Industrial application of optimization with Modelica and Optimica using intelligent Python scripting

K. Dietl[a], S. Gallardo Yances[a], A. Johnsson[b]
J. Åkesson[c], K. Link[a], S. Velut[c]
[a]Siemens AG, Energy Sector, Erlangen, Germany
[b]Lund University, Department of Automatic Control, Lund, Sweden.
[c]Modelon AB, Lund, Sweden.

Abstract

This paper shows how different kinds of optimization related task such as offline optimization or optimal control are solved using a combination of Modelica, Optimica, JModelica.org and Python. The application examples presented in this paper are all real industrial applications in the field of Combined Cycle Power Plants. Therefore different workflows have to be combined to solve the underlying task. This paper shows that these workflows can be conveniently connected using Python.

Keywords: Dynamic optimization, Nonlinear Model Predictive Control, Extended Kalman Filter

1 Introduction

Using simulation models to study plant behavior is state of the art today. So more and more attention is paid to applications related to optimization tasks. This includes e.g. offline optimization of plants, optimal plant control or parameter estimation using measurement data. These tasks often need different parts as initialization, simulation and optimization.

This paper shows a methodology which combines the optimization platform JModelica.org [1], the modeling language Modelica, an optimization extension to Modelica (Optimica) and a scripting environment (Python [2][1]) in order to solve the different optimization tasks mentioned above.

Each optimization task is illustrated by an industrial application.

The paper is structured as follows: Section 2 gives some background information about Optimica and Python, while section 3 explains the different industrial applications with focus on scripting. Section 4 summarizes the results of the paper.

2 Background

JModelica.org is an extensible Modelica-based open source platform for optimization, simulation and analysis of complex dynamic systems. A unique feature of JModelica.org is the support for the extension Optimica. Optimica enables users to conveniently formulate optimization problems based on Modelica models using simple but powerful constructs for encoding of optimization interval, cost function and constraints. For user interaction JModelica.org relies on the Python language. Python offers an interactive environment suitable. The following subsections give an introduction to Optimica and show the advantages of the scripting language Python.

2.1 Optimica

The Optimica extension mainly consists of an additional class, *optimization*, which includes the attributes such as *startTime*, *stopTime* and *objective*, which specify the objective function of the optimization problem. Another supplement is the *constraint* section, which can handle different kinds of linear and non-linear equality- and inequality constraints.

A wide range of optimization problems may be solved formulated with Optimica, for example: parameter estimation, tracking, optimal control and so on. Here is a simple example, an optimization problem, based on the double integrator:

$$\min_{u(t)} \int_0^{t_f} 1 \, dt$$

Subject to:

$$\begin{aligned} \dot{x}(t) &= v(t) & x(0) &= 0 \\ \dot{v}(t) &= u(t) & v(0) &= 0 \\ x(t_f) &= 1 & v(t_f) &= 0 \\ v(t) &\leq 0.5 & -1 \leq u(t) &\leq 1 \end{aligned}$$

For this problem, t_f is considered to be free and the objective is to minimize the time it takes to transfer the states from (0, 0) to (0, 1) without violating the constraints. The corresponding problem formulated in Modelica and Optimica looks like this:

```
model DoubleIntegrator
   Real x(start=0);
   Real v(start=0);
   input Real u;
equation
   der(x)=v;
   der(v)=u;
end DoubleIntegrator;

optimization DIMinTime (
        objective= finalTime,
        startTime=0,
        finalTime(free=true,initialGuess=1))

   extends DoubleIntegrator(
        u(free=true, initialGuess = 0.0));
constraint
   x(finalTime)=1;
   v(finalTime)=0;
   v<=0.5;
   u>=-1;
   u<=1;
end DIMinTime;
```

The attribute *free* = true indicates that the variable is an optimization variable and the attribute *initialGuess* provides the numerical solver with an initial guess.

The optimization problem is solved numerically and there is hence an aspect of discretization to consider, but this is considered outside of Optimica. Optimica only represents the mathematical formulation of the problem and several different solver algorithms can be used for solving the different problems at hand. For details on Optimica, see [9].

2.2 Python

Python is open source and a very powerful dynamic programming language that is used in a wide range of application domains. Especially interesting features of Python related to the applications covered in this paper are how well it works with other tools and its scripting possibilities. There are for example interfaces for Gnuplot [10] and Matplotlib [11], which are both suitable for plotting purpose.

