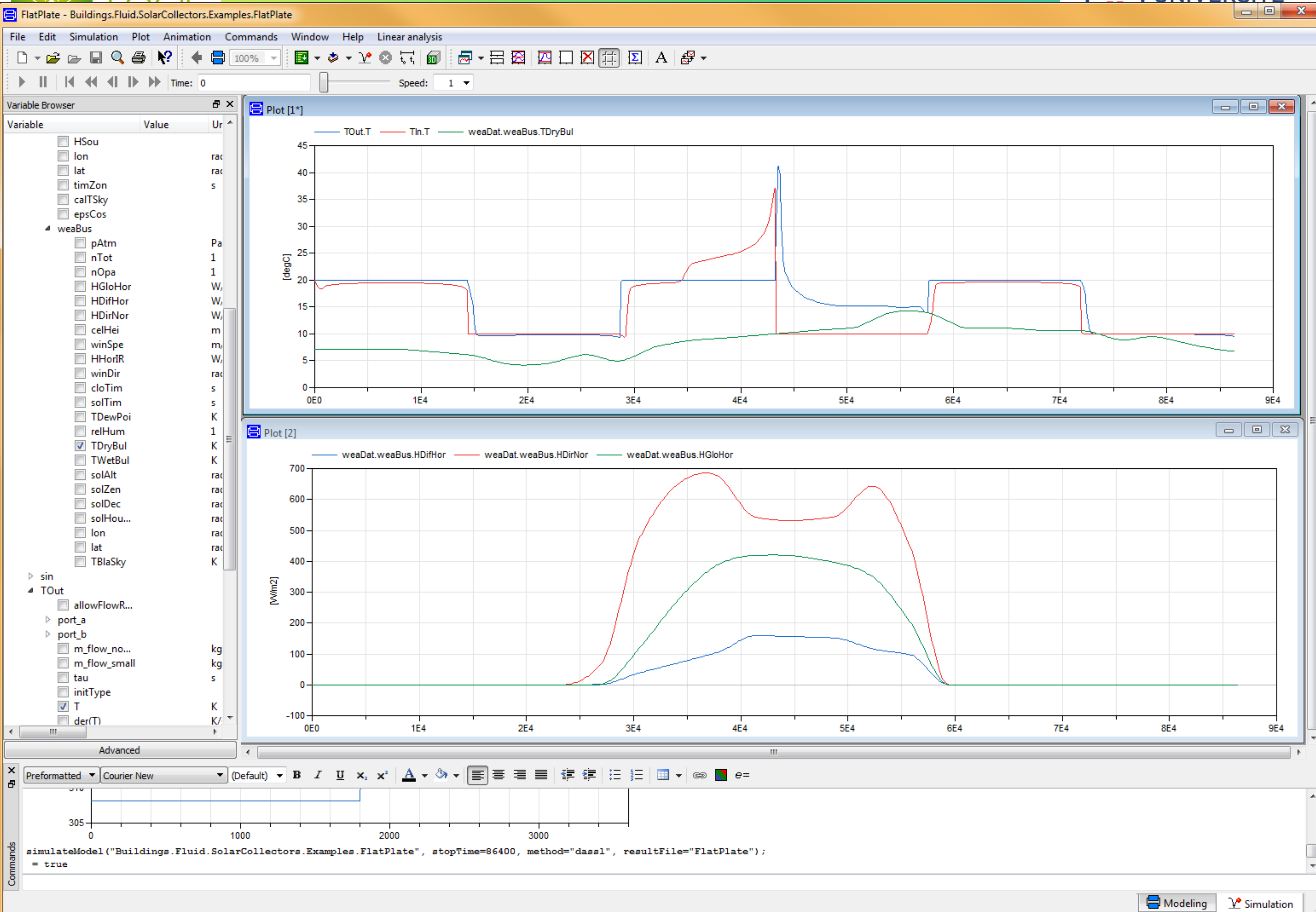
GENSIM scientific school
24-28 oct. 2016 – Porticcio, Corsica

Workflow automation tools

Simon Rouchier, Ph.D.
Université Savoie Mont-Blanc
simon.rouchier@univ-smb.fr


FlatPlate - Buildings.Fluid.SolarCollectors.Examples.FlatPlate
File
Edit
Simulation
Plot
Animation
Commands
Window
Help
Linear analysis
Time: 0
Speed: 1
Variable Browser
Variable
Value
HSou
lon
lat
timZon
calTSky
epsCos
weaBus
pAtm
nTot
nOpa
HGloHor
HDifHor
HDirNor
celHei
winSpe
HHorIR
winDir
cloTim
solTim
TDewPoi
relHum
TDryBul
TWetBul
solAlt
solZen
solDec
solHou...
lon
lat
TBlaSky
sin
TOut
allowFlowR...
port_a
port_b
m_flow_no...
m_flow_small
tau
initType
T
der(T)
Advanced
Plot [1*]
TOut.T
TIn.T
weaDat.weaBus.TDryBul
[degC]
Plot [2]
weaDat.weaBus.HDifHor
weaDat.weaBus.HDirNor
weaDat.weaBus.HGloHor
[W/m2]
Preformatted
Courier New
(Default)
simulateModel("Buildings.Fluid.SolarCollectors.Examples.FlatPlate", stopTime=86400, method="dassl", resultFile="FlatPlate");
= true
Commands
Modeling
Simulation

