Single	Parallel speed factors		
		Elapsed time	2 threads	4 threads	8 threads
5	1.60 s	30 min	1.85	2.95	3.15
8	1.48 s	150 min	1.96	3.11	3.38
10	1.42 s	251 min	1.95	3.12	3.34
20	1.40 s	908 min	2.00	3.33	3.58
30	1.40 s	1228 min	2.02	3.37	3.68

Figure 13 illustrates the solutions $q_2(t)$, $\dot{q}_2(t)$ and $\ddot{q}_2(t)$ for $N = 5, 8, 10$ and 20. It is obvious, that the velocity constraint $\dot{q}_2 \leq v_2 = 1.5\, rad/s$ is an active constraint. In Figure 14 it can also be seen, that the motor torque is inside the demanded ranges. The trajectory for the torque of axis 3 hits the border lines several times.

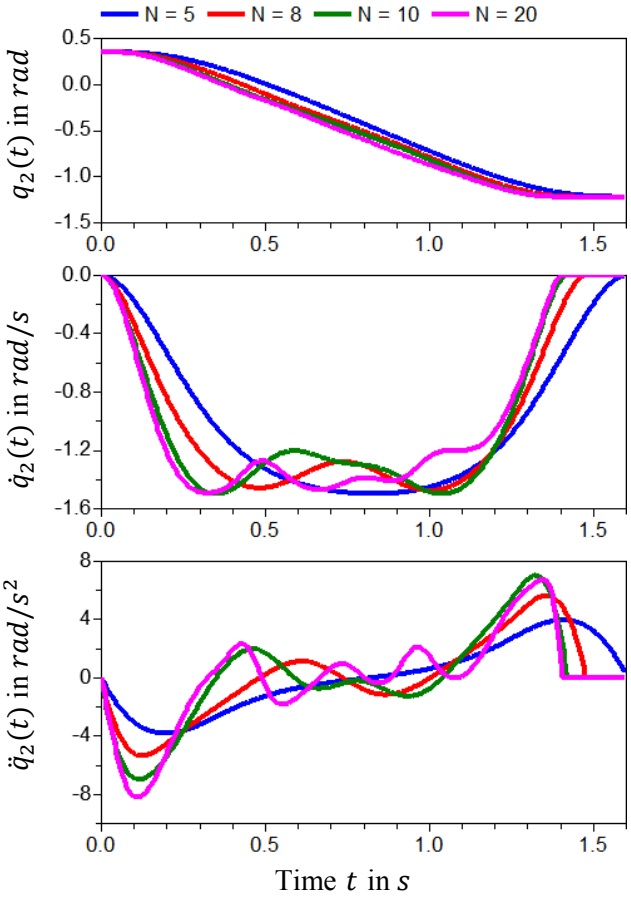


Figure 13: Result trajectories with different number N of sample points for the B-splines.

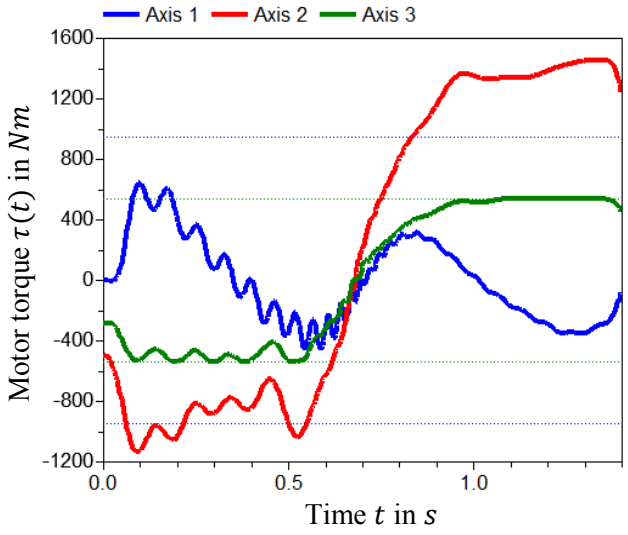


Figure 14: Motor torque for different axes. The optimization solution is computed with $N = 20$ sample points.

DOI 10.3384/ecp12076669

5 Conclusions

A library for solving interactive optimization tasks is presented. Both function and different model based optimization tasks are available to support the engineer in improving his system design by sophisticated numerical optimization algorithms. Additionally, optimization runs may be accelerated by automated parallel model simulations on multi-core machines. Version 2.1 of the Optimization library is available along with the release of Dymola 2013.