Python is compatible with Optimica and it is possible to interact with Modelica and Optimica models through Python scripting (the Python interface of the Jmodelica.org platform is great for this). For example, it enables the possibility to do parameter manipulations and to perform simulation and optimization for a variety of setups. The discretization and solver options (e.g. tolerances) for simulation and optimization can easily be set through the Python script.

The Modelica and Optimica models consists of a set of states, algebraic variables and input variables and they are represented by two model object types; FMU (Functional Mock-up Unit) and JMU (JModelica Model Unit), respectively. FMUs are mainly used for simulation and they follow the FMI-standard (Functional Mock-up Interface), which specifies how the models should be represented and stored [16]. JMUs are mainly used for optimization and follow the JModelica.org standard, similar to the FMI standard for FMUs. For more details see [14]. Both model object types can be imported in to Python. Besides JModelica.org the open source simulation and optimization tool OpenModelica [8] supports Python with its interface called OMPython enabling the user to use the modeling and simulation capabilities of OpenModelica within the Python environment.

The Python packages SciPy [12] and NumPy [13] support linear algebra and matrix operations and are useful when scripting, both for pre- and post-processing, in plotting and in the implementation of algorithms.

Below is a simple example that demonstrates what scripting in Python may look like, using JModelica.org. The script solves the optimization problem described in section 2.1. `DoubleIntegrator.mo` contains the model and `DIMinTime.mop` contains the optimization model. A JMU object is created based on these.

DOI 10.3384/ECP14096777

```
# Importing necessary packages and functions
import numpy as N
from pymodelica import compile_jmu
from pyjmi import JMUModel
import matplotlib.pyplot as plt

# Compiling a JMU- object for optimization
# based on the double integrator
jmu_name= compile_jmu("DIMinTime",
   ["DoubleIntegrator.mo", "DIMinTime.mop"])

# Loading the JMU-object
model_opt = JMUModel(jmu_name)

# Calling the optimiztion function with
# default settings
res = model_opt.optimize();

# Plotting the results
x= res['x']
v =res['v']

# u is the optimal trajectory
u =res['u']
time = res['time']

plt.plot(time, x, ':k',  time, v, '--k',
   time, u,'-k')
plt.grid(True)

# Increasing the plot window to show results
plt.ylim([-1.5,2])
plt.legend(('x','v','u'))
plt.title('Simple example')
plt.show()
```

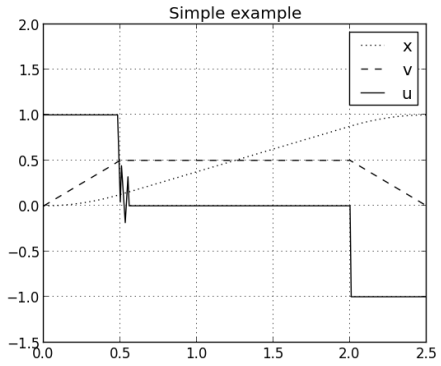

Figure 1: The optimization results

3 Case studies

The JModelica.org platform can be used to perform offline optimization. Python scripting is used for pre- and post-processing.

3.1 Offline optimization: Start-up of a combined cycle power plant

The aim of this offline optimization is to maximize power output of gas and steam turbines without violating given constraints on temperature differences in several components. The optimizer shall find optimized control inputs for gas turbine power and four different control valves (see Figure 2).
This model is an extension of the model presented in [6]: the model scope has been enlarged to also include high pressure (HP) and intermediate pressure (IP) turbine and the corresponding valves.

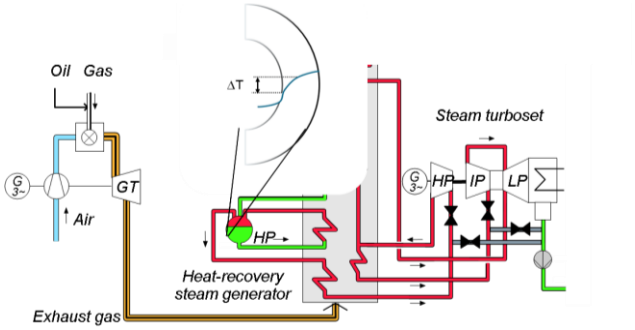

Figure 2: Model for startup optimization

3.1.1. Python Scripting

Figure 3 illustrates the steps written in Python to perform offline optimization.