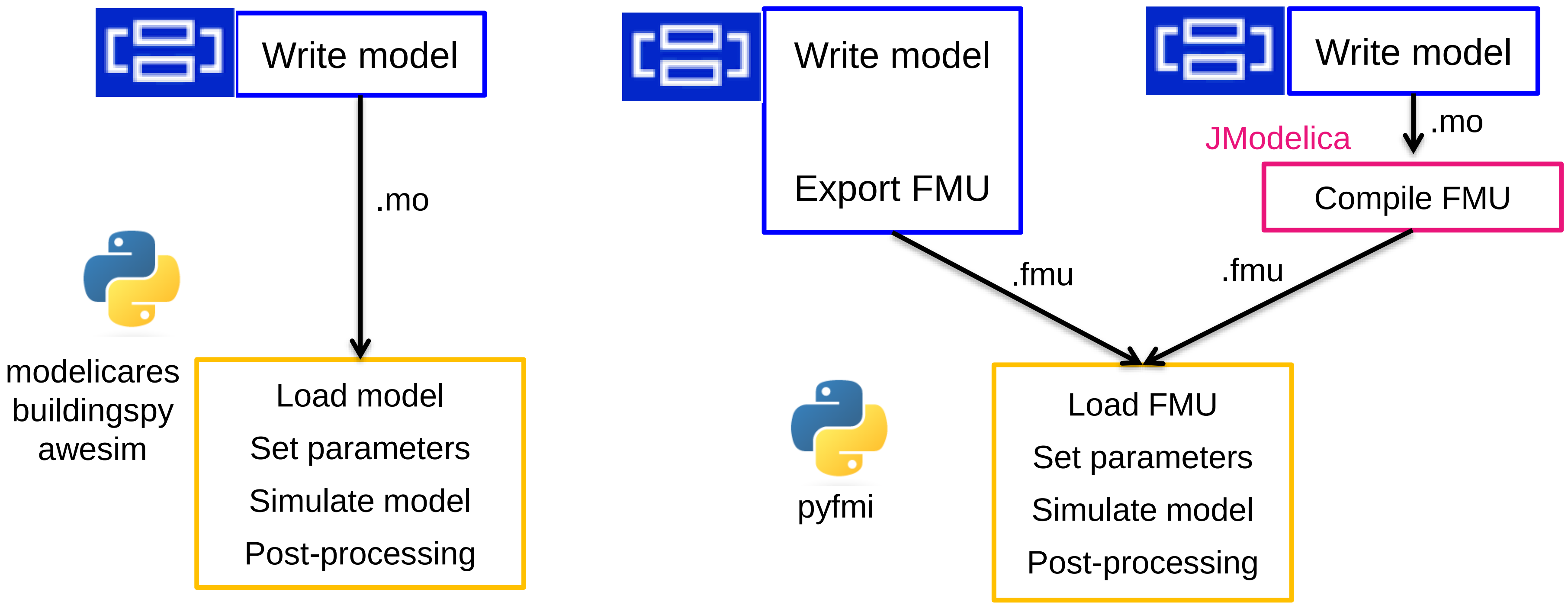
Why do we need workflow automation tools ?

- better post-processing capabilities
- setting up series of simulations (parametric studies, optimisation, system identification...)
- parallel computing

