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

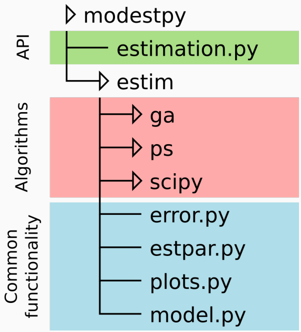

Figure 1: Package structure.

Dependencies:

`pyfmi`, `numpy`, `scipy`, `pandas`, `matplotlib`

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)

Error Metrics

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.

Usage

```
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

Test Case: Thermal Zone Gray-Box Model

- 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`, `CO2`, `verate`, `qrad`
- Error metric: $NMSE_{tot}$

`T` – indoor temperature [$^{\circ}$C], `CO2` – indoor CO_2 [ppm], `verate` – ventilation airflow rate [$m^3 s^{-1}$], `qrad` – radiator heating rate [W]

Gray-Box Model

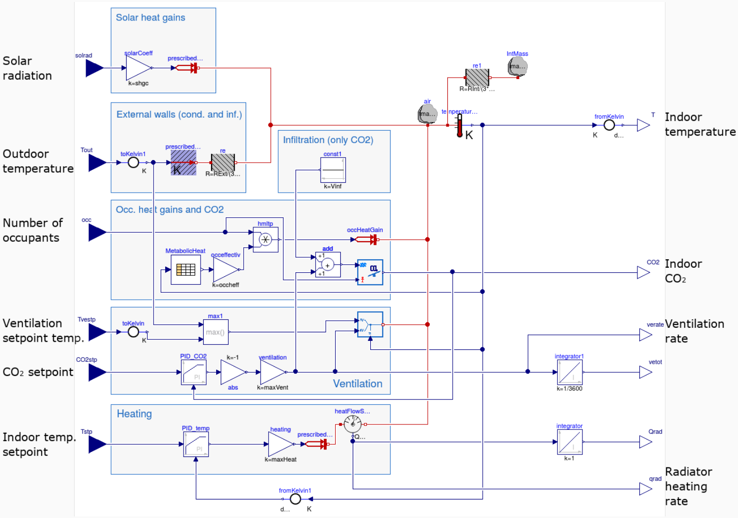

Figure 2: Gray-box zone model developed in Modelica (using Dymola).

Estimation Setup

Table 1: Setup of model parameters

Parameter	Initial guess*	Lower bound	Upper bound
`shgc`	5	0.1	10
`tmass`	50	1	100
`imass`	50	1	100
`RExt`	5	0.1	10
`RInt`	5	0.1	10
`Vinf`	5	0.1	10
`maxVent`	5	0.1	10

* Not used by GA

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

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

Method	Training $NMSE_{tot}$	Validation $NMSE_{tot}$	CPU Time [s]
GA+SLSQP	0.377	0.353	920
GA+GPS	0.351	0.371	1319
GA+TNC	0.393	0.372	801
GA	0.394	0.373	723
GA+L-BFGS-B	0.349	0.379	934