Since the gas turbine power is fixed until synchronization of the gas turbine, the optimization can not start at t = 0s, but only after synchronization, at t = t_opt_start. Therefore a first simulation determines the initial conditions for the optimization. Additionally to the state initialization at optimization starting point, the optimization algorithm also needs a guess trajectory for all variables. If no explicit guess trajectory is supplied it can be obtained using a second simulation. After the optimized control input has been obtained, several post-processing steps are taken (see Figure 3). In this specific application, the optimization result and the result of the first simulation (until synchronization of the gas turbine) are concatenated to obtain the complete time-dependent behavior of the plant. Also it proved to be useful to run a check, whether all variables are well scaled and whether the constraints are also kept between the collocation points. It has to be stated, that the Python scripting environment offers a very convenient and flexible way to include different pre- and post-optimization tasks.

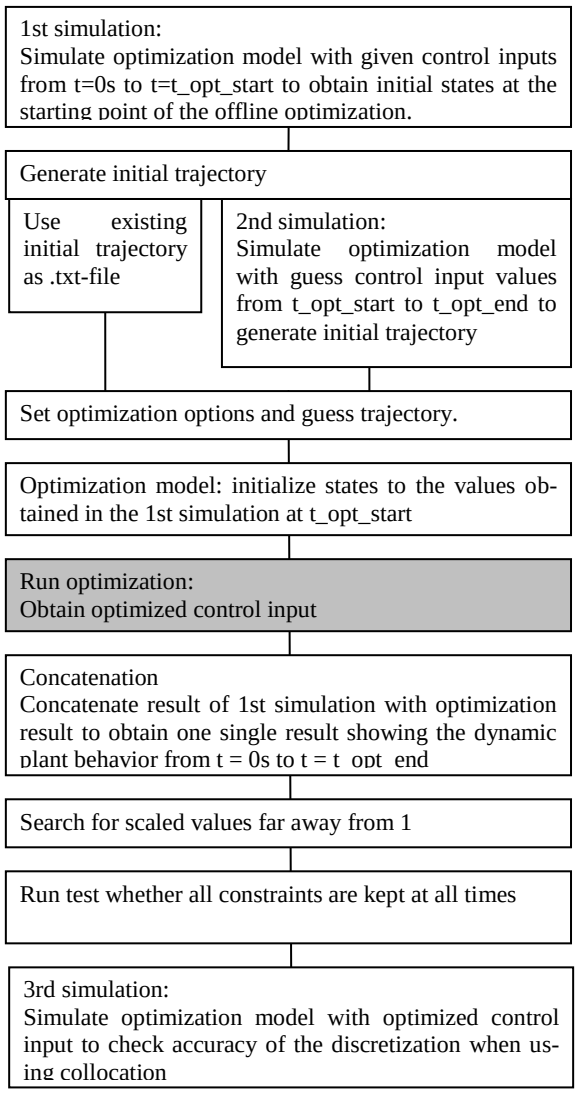

Figure 3: Steps written in Python when performing offline optimization

3.1.2. Results of the offline startup optimization

An offline startup optimization of the system shown in Figure 2 has been performed where the characteristics are given below:

Constraints:

- Model equations
- Maximum pressure of 170 bar
- Pressure dependent maximum wall temperature difference for several components (i.e. HP and IP turbine casing, heat exchanger manifolds etc.)
- Minimum mass flow rate change for a certain mass flow range
- Maximum negative GT power derivative

Control inputs:

- Gas turbine power
- HP and IP turbine bypass valve
- HP and IP turbine control valve

Optimization objective:

- Maximization of total power output (gas turbine power + power of HP and IP turbine)

Some optimization results are presented in the following figures.

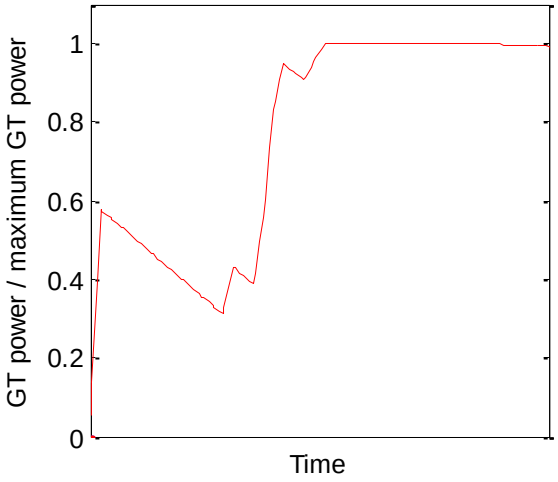

Figure 4: Normalized gas turbine (GT) load (actual divided by maximum gas turbine load).