Workflow

Option 1: direct coupling between Modelica and Python

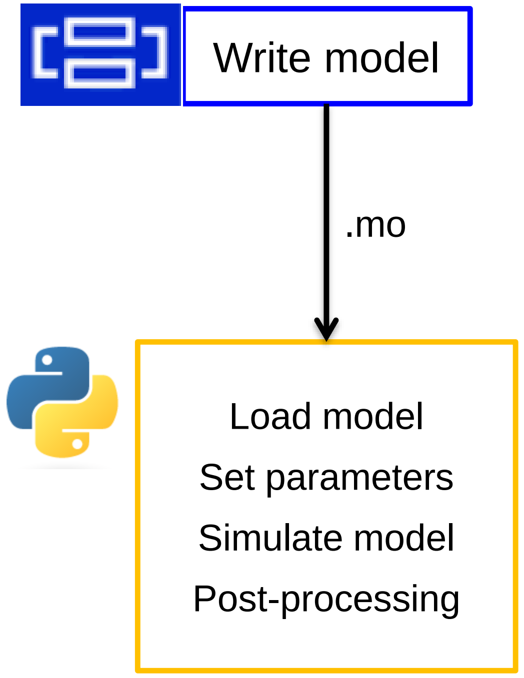
Option 2: using FMUs



Software prerequisites

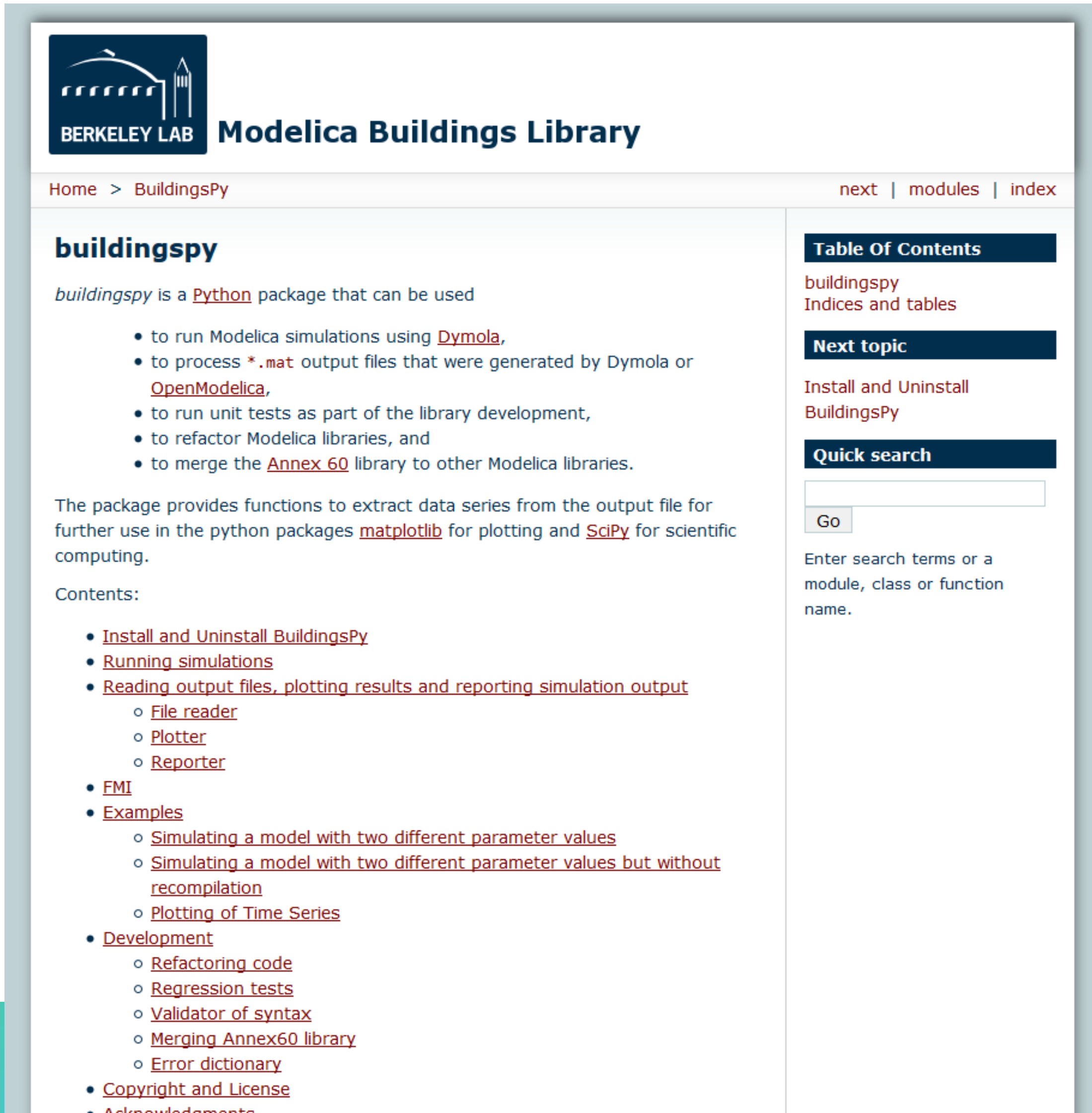
- Modelica environment (preferably Dymola)
- Python environment https://www.continuum.io/downloads
- Python packages
  - BuildingsPy http://simulationresearch.lbl.gov/modelica/buildingspy/
  - PyFMI https://pypi.python.org/pypi/PyFMI

BuildingsPy



BuildingsPy

http://simulationresearch.lbl.gov/modelica/buildingspy/



BERKELEY LAB **Modelica Buildings Library**

Home > BuildingsPy   next | modules | index

buildingspy

buildingspy is a Python package that can be used

- to run Modelica simulations using Dymola,
- to process `*.mat` output files that were generated by Dymola or OpenModelica,
- to run unit tests as part of the library development,
- to refactor Modelica libraries, and
- to merge the Annex 60 library to other Modelica libraries.

The package provides functions to extract data series from the output file for further use in the python packages matplotlib for plotting and SciPy for scientific computing.

Contents:

Table Of Contents

Next topic

Install and Uninstall BuildingsPy

Quick search

Go

Enter search terms or a module, class or function name.

