ModestPy

An Open-Source Python Tool for Parameter Estimation in Functional Mock-up Units

Krzysztof Arendt, M. Jradi, M. Wetter, C.T. Veje
The American Modelica Conference 2018, October 9-10, 2018

Center for Energy Informatics, University of Southern Denmark
Introduction
Background

- Functional Mock-up Interface (FMI) is becoming a de facto standard co-simulation interface, already supported by over 100 simulation tools
- **FMI offers flexibility** in terms of modeling environments
- **FMI attracts generic tools** for co-simulation, system identification, and optimization
- There are several tools for **parameter estimation** in Functional Mock-up Units (FMUs), but most of them are tied to at least one of the following:
  - Specific optimization algorithms
  - Specific proprietary platforms
  - Large software environments
Objective

The objective was to develop a tool for parameter estimation in FMUs that would:

- be lightweight,
- support multiple optimization methods,
- support chaining of global and local methods,
- be easily deployable.
Software Description
Architecture

Figure 1: Package structure.

Available algorithms:

- Genetic Algorithm (GA)
- Generalized Pattern Search (GPS)
- SciPy:
  - Sequential Least Squares Programming (SLSQP)
  - Limited Memory Broyden-Fletcher-Goldfarb-Shanno with box constraints (L-BFGS-B)
  - Truncated Newton Method (TNC)

Dependencies:

pyfmi, numpy, scipy, pandas, matplotlib
Currently, two type of error metrics are implemented, the total mean-square error ($MSE_{tot}$) and the total normalized mean-square error ($NMSE_{tot}$). $NMSE_{tot}$ is suggested for multi-output models.

\[
MSE_{tot} = \sum_i \frac{\sum_{t=1}^{N} (\hat{Y}_i^t - Y_i^t)^2}{N}
\]

\[
NMSE_{tot} = \sum_i \frac{MSE_i}{\bar{Y}_i^2}
\]

where $\hat{Y}_i^t$ is the measured value of variable $i$ at time step $t$, $Y_i^t$ is the simulated value of variable $i$ at time step $t$, $\bar{Y}_i$ is the mean measured value of variable $i$, $N$ is the number of time steps, and $MSE_i$ is the mean-square error for variable $i$. 
Installation

Through conda (recommended):

conda config --add channels conda-forge
conda install modestpy

Through pip:

python -m pip install modestpy

Installation through pip requires pyfmi to be installed separately.
from modestpy import Estimation

session = Estimation(workdir, fmu_path, 
                    inputs, known_parameters, 
                    estimated_parameters, measurements, 
                    method=('GA', 'GPS'), 
                    ga_opts={'maxiter': 5, 'tol': 1e-4}, 
                    gps_opts={'maxiter': 500, 'tol': 1e-6}, 
                    ftype='MSE')

estimates = session.estimate()
err, res = session.validate()
Example
Gray-box model is calibrated to mimic the dynamics of a white-box model implemented in *EnergyPlus*.

Model outputs used in the cost function: $T$, $CO_2$, $\text{verate}$, $q_{\text{rad}}$.

Error metric: $NMSE_{\text{tot}}$.

---

$T$ – indoor temperature [$^\circ\text{C}$], $CO_2$ – indoor $CO_2$ [ppm], $\text{verate}$ – ventilation airflow rate [$m^3s^{-1}$], $q_{\text{rad}}$ – radiator heating rate [W].
Figure 2: Gray-box zone model developed in Modelica (using Dymola).
## Estimation Setup

### Table 1: Setup of model parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Initial guess*</th>
<th>Lower bound</th>
<th>Upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>shgc</td>
<td>5</td>
<td>0.1</td>
<td>10</td>
</tr>
<tr>
<td>tmass</td>
<td>50</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>imass</td>
<td>50</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>RExt</td>
<td>5</td>
<td>0.1</td>
<td>10</td>
</tr>
<tr>
<td>RInt</td>
<td>5</td>
<td>0.1</td>
<td>10</td>
</tr>
<tr>
<td>Vinf</td>
<td>5</td>
<td>0.1</td>
<td>10</td>
</tr>
<tr>
<td>maxVent</td>
<td>5</td>
<td>0.1</td>
<td>10</td>
</tr>
</tbody>
</table>