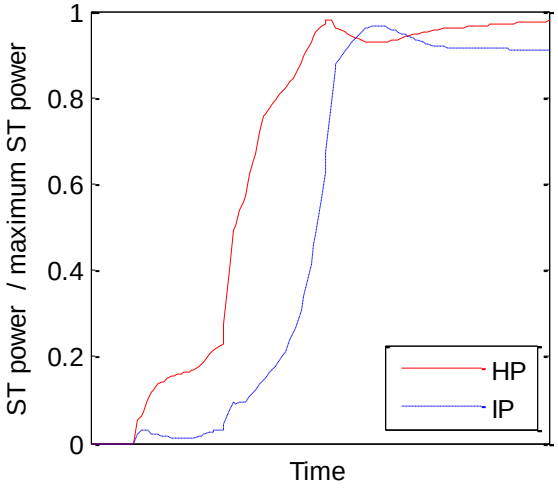

Figure 5: Normalized steam turbine power (solid line: HP turbine, dashed line: IP turbine)

DOI 10.3384/ECP14096777

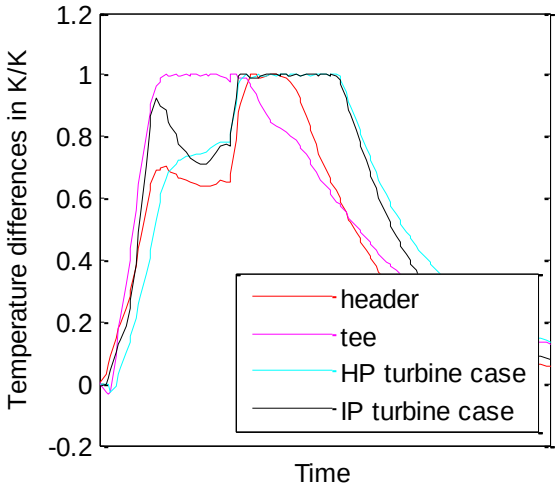

Figure 6: Normalized temperature differences (actual temperature difference divided by maximum allowable temperature difference) in critical components

It can be seen that the optimizer first strongly increases the gas turbine power to maximize the power output (Figure 4). Then the gas turbine power is decreased again to keep the temperature difference in the critical components below their maximum value (Figure 6).

3.2 Nonlinear Model Predictive Control with State Estimation: Extended Kalman Filter

The basic concept of Nonlinear Model Predictive Control (NMPC) is to use a dynamic model to forecast system behavior and optimize the forecast to produce the best decision. In practice an optimal control problem is solved over a finite future horizon, but only the first optimal control signal is applied to the system. Then the optimization horizon is shifted and the calculations are repeated.

The solution of the optimal control problem depends on the initial state of the model which is the current state of the plant. In general, measurements are disturbed by noise or are missing, so that a state estimation algorithm is needed to determine the initial states under consideration of the past record of measurements.

3.2.1 Python Scripting

The python script for optimization with JModelica.org looks as in Figure 7.

This scheme describes the NMPC loop with two dynamic models – one model called optimization model, the other real plant model. The real plant model illustrates the real plant behavior and is more detailed than the optimization model. In future the real plant model will be replaced by measurements of the real plant.

The NMPC loop starts with the generation of an initial trajectory for the optimization. As an alternative the optimization can be initialized with an optimization result too. The initial trajectory is generated by simulating a FMU of the optimization model.

The second step is to solve the optimization problem from t to t+H, where t is the actual time and H the length of the finite optimization horizon (see Figure 8). The optimization horizon is divided into N steps, but only the first control signal is applied to real plant model and the model is simulated to get the new state of the plant. The following step is the state estimation which is explained in more detail below. After the initial state of the model is updated the optimization horizon is moved and the optimization is solved again from t= t+h (h= H/N) to t+2*h. All steps will be repeated until the final time of the optimization t_opt_end is reached.