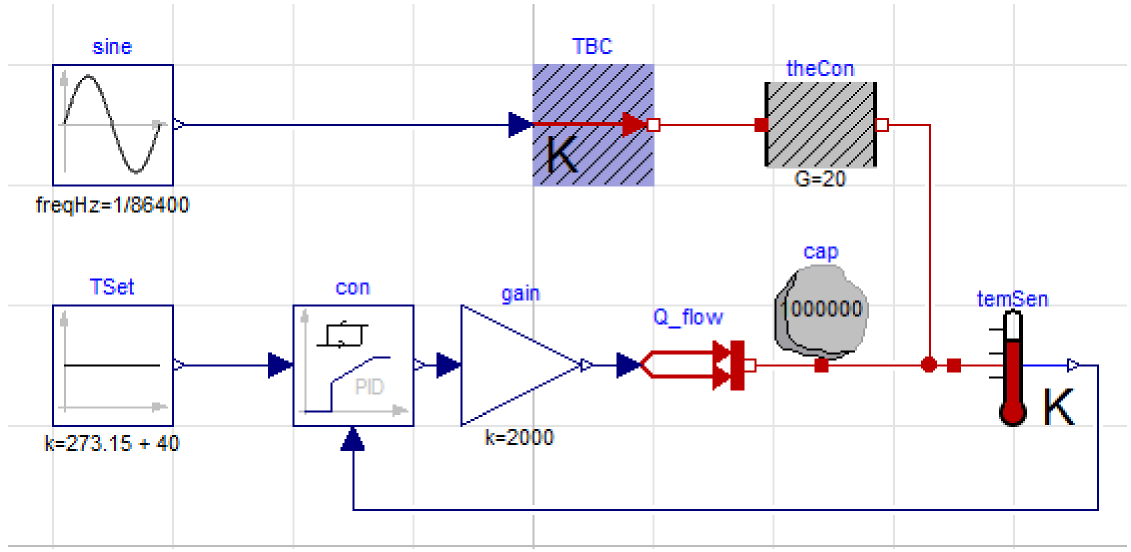
class Simulator

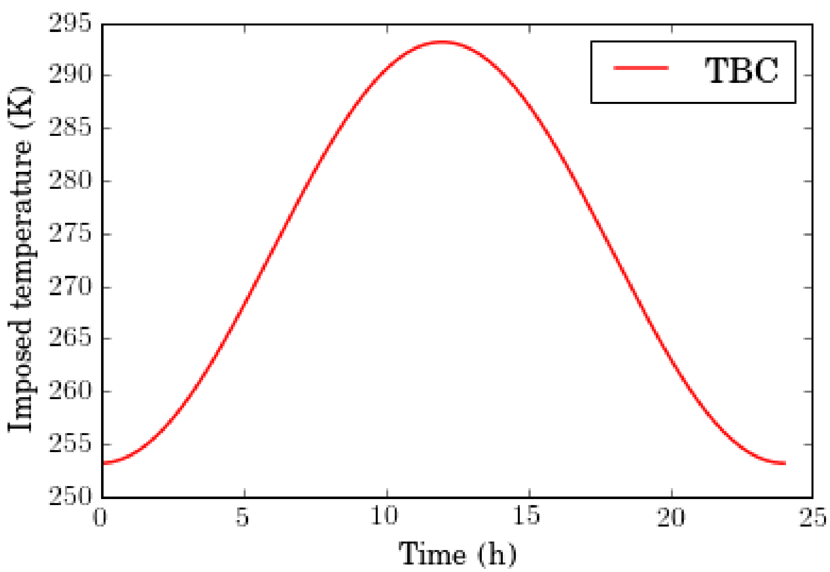
Used to run Modelica simulations, add model modifiers, parameter declarations, set solver type, results directory, stop time...

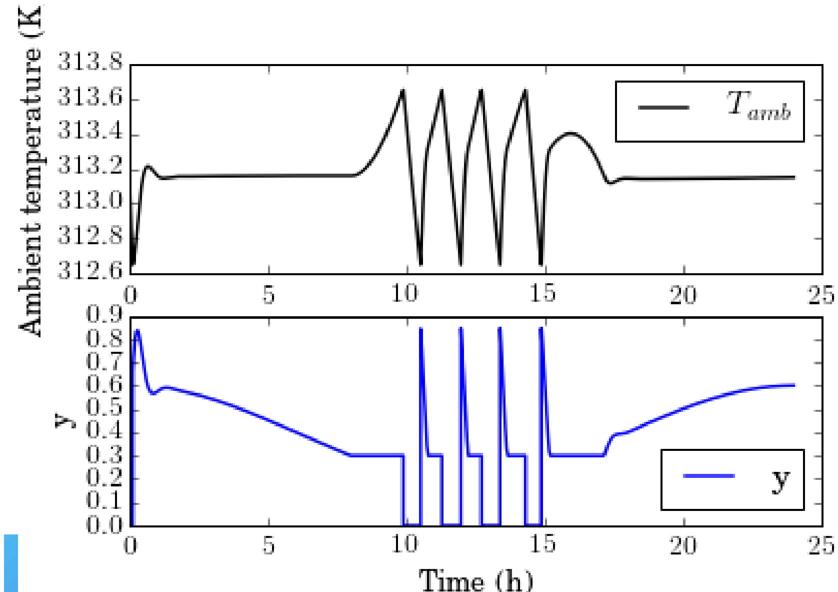
class Reader

reads *.mat files that were generated by Dymola or OpenModelica

Buildings.Controls.Continuous.Examples.PIDHysteresis





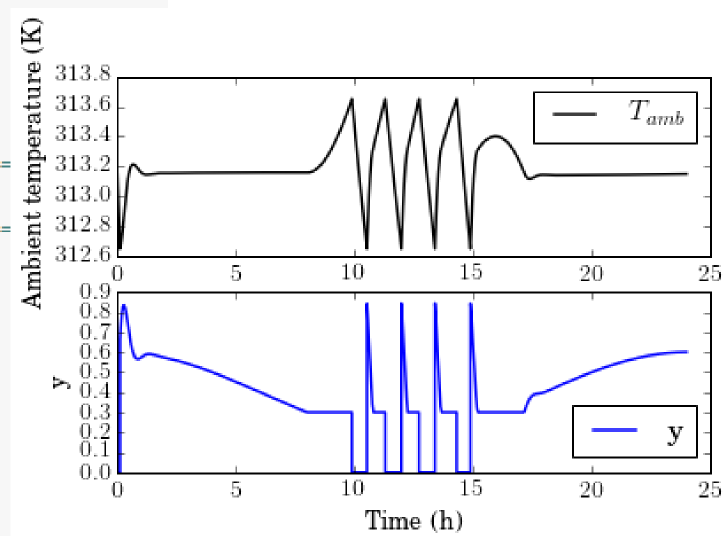


http://nbviewer.jupyter.org/github/ibpsa/project1/tree/master/meetings/2016-10-24-gensim/thursday/

```python
#==============================================================================
# Loading the model file
#==============================================================================

# Set the path to the Buildings library on your drive
ppath = 'D:\\path_to_the_buildings_library_on_your_drive'
# Set the path to the model inside the Buildings library
model = 'Buildings.Controls.Continuous.Examples.PIDHysteresis'

# Load the Simulator class from BuildingsPy
import buildingspy.simulate.Simulator as si
s = si.Simulator(model, 'dymola', packagePath = ppath)

#==============================================================================
# Setting up and starting the simulation
#==============================================================================

# Modify some parameter value
s.addParameters({'con.eOn': 0.5})

# Setup and start the simulation
s.setStopTime(86400)
s.printModelAndTime()
s.simulate()

#==============================================================================
# Loading and reading results
#==============================================================================

# Load results which have been saved in a .mat file
from buildingspy.io.outputfile import Reader
r = Reader('PIDHysteresis.mat', 'dymola')

# Assign values from the reader to variables
(t, T) = r.values('temSen.T')
(t, y) = r.values('con.y')
```