GPS	1.306	3.428	986
TNC	4.967	5.856	101
L-BFGS-B	4.929	6.808	38
SLSQP	5.040	6.920	12

Results

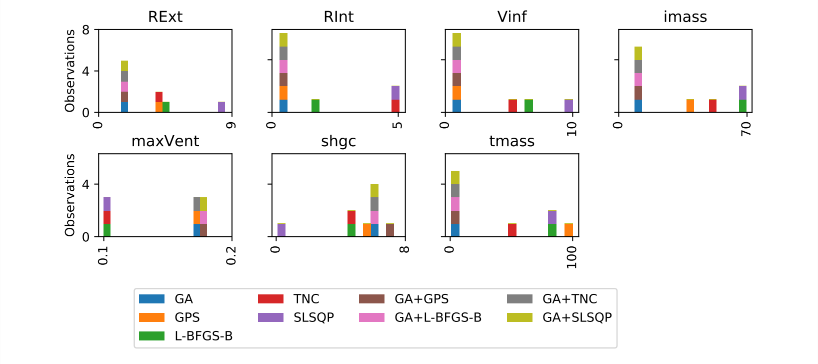

Figure 3: Histogram of estimates yielded by the 9 method sequences.

Results

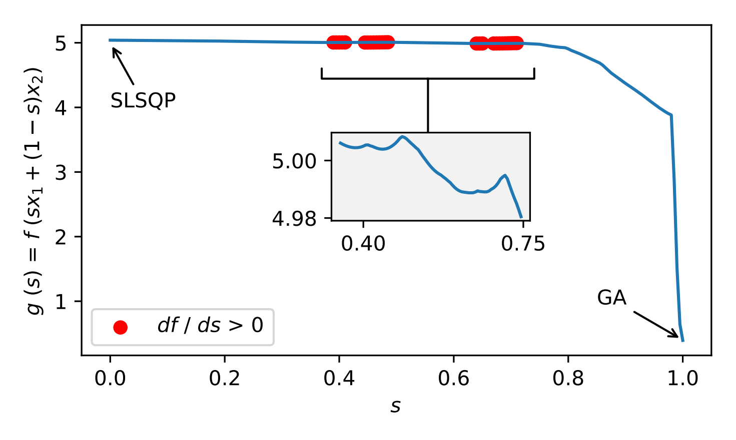

Figure 4: Cost function evaluated on the training data based on linear combinations of parameters yielded by GA ($\mathbf{x}_1$) and SLSQP ($\mathbf{x}_2$). Sections with positive derivatives with respect to s marked in red.

Results

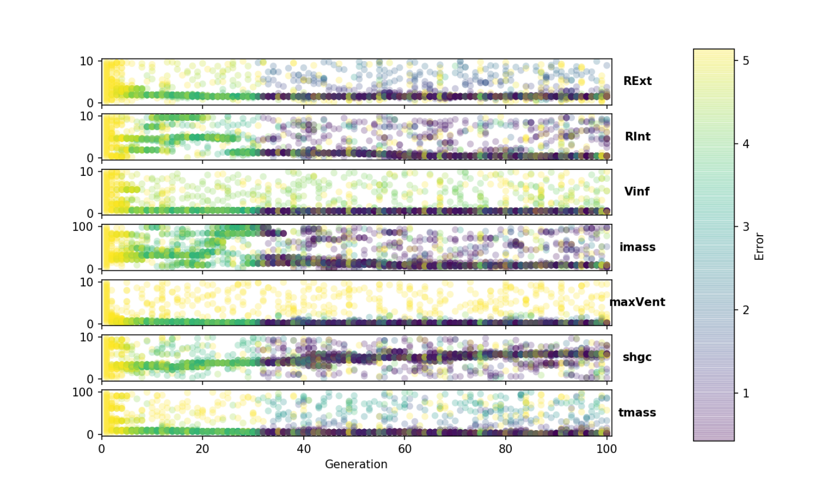

Figure 5: Parameter evolution in the genetic algorithm – color represents the training error (darker more accurate).

Results

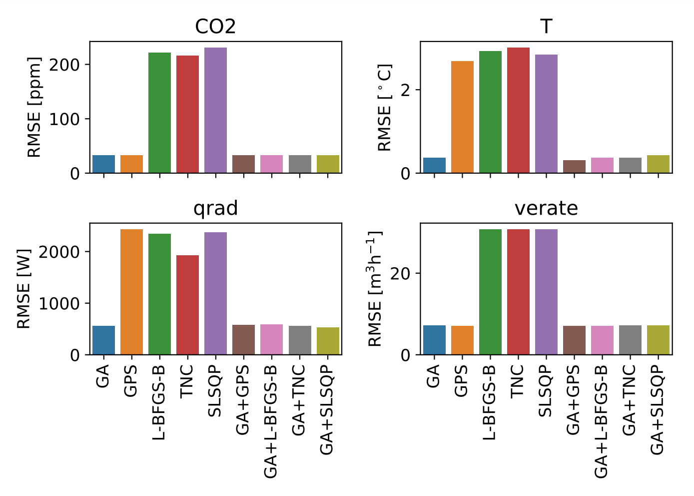

Figure 6: Validation root-mean-square error (RMSE) per output variable.

Results

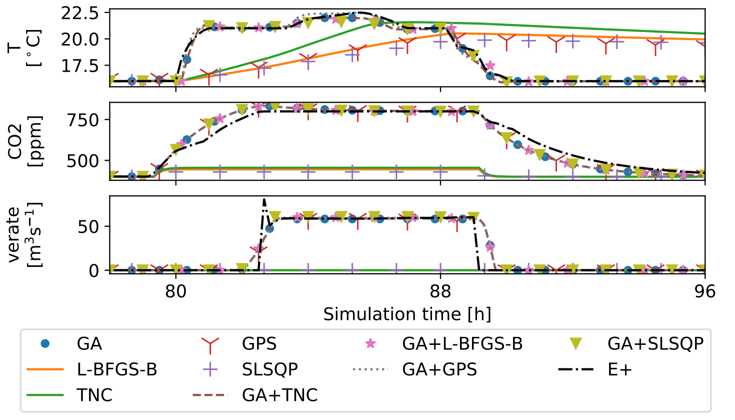

Figure 7: Validation results: temperature (`T`), CO_2 (`CO2`), ventilation airflow rate (`verate`).

Results

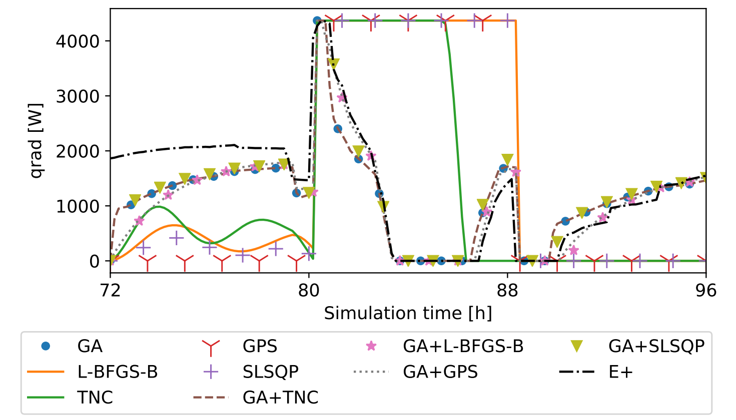

Figure 8: Validation results: radiator heating rate (`qrad`).

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.[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.

[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.

Future Work

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.

Repository

Project repository:

https://github.com/sdu-cfei/modest-py