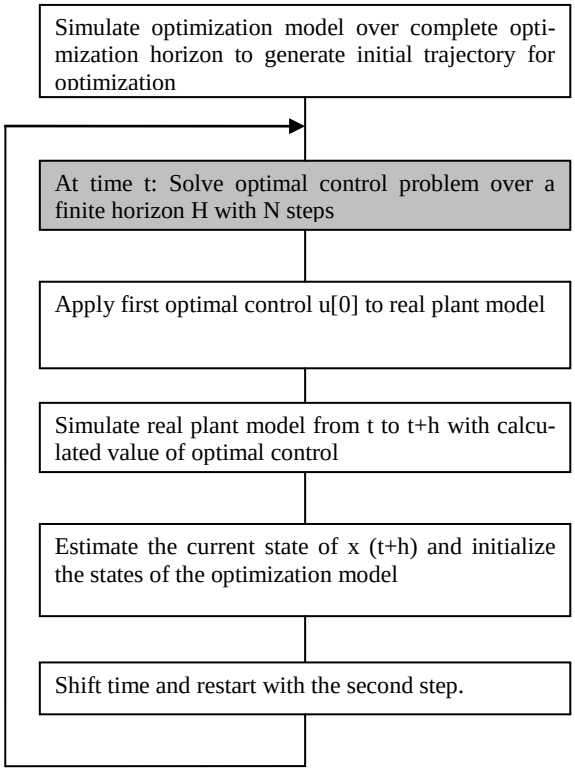

Figure 7: Steps written in Python performing NMPC with state estimation

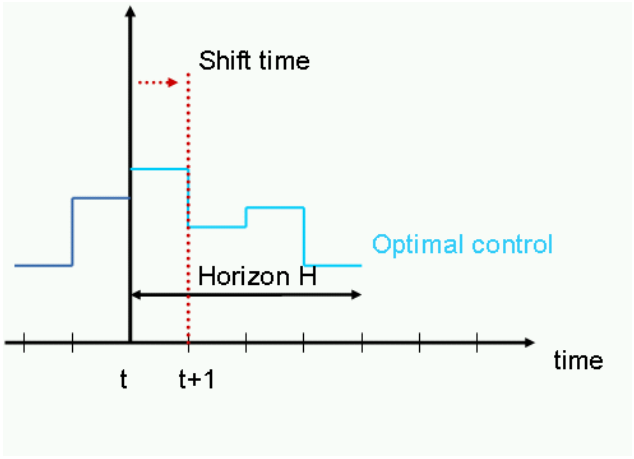

Figure 8: NMPC scheme

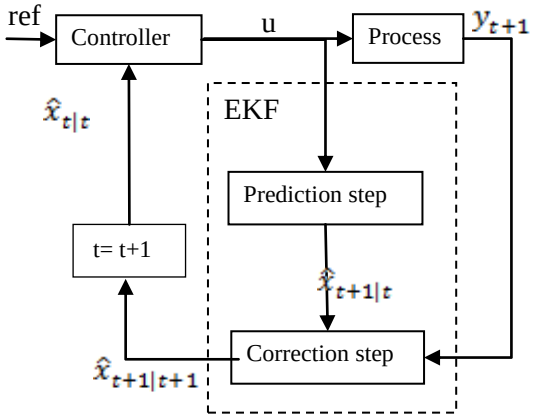

Figure 9: Structure of EKF, where ref corresponds to the control reference, u to the control signal, and x to the state estimation at different stages and y to the real plant measurements.

The strategy of Extended Kalman Filter (EKF) is used for state estimation. The Kalman filter originates from probability theory and it is well established that the Kalman filter is the optimal state estimator for a linear system affected by white noise. The Kalman filter minimizes the estimation error by considering past data of the system. This can be described in a recursive way which is convenient for implementation purposes [15]. Concerning the notation, $\hat{x}_{t+1|t}$ corresponds to the estimation of x at *time = t+1* given the information at *time = t*.

The setup of EKF is according to Figure 9. The estimation consists of two main steps; the prediction and the correction. In the prediction step, the state values at time t+1 are estimated from the system representation ($\hat{x}_{t+1|t}$). The covariance matrix of the estimation error at the prediction step ($P_{t+1|t}$) is also updated in this step.

In the correction step, the Kalman gain (K_{t+1}) is updated. It is then used to derive the corrected state estimation ($\hat{x}_{t+1|t+1}$) by combining the result of prediction step and the latest plant measurements (y_{t+1}). The covariance matrix for the estimation error at the correction step ($P_{t+1|t+1}$) is also updated here. The steps are described in more detail below.

The EKF is an extension of the Kalman filter for nonlinear process models and the approach is basically the same as in the linear case, with an additional linearization to get approximations of $A_{t|t}$ and $C_{t+1|t}$ matrices (using standard notation for linear systems), which are used by the filter. The linearization step was realized with the JModelica.org library pyjmi.linearization.