6 Acknowledgement

The support of H.-D. Joos, M. Otter, M. Reiner and K. Schnepper (all members of DLR Institute of System Dynamics and Control) in developing the Optimization library and the application example is gratefully appreciated. Improvements of Dymola by Dassault Systèmes AB to support the Optimization library are acknowledged. Partial financial support of DLR by BMBF (BMBF Förderkennzeichen: 01IS07022F) for this work within the ITEA2 project EUROSYSLIB [E12] is highly appreciated. Also, the constructive comments of the anonymous paper reviewers are appreciated.

References

[AAG+10] Åkesson J., Årzén K.-E., Gäfvert M., Bergdahl T. and Tummescheit H.: *Modeling and Optimization with Optimica and JModelica.org – Languages and Tools for Solving Large-Scale Dynamic Optimization Problems*. Computers and Chemical Engineering, Vol. 34, Issue 11, pp. 1737-1749, 2010.

[B01] Betts J. T.: *Practical Methods for Optimal Control Using Nonlinear Programming*. SIAM Press, Philadelphia, Pennsylvania, USA, 2001.

[CJP08] Chapman B., Jost G. and van der Pas R.: *Using OpenMP, Portable Shared Memory Parallel Programming*. The MIT Press, Cambridge, Massachusetts, London, England, 2008.

[DH02] Deuflhard P. and Hohmann A.: *Numerische Mathematik I. Eine algorithmisch orientierte Einführung*. 3. Auflage, de Gruyter, Berlin, Germany, 2002.

[DS12a] Dassault Systèmes AB: *CATIA*. www.3ds.com/products/catia.

[DS12b] Dassault Systèmes AB: *Dymola*. www.dymola.com.

[E05] Ehrgott M.: *Multicriteria Optimization*. Second Edition, Springer, Berlin, Heidelberg, Germany, 2005.

[E12] EUROSYSLIB, ITEA2 06020, www.eurosyslib.com.

[EOM+05] Elmqvist H., Olsson H., Mattsson S. E., Brück D., Schweiger C., Joos D. and Otter M.: *Optimization for Design and Parameter Estimation*. Proc. of 4th International Modelica Conference, pp. 255-266, Hamburg, Germany, 2005.

[J11] Joos H.-D.: *MOPS - Multi-Objective Parameter Synthesis, User's Guide V6.2*. Institute of Robotics and Mechatronics, DLR Oberpfaffenhofen, Germany, 2011.

[JBL+02] Joos H.-D., Bals J., Looye G., Schnepper K. and Varga A.: *A Multi-Objective Optimisation based Software Environment for Control System Design*. Proc. IEEE International Conference on Control Applications, pp. 7-14, Glasgow, Scotland, Sept. 18-20, 2002.

[K10] Köppern J.: *Integrierte Fahrzeugregelung durch einen hybriden Ansatz aus inversem Modell und modellprädiktiver Optimierung*. GMA-Fachausschuss 1.40 "Theoretische Verfahren der Regelungstechnik", Salzburg, Austria, 2010.

[LAS97] Lewis A, Abramson D. and Simpson R., *Parallel non-linear optimization: Towards the design of a decision support system for air quality management.* Proc. of IEEE Supercomputing 97, San Jose, USA, 1997.

[OT88] Otter M. and Türk S.: *The DFVLR Models 1 and 2 of the Manutec r3 Robot*. DFVLR-Mitteilung 88-13, Institut für Dynamik der Flugsysteme, DLR Oberpfaffenhofen, Germany, 1988.

[R11] Reiner M.: *Modellierung und Steuerung von strukturelastischen Robotern*. Ph.D. thesis, University of Technology, Munich, 2011.

[TOB01] Thümmel M., Otter M. and Bals J.: *Control of Robots with Elastic Joints based on Automatic Generation of Inverse Dynamics Models*. Proc. of IROS, pp. 925-930, Maui, Hawaii, USA, 2001.

[TNT+11] Thieriot H., Nemer M., Torabzadeh-Tari M., Fritzson P., Singh R. and Kocherry J. J.: *Towards Design Optimization with OpenModelica Emphasizing Parameter Optimization with Genetic Algorithms*. Proc. of 8th International Modelica Conference, pp. 756-762, Dresden, Germany, 2011.

DOI
10.3384/ecp12076669