* Not used by GA

shgc – solar heat gain coefficient [\(\cdot\)], tmass – indoor air thermal mass [\(\text{JK}^{-1}\text{m}^{-3}\)], imass – internal thermal mass [\(\text{JK}^{-1}\text{m}^{-2}\)], RExt – external wall resistance [\(\text{m}^2\text{WK}^{-1}\)], RInt – internal wall resistance [\(\text{m}^2\text{WK}^{-1}\)], Vinf – infiltration air change rate [\(\text{h}^{-1}\)], maxVent – max. ventilation air change rate [\(\text{h}^{-1}\)]
### Results

**Table 2:** CPU time and $NMSE_{tot}$ for validation and training, sorted in ascending order by validation error

<table>
<thead>
<tr>
<th>Method</th>
<th>Training $NMSE_{tot}$</th>
<th>Validation $NMSE_{tot}$</th>
<th>CPU Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA+SLSQP</td>
<td>0.377</td>
<td>0.353</td>
<td>920</td>
</tr>
<tr>
<td>GA+GPS</td>
<td>0.351</td>
<td>0.371</td>
<td>1319</td>
</tr>
<tr>
<td>GA+TNC</td>
<td>0.393</td>
<td>0.372</td>
<td>801</td>
</tr>
<tr>
<td>GA</td>
<td>0.394</td>
<td>0.373</td>
<td>723</td>
</tr>
<tr>
<td>GA+L-BFGS-B</td>
<td>0.349</td>
<td>0.379</td>
<td>934</td>
</tr>
<tr>
<td>GPS</td>
<td>1.306</td>
<td>3.428</td>
<td>986</td>
</tr>
<tr>
<td>TNC</td>
<td>4.967</td>
<td>5.856</td>
<td>101</td>
</tr>
<tr>
<td>L-BFGS-B</td>
<td>4.929</td>
<td>6.808</td>
<td>38</td>
</tr>
<tr>
<td>SLSQP</td>
<td>5.040</td>
<td>6.920</td>
<td>12</td>
</tr>
</tbody>
</table>
Figure 3: Histogram of estimates yielded by the 9 method sequences.
**Figure 4:** Cost function evaluated on the training data based on linear combinations of parameters yielded by GA ($x_1$) and SLSQP ($x_2$). Sections with positive derivatives with respect to $s$ marked in red.
Figure 5: Parameter evolution in the genetic algorithm – color represents the training error (darker more accurate).
Figure 6: Validation root-mean-square error (RMSE) per output variable.
Figure 7: Validation results: temperature (T), CO₂ (CO₂), ventilation airflow rate (verate).
Figure 8: Validation results: radiator heating rate ($q_{\text{rad}}$).
Conclusions and Future Work
Conclusions

• The in-house algorithms (GA, GPS) were validated.
• Using GA for a preliminary global search significantly improved the model accuracy in the test case.\textsuperscript{1}
• The current functionality of the tool is already sufficient for a general use. It is used by the authors for calibrating gray-box models of buildings and HVAC systems for the use in MPC.

\textsuperscript{1}It should be noted, that the initial global search would not be needed if the approximate initial values of parameters were known. In such a case the gradient-based methods would easily outperform GA. Another solution could be to run gradient-based methods with multiple initial guesses.
The development work continues and there are plans to include the following functionality:

- a simple graphical user interface to attract users less experienced in the Python programming language,
- support for on-line estimation methods (e.g. Kalman filter),
- support for multi-period stochastic gradient descent training,
- support for parallel processing methods.
Project repository:
https://github.com/sdu-cfei/modest-py