Prediction:

$\hat{x}_{t+1|t}$ By simulating a FMU of the optimization model.

$A_{t|t}$ By linearizing a JMU of the optimization model (at $\hat{x}_{t|t}$).

$$P_{t+1|t} = A_{t|t}P_{t|t}A_{t|t}^T + Q_t$$

Correction:

$C_{t+1|t}$ By linearizing a JMU of the optimization model (at $\hat{x}_{t+1|t}$)

$$K_{t+1} = P_{t+1|t}C_{t+1|t}^T(C_{t+1|t}P_{t+1|t}C_{t+1|t}^T + R_t)^{-1}$$

y_{t+1} By simulating a FMU of the real plant model.

$$\hat{x}_{t+1|t+1} = \hat{x}_{t+1|t} + K_{t+1}(y_{t+1} + \hat{x}_{t+1|t})$$

$$P_{t+1|t+1} = P_{t+1|t} + K_{t+1}C_{t+1|t}P_{t+1|t}$$

Q and R represent the covariance matrices for process and measurement noise.

For this example, we assumed that both noises are uncorrelated and conform to a normal distribution. Q and R are diagonal matrices. The entries on the diagonal of the covariance matrices Q and R are the vari-

DOI 10.3384/ECP14096777

ances of each process variable and measurement and are set to 1 in most cases. Q and R where approximated to be uncorrelated in time and the diagonal elements where set to: one over the square root of the standard deviation of the corresponding state/measurement, in most cases to 1.

The EKF does not consider constraints, and this has to be compensated for in an additional step (the feasibility correction). This is important to note since the optimization strategy is interior point optimization, and no solution will be found if the starting point is outside of the feasible region.

The strategy to considering this fact was simply to use the prediction as state estimation, without the correction step, since the prediction always will be feasible.

3.2.2 Results of NMPC with State Estimation: Extended Kalman Filter

In reality, there will be large differences between the controller model and the real plant, this is natural. However, for the evaluation of the implementation, the models were kept basically the same, with the same state representation. The optimization model was augmented in order to compensate for the differences between the plant models. The differences were approximated as constant disturbances, by introducing the additional states *d*, here on referred to as *the disturbance states*. These are unmeasurable states of the real plant and their values can be estimated by the EKF.
See a simple example of the augmentation below, where *x* represent the original states, *f(x, u)* represents the process with input *u* and *d* represent the disturbance states.

$$\begin{bmatrix} x_{k+1} \\ d_{k+1} \end{bmatrix} = \begin{bmatrix} f(x_k, u_k) \\ d_k \end{bmatrix}$$

The NMPC loop combined with state estimation was evaluated using the example of an enthalpy controller of the heat recovery steam generator (HRSG) (consisting of economizer, evaporator and superheater, see Figure 10) of a combined cycle power plant.

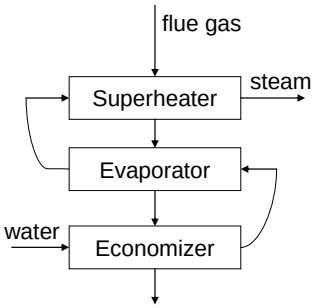

Figure 10: Rough sketch of the system to be controlled.

Two disturbance states were added to the optimization model (one to the flue gas mass flow rate and one to the water pressure) in order to consider differences. Additionally a parameter related to the heat transfer was modified to represent a typical modeling error in addition to initial offsets for each state.

There were three objectives for the controller and they were considered in the formulation of the optimization problem:

1. Keep the steam temperature at the superheater outlet at desired set point.
2. Guarantee subcooling at the evaporator inlet by keeping the temperature below a specified maximum value.
3. Have an adequate degree of superheating at the outlet of the evaporator section.

Figure 11 and Figure 12 display the results for this setup.