Exercise 1 – Single simulation

Load and run the PIDHysteresis model



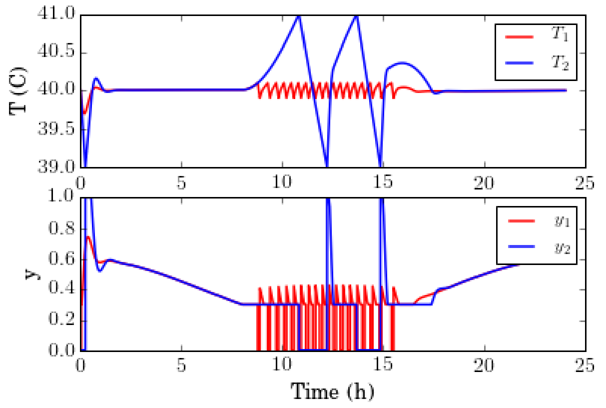
```python
#==============================================================================
# Loading the model file
#==============================================================================

# Set the path to the Buildings library on your disk
ppath = 'D:\\path_to_the_buildings_library_on_your_drive'
# Set the path to the model inside the Buildings library
model = 'Buildings.Controls.Continuous.Examples.PIDHysteresis'

import buildingspy.simulate.Simulator as si

#==============================================================================
# Function to set common parameters and to run the simulation
#==============================================================================

def simulateCase(s):

    s.setStopTime(86400)
    s.showProgressBar(False)
    s.printModelAndTime()
    s.simulate()

#==============================================================================
# Simulate each case
#==============================================================================

# First model
s = si.Simulator(model, 'dymola', 'case1', packagePath = ppath)
s.addParameters({'con.eOn': 0.1})
simulateCase(s)

# second model
s = si.Simulator(model, 'dymola', 'case2', packagePath = ppath)
s.addParameters({'con.eOn': 1})
simulateCase(s)
```

Exercise 2 – Several simulations

2 simulations with different PID offsets

con.eOn = 0.1

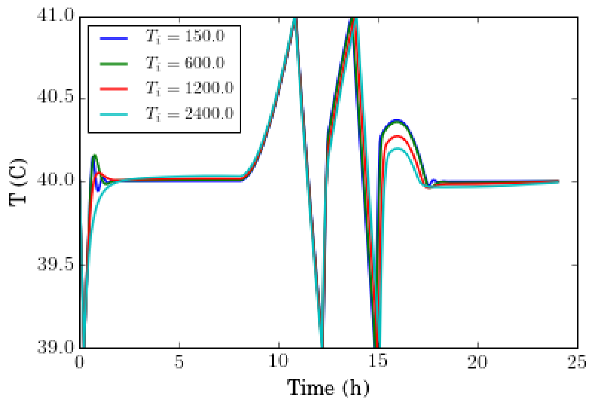
con.eOn = 1



Exercise 3 – Without recompilation

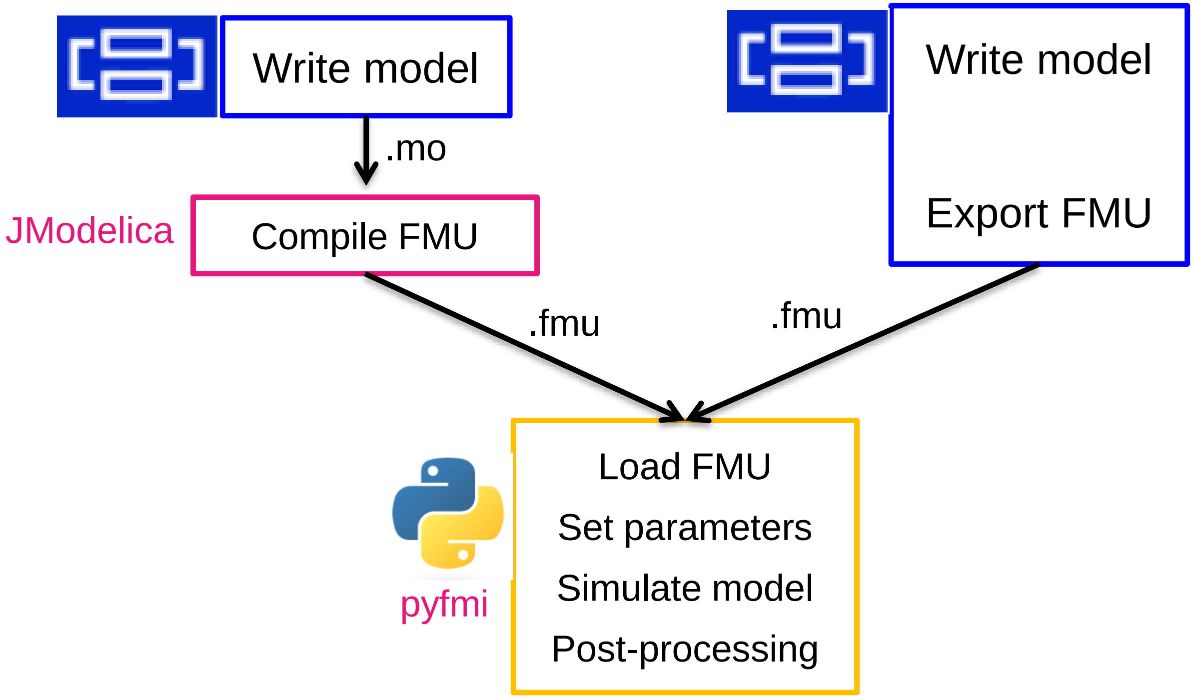
4 simulations with different integrator time constants

```python
#===========================================================================
# Translate the first model
#===========================================================================

import buildingspy.simulate.Simulator as si
s = si.Simulator(model, 'dymola', packagePath = ppath)

s.setSolver('dassl')
s.showGUI(False)
s.setStopTime(86400)
s.setTimeOut(60)
s.translate()

#===========================================================================
# Run a series of simulations with different integrator times
#===========================================================================

from copy import deepcopy
Ti_list = [150, 600, 1200, 2400]

for Ti, i in zip(Ti_list, range(len(Ti_list))):

    s_new = deepcopy(s)

    outputFileName = 'case'+str(i)
    s_new.setOutputDirectory(outputFileName)

    s_new.addParameters({'con.Ti': Ti})

    s_new.printModelAndTime()
    s_new.simulate_translated()
```




```python
#===========================================================================
# Read and plot results
#===========================================================================

from buildingspy.io.outputfile import Reader
import os
import shutil
import matplotlib.pyplot as plt

from matplotlib import rc
rc('text', usetex=True)
rc('font', family='serif', size = 14)

plt.figure()
for Ti, i in zip(Ti_list, range(len(Ti_list))):

    outputFileName = 'case'+str(i)
    r = Reader(os.path.join(outputFileName,'PIDHysteresis.mat'), 'dymola')

    (t, T) = r.values('temSen.T')
    (t, y) = r.values('con.y')

    label_Ti = '$T_i = %0.1f $' % Ti
    plt.plot(t/3600, T-273.15, linewidth = 1.5, label = label_Ti)

    shutil.rmtree(outputFileName)
```

