**PyFMI instructions**

**Download and install the Python PyFMI package**

* PyFMI package currently can only be used in a Windows machine. Using PyFMI package in a linux environment is not impossible, but it is very difficult to achieve
* First, make sure you have Anaconda (<https://docs.anaconda.com/free/anaconda/install/index.html>) installed
* Use Anaconda to open a terminal window
* Type the following to create a Python environment named “pyfmi-env” (or any other name you desired)
  + conda create --name pyfmi-env
* Then type the following to install the PyFMI package:
  + conda install -c conda-forge pyfmi
* After the PyFMI package is installed, and every time you use the PyFMI package, you need to first activate the PyFMI environment
  + conda activate pyfmi-env
* After activating the pyfmi environment, you can now edit Python scripts that use the PyFMI package.
* Once your work with the PyFMI package is finished, you can deactivate the PyFMI environment
  + conda deactivate
* For your reference, the github page for the PyFMI package is here:
  + <https://github.com/modelon-community/PyFMI>

**Files you will receive**

* R32HpTesTest\_parallel.fmu
  + The functional mockup unit (FMU) file, which contains information of the heat pump system model in binary format
* closed\_loop\_simulation.ipynb
  + This is the jupyter notebook file that is used to perform a simulation with the FMU file
* closed\_loop\_simulation.py
  + This is the Python file that is used to perform a simulation with the FMU file. It has the same content as the “closed\_loop\_simulation.ipynb” file.
* MAT\_to\_CSV\_conversion.py
  + This file is used to convert .mat simulation output files to .csv files. This file is called within the “Closed\_loop\_simulation.ipynb” or “closed\_loop\_simulation.py” file. This file doesn’t need to be changed.
* R32HpTesTest\_parallel\_data\_points.csv
  + This file contains information about how to convert the .mat simulation output files to .csv files, such as information about point names and unit conversions. This file is called from the “Closed\_loop\_simulation.ipynb” file or the “closed\_loop\_simulation.py” file alongside the “MAT\_to\_CSV\_conversion.py” file. This file doesn’t need to be changed.
* R32HpTesTest\_parallel\_data\_points\_description
  + A Google Spreadsheet containing descriptions of what each input and output variable means
* R32HpTesTest\_parallel\_result.csv
  + The simulation result file in .csv format. You won’t see this file now because this file is generated only after you have run a simulation using the “closed\_loop\_simulation.ipynb” or the “closed\_loop\_simulation.py” file.

**How you can edit the “closed\_loop\_simulation.ipynb” or “closed\_loop\_simulation.py” file to perform a closed loop simulation**

* We will first encounter 4 variables (The unit is always seconds): start\_time, final\_time, control\_timestep, data\_interval.
  + “start\_time” is the beginning of a full closed loop simulation.
  + “end\_time” is the end of a full closed loop simulation.
  + “control\_timestep” is the elapsed time for each control action. For example, if you want to set up a model predictive control (MPC) algorithm to this heat pump hybrid plant and run the MPC algorithm every 5 minutes, then you want to set the “control\_timestep” to be 300 seconds.
  + “data\_interval” is the data point time interval that you want the simulation output “.csv” file to have. Usually set to 60 seconds.
* For each “control\_timestep”, the variable “current\_control\_timestep\_start\_time” is the start time of this “control\_timestep”. The variable “current\_control\_timestep\_final\_time” is the end time of this “control\_timestep”. The relationship is “current\_control\_timestep\_start\_time” + “control\_timestep” = “current\_control\_timestep\_final\_time”.
* The "inputs\_dict\_at\_\*" variables and the "inputs" variable are used to define the inputs to the FMU model at each “control\_timestep”. There are 3 different ways to define the "inputs" variable within each “control\_timestep”. Each way is shown on Table 1 to Table 3 below. Note from Table 3 that since the "inputs" variable is defined linearly by default, doing step-wise change requires a bit more work.

Table 1: An implementation of constant “ZoneHeatingLoad” input variable within each “control\_timestep”.

| Time | Value of the “ZoneHeatingLoad” input variable |
| --- | --- |
| current\_control\_timestep\_start\_time | 500 |
| Current\_control\_timestep\_end\_time | 500 |

Table 2: An implementation of linear variation of the “ZoneHeatingLoad” input variable within each “control\_timestep”.

| Time | Value of the “ZoneHeatingLoad” input variable |
| --- | --- |
| current\_control\_timestep\_start\_time | 500 |
| Current\_control\_timestep\_start\_time + control\_timestep\*0.2 | 750 |
| Current\_control\_timestep\_start\_time + control\_timestep\*0.6 | 1000 |
| Current\_control\_timestep\_end\_time | 750 |

Table 3: An implementation of step-wise change of the “ZoneHeatingLoad” input variable within each “control\_timestep”.

| Time | Value of the “ZoneHeatingLoad” input variable |
| --- | --- |
| current\_control\_timestep\_start\_time | 500 |
| Current\_control\_timestep\_start\_time + control\_timestep\*0.2 | 500 |
| Current\_control\_timestep\_start\_time + control\_timestep\*0.2 | 750 |
| Current\_control\_timestep\_start\_time + control\_timestep\*0.6 | 750 |
| Current\_control\_timestep\_start\_time + control\_timestep\*0.6 | 1000 |
| Current\_control\_timestep\_end\_time | 1000 |
| Current\_control\_timestep\_end\_time | 750 |

* The 'output\_variables\_dict' variable is used to store the simulation outputs at the end of each control timestep. If you want to apply external control to this simulation, you could develop your own control algorithm to use the 'output\_variables\_dict' variable to control the 'inputs' variable. Notice that I didn’t write the code to “close the loop” on this close loop simulation. I will leave it to you to decide how to modify this “closed\_loop\_simulation.ipynb” or “closed\_loop\_simulation.py” file to loop the 'output\_variables\_dict' variable to control (change the value of) the 'inputs' variable

**Other information you need to know / troubleshooting instructions**

* For the “SysMode” parameter
  + Mode 1: heat pump serve load; Mode 2: heat pump serve PCM; Mode 3: PCM serve load; Mode 2 and 3 cannot be happening together! If at mode 2 while the zone calls for heating, switch to mode 1 or mode 3
* For the “HeatPumpWaterSupplyMassFlow” parameter
  + supply water mass flow of heat pump to the load in Mode 1, and supply water mass flow of heat pump to the PCM in Mode 2. This input needs to be set to 0 during Mode 3, otherwise the FMU model will return a simulation error
* For the “HeatPumpOnOff” parameter
  + 1= on; 0=off; note that heat pump must turn off during Mode 3, otherwise FMU will have simulation errors
* For the “PCMPumpWaterSupplyMassFlow” parameter
  + Pump Water Supply Mass Flow to the PCM during Mode 3. This input needs to be set to 0 during Mode 1 and Mode 2, otherwise the FMU will return a simulation error
* Note that pyfmi is always using SI units. This means that temperature is in units of Kelvin, and not Celsius or Fahrenheit.
* If at any point, you received an error message saying that one of the Python packages is not found, you could go to your Anaconda terminal (make sure you have run “conda activate pyfmi-env” already), and then type the following to install the Python package
  + pip install your\_package
* If you have other questions or run into any problems, don’t hesitate to reach out to us!