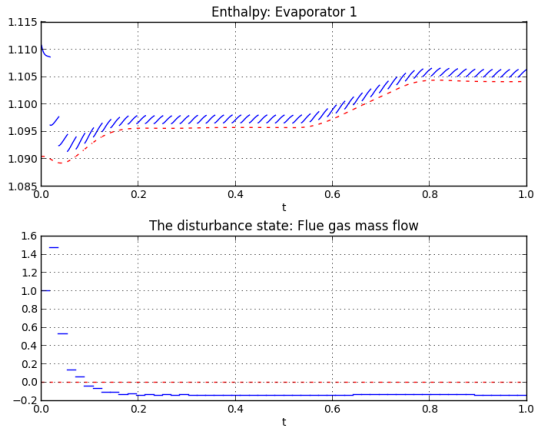

Figure 11: The progress of the state estimation, scaled according to nominal value and time. The red line represents the real plant behavior. The *top plot* represents one of the process states, the enthalpy at the inlet of first evaporator. The *bottom plot* represents the introduced disturbance state, with an initial error

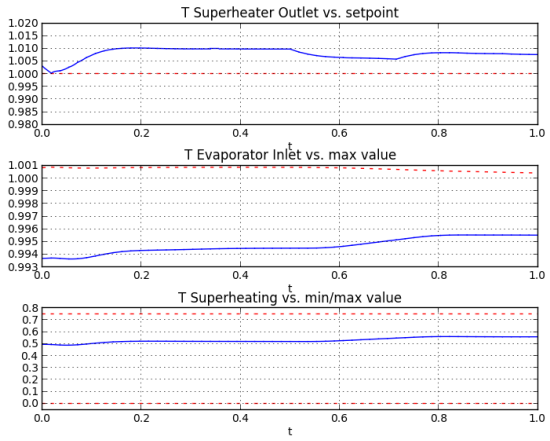

Figure 12: The control objective, scaled according to nominal value and time. The *top plot* displays the temperature set point control; the red line represents the set point. The *middle plot* displays the sub cooling control; the red line represents the maximum value for the temperature at the economizer outlet (pressure dependent). The *bottom plot* displays the superheating control, the red lines represents the minimum/maximum degree of superheating.

The jumping in Figure 11 is related to the correction step of the EKF. Also worth noting is that the disturbance state does not estimate the constant disturbance to 0, and this is probably due to the introduced modeling error. The disturbance state tries to compensate for this as well.

The controller performance is quite satisfying, but there are some offsets that should be considered in the future work with optimization problem formulation. The reason for the offset can for example be related to the fact that the plant did not reach steady state within the time frame for the experiment. It is not clear from Figure 11, but it is however the case. Evaluating the performance for a longer time might get rid of the offset, but this was not possible to realize at the time of the application evaluation because of some memory leaks. Modifications to the objective function could be increasing the weight on the elements in the objective function related to set point deviations.

3.3 Parameter estimation

Parameters are typically estimated by some form of least squares. This method minimizes the sum of the squared discrepancies between measurement and expected value. M is the number of measurements and N the number of discrete time steps.

$$\min \sum_{i=1}^{N} \sum_{j=1}^{M} \left(\left(x_{i,j} - x^{meas}{}_{i,j}(t[i]) \right)^2 \right)$$

The default algorithm for solving optimal control problems and parameter estimation problems in JModelica.org is the collocation algorithm. For our application we used the Nelder-Mead method, a heuristic search method using the concept of a (N+1)-dimensional simplex, where N is the number of estimated parameters. These kinds of derivative free algorithms are implemented in JModelica.org ([2]).

The Nelder-Mead algorithm was the preferred choice, although this method is quite slow, but has the best convergence behavior especially for many measurements and a lot of parameters.

3.3.1 Python Scripting

As the Nelder-Mead method is already implemented in JModelica.org as described in [9], the python script is quite simple. In a first step all measurements are imported from .mat file. Then the Nelder-Mead function nelme is called which solves the optimization problem and uses the defined objective function as input.

3.3.2 Results of Parameter estimation

A parameter estimation of the Modelica model for optimizing the start up process of a combined cycle power plant in 3.1 has been implemented.
The real plant measurements were given for a period of 1h. The data were loaded in python as .mat file. The gas turbine power, the gas turbine mass flow, the injection mass flow and the back pressure measurement were set directly as boundaries of the model, the other measurements (wall temperatures, fluid temperatures and pressure) are used to minimize the error between the measurements and their simulated values.

The algorithm was tested for two measurement sets to verify the result, one for a hot start and one for a warm start of the power plant. As expected the estimated parameters have smaller differences, but have equal dimensions.

Figure 13 shows the simulation result with original and optimized parameters compared to measurements. As can be seen, the simulation result with the optimized parameter fits better to the measurements. Nevertheless there are still differences between the model and the real plant behavior. Uncertainty of measurements, not modeled effects and components are some reasons for the deviations.