PyFMI


Write model
.mo
JModelica
Compile FMU
Write model
Export FMU
.fmu
.fmu
pyfmi
Load FMU
Set parameters
Simulate model
Post-processing

LOCIE

UNIVERSITÉ SAVOIE MONT BLANC

http://www.jmodelica.org/page/4924

JModelica.org

Search this site: Search

Forums | Blogs | Users | Developers | Download | About | Assimulo | PyFMI | FMI Library | Log in

Home

PyFMI

PyFMI home | Documentation | Tutorial | Examples | Installation   modules | index

Welcome

PyFMI is a package for loading and interacting with Functional Mock-Up Units (FMUs) both for Model Exchange and Co-Simulation, which are compiled dynamic models compliant with the Functional Mock-Up Interface (FMI), see here for more information.

FMI is a standard that enables tool independent exchange of dynamic models on binary format. Several industrial simulation platforms supports export of FMUs, including, Dymola, JModelica.org, OpenModelica and SimulationX, see here for a complete list. PyFMI offers a Python interface for interacting with FMUs and enables for example loading of FMU models, setting of model parameters and evaluation of model equations.

PyFMI is available as a stand-alone package or as part of the JModelica.org distribution. Using PyFMI together with the Python simulation package Assimulo adds industrial grade simulation capabilities of FMUs to Python.