DOI 10.3384/ECP14096777

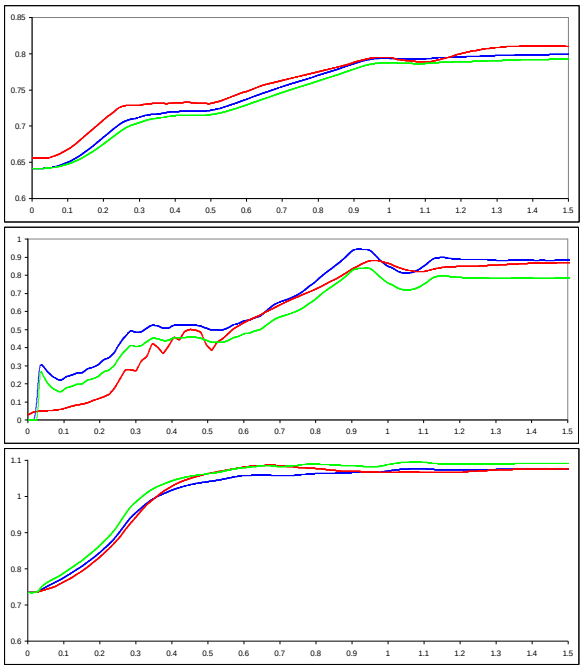

Figure 13: Simulation result with original parameters (green) optimized parameters (blue) and measurements (red). The top plot displays the wall temperature of the separator; the middle plot displays the reheat mass flow and the bottom plot the outlet temperature of the first superheater.

4 Summary

This paper shows how different industrial optimization applications can be solved combining Modelica and Optimica with the scripting language Python.

Three different optimization tasks have been considered to improve the dynamic processes in a combined cycle power plant: offline optimization of the start-up process, online enthalpy control of the HRSG and parameter estimation for the start-up optimization.

All optimization tasks have been formulated with Optimica based on Modelica models. For pre- and post processing issues and interaction of JModelica.org and the Modelica models, the scripting tool Python was used.

As shown in section 3 'Case studies', for complex optimization techniques like NMPC with combinations of several optimization and simulation steps, Python is ideally suited. For industrial applications of power plant sector the definition of such controllers inside the Modelica model is not desired and unnecessary since other interfaces (e.g. connection to database) have to be realized additionally.

The work presented in this paper is one step towards a complete online-optimization tool chain for NMPC.

5 Acknowledgements

The German Ministry BMBF has partially funded this work (BMBF funding code: 01IS12022A) within the ITEA2 project MODRIO [7]. Modelon's contribution to this work was partially funded by Vinnova, through the ITEA 2 project MODRIO.

References

[1] JModelica.org, http://jmodelica.org/, viewed 2013-12-05.

[2] Python Software Foundation. Python Programming Language - Official Website, http://www.python.org/, 2012, viewed 2013-12-05

[3] A. Johnsson, Nonlinear Model Predictive Control for Combined Cycle Power Plants, Master's Thesis, Lund University, Department of Automatic Control, 2013.

[4] C. Andersson, S. Gedda, J. Akesson, S. Diehl, Derivative-free Parameter Optimization of Functional Mock-up Units, 9th International Modelica Conference, Munich, Germany, 2012.

[5] Lie, B., Haugen Finn, Scripting Modelica Using Python, Telemark University College, Porsgrunn, Norway, 2012

[6] A. Lind, E. Sällberg, S. Velut, S. Gallardo Yances, J. Åkesson, K. Link: Start-up Optimization of a Combined Cycle Power Plant, Proceedings of the 9th International MODELICA Conference, September 3-5, 2012, Munich, Germany

[7] https://www.modrio.org/

[8] https://openmodelica.org/

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

[10] GnuPlot, http://www.gnuplot.info/, viewed 2013-12-05.

[11] MatPlotLib, http://matplotlib.org/, viewed 2013-12-05.

[12] SciPy, http://www.scipy.org, viewed 2013-12-05.

[13] Numpy, http://www.numpy.org/, viewed 2013-12-05.

[14] The JModelica.org User Guide, http://www.jmodelica.org/api-docs/usersguide/JModelicaUsersGuide-1.11.0.pdf, viewed 2013-12-05.

[15] E. Haselting, J. Rawlings, A Critical Evaluation of Extended Kalman Filtering and Moving Horizon Estimation, http://jbrwww.che.wisc.edu/tech-reports/twmcc-2002-03.pdf, 2003, viewed: 2013-05-07.

[16] The FMI-standard, https://www.fmi-standard.org/, viewed 2014-01-21.

DOI 10.3384/ECP14096777