The latest version is available for download here

Documentation

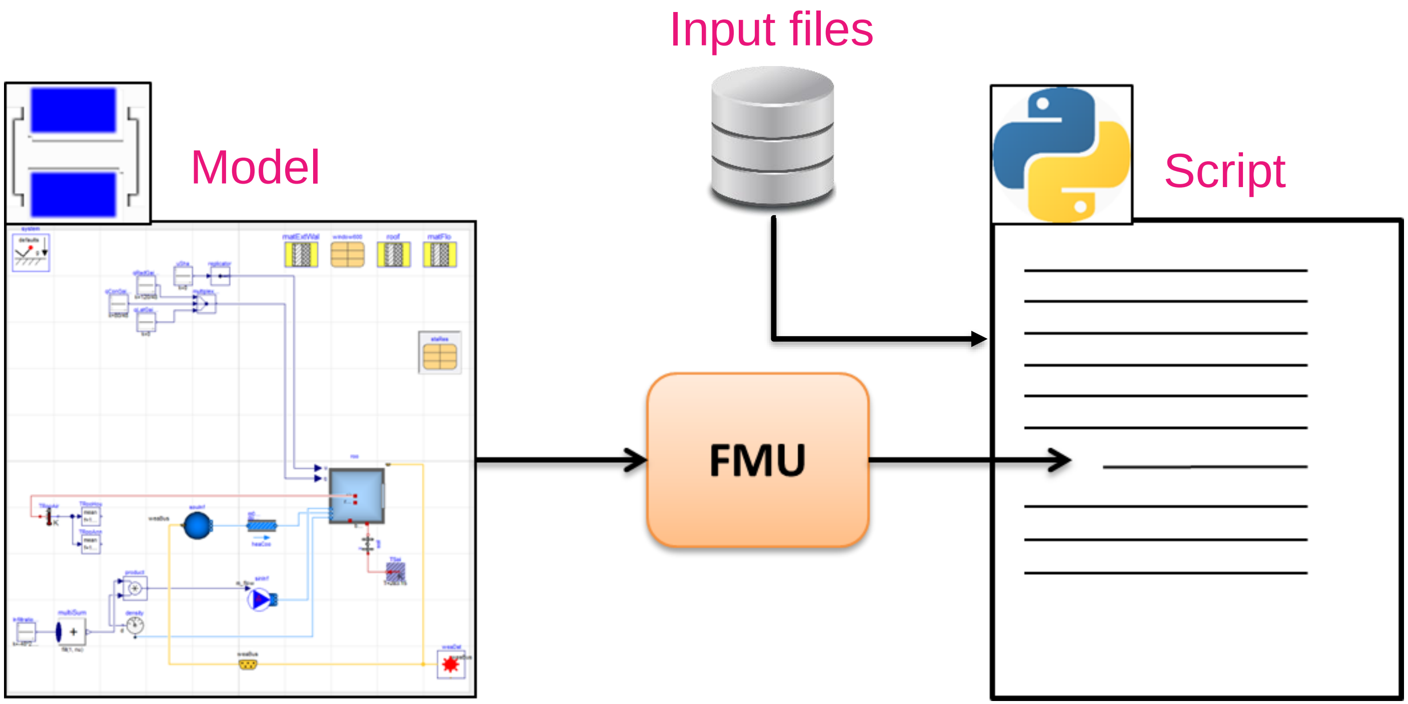
Tutorial
getting started

Examples
view a range of examples

Contents
overview of the

Search page
search the documentation

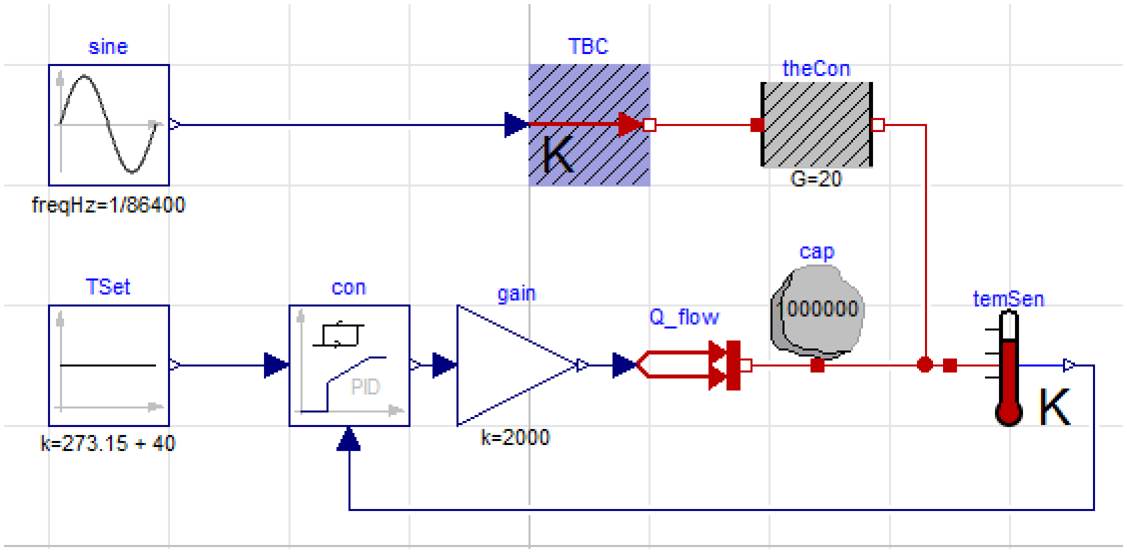
PyFMI

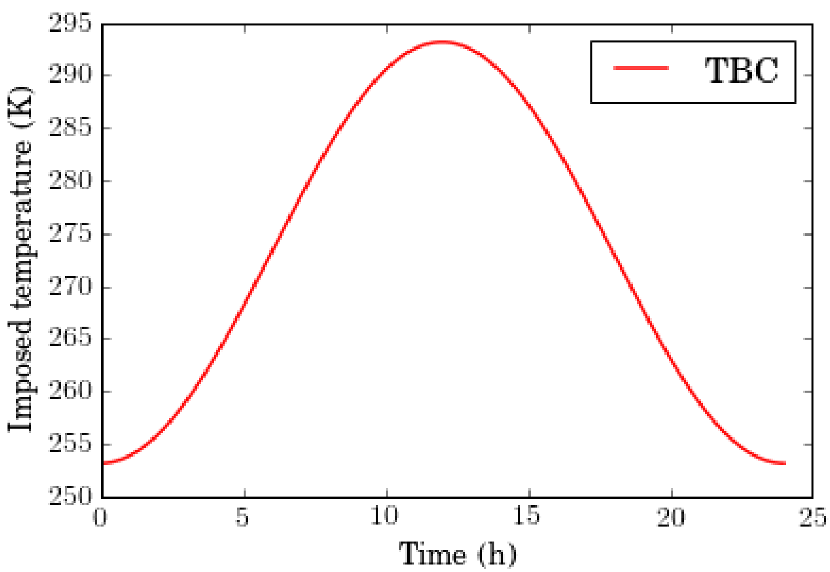


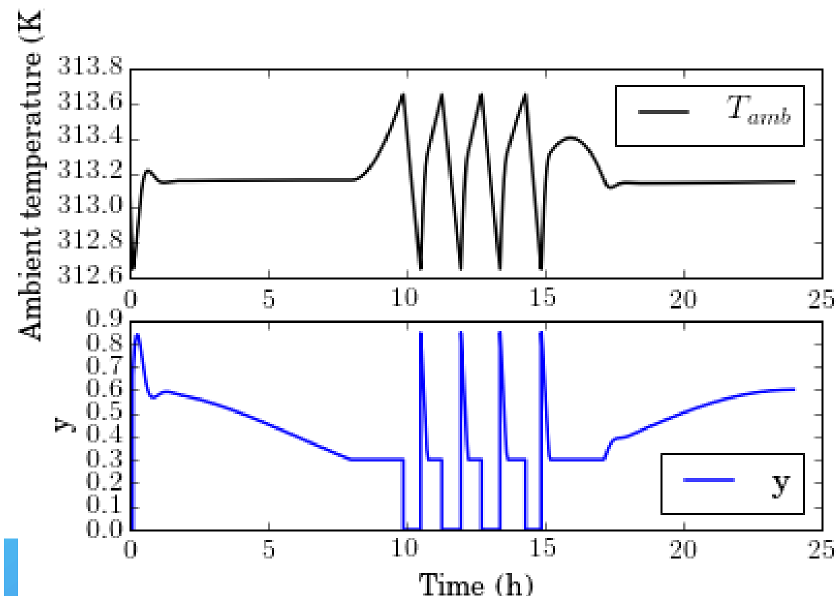
Can read input files outside of the Modelica model
Does not rely on Dymola for the simulation

Only compatible with Python 2.7
Debatable user-friendliness

Buildings.Controls.Continuous.Examples.PIDHysteresis



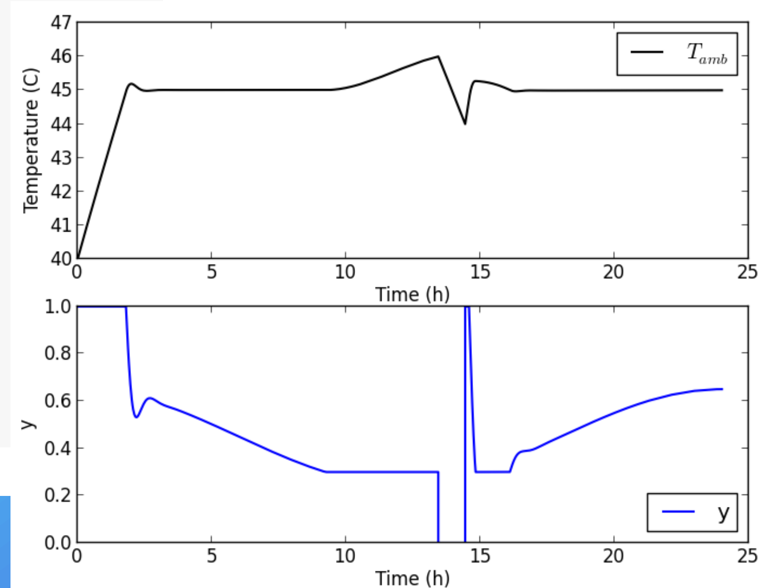




```
#==============================================================================
# Compile the FMU from the .mo file
#==============================================================================
from pymodelica import compile_fmu
ppath = 'C:\\path_to_the_buildings_library_on_your_drive'
model_name = 'Buildings.Controls.Continuous.Examples.PIDHysteresis'
fmu1 = compile_fmu(model_name, ppath)


#==============================================================================
# Loading the FMU
#==============================================================================
# Without JModelica, you can skip the part above and load a .fmu file that
# has been generated by another simulator

from pyfmi import load_fmu
PID = load_fmu('Buildings_Controls_Continuous_Examples_PIDHysteresis.fmu')

# Choice of the time discretisation
tStart = 0
tStop = 3600*24

# Simulation
Tset_init = PID.get('TSet.k')
PID.set('TSet.k', 273.15 + 40)
PID_res = PID.simulate(tStart, tStop)

# Extract output values from the dictionary PID_res
t = PID_res['time']
T = PID_res['temSen.T']
y = PID_res['con.y']
```

Exercise 1 – Single simulation

Load and run the PIDHysteresis model with a different temperature set point



- Save the code in a .py file
- Open the JModelica pylab or IPython console
- Change directory to the one containing your .py file
- Run the code

pylab

```
IPython 0.13.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

Welcome to pylab, a matplotlib-based Python environment [backend: TkAgg].
For more information, type 'help(pylab)'.

In [1]: import os

In [2]: os.chdir('D:\MCF\Simulation\Modelica\ModelicaPython\pyfmi')

In [3]: run JM
JModelica_Ex1_SingleSimulation.py   JModelica_Ex2_SeveralSimulations.py

In [3]: run JModelica_Ex1_SingleSimulation.py
Warning at line 126, column 36, in file 'C:\JModelica.org-1.17\install\ThirdPart
y\MSL\Modelica\Utilities\Streams.mo':
  String arguments in functions are only partially supported

Final Run Statistics: Buildings.Controls.Continuous.Examples.PIDHysteresis

 Number of steps                                      : 352
 Number of function evaluations                       : 594
 Number of Jacobian evaluations                       : 20
 Number of function eval. due to Jacobian eval.       : 40
 Number of error test failures                        : 32
 Number of nonlinear iterations                       : 529
 Number of nonlinear convergence failures             : 0
 Number of state function evaluations                 : 487
 Number of state events                               : 15

Solver options:

 Solver                   : CVode
 Linear multistep method  : BDF
 Nonlinear solver         : Newton
 Linear solver type       : DENSE
 Maximal order            : 5
 Tolerances (absolute)    : [  1.00000000e-06   3.00000000e-04]
 Tolerances (relative)    : 0.0001

Simulation interval    : 0.0 - 86400.0 seconds.
Elapsed simulation time: 0.201652935989 seconds.

In [4]: _
```

Exercise 2 - optimisation

Find which value of the PID gain results in the smallest temperature